Parameterized Complexities of Dominating and Independent Set Reconfiguration

Hans L. Bodlaender
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Carla Groenland
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Céline M. F. Swennenhuis
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Abstract

We settle the parameterized complexities of several variants of independent set reconfiguration and dominating set reconfiguration, parameterized by the number of tokens. We show that both problems are XL-complete when there is no limit on the number of moves and XNL-complete when a maximum length ℓ for the sequence is given in binary in the input. The problems are known to be XNLP-complete when ℓ is given in unary instead, and W[1]- and W[2]-hard respectively when ℓ is also a parameter. We complete the picture by showing membership in those classes.

Moreover, we show that for all the variants that we consider, token sliding and token jumping are equivalent under pl-reductions. We introduce partitioned variants of token jumping and token sliding, and give pl-reductions between the four variants that have precise control over the number of tokens and the length of the reconfiguration sequence.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, independent set reconfiguration, dominating set reconfiguration, W-hierarchy, XL, XNL, XNLP

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.9

Related Version *Full Version*: https://arxiv.org/abs/2106.15907 [2]

Funding *Carla Groenland*: Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research innovation programme (grant agreement No 853234).
Céline M. F. Swennenhuis: Supported by the Netherlands Organization for Scientific Research under project no. 613.009.031b.

Acknowledgements We would like to thank the referees for useful comments.

1 Introduction

In this paper, we study the parameterized complexity of reconfiguration of independent sets, and of dominating sets, with the sizes of the sets as parameter. Interestingly, the complexity varies depending on the assumptions on the length of the reconfiguration sequence, which can be unbounded, given in binary, given in unary, or given as second parameter. One can study the reconfiguration problems for different reconfiguration rules; we will show equivalence regarding the complexity for several reconfiguration rules.

Independent Set Reconfiguration

In the Independent Set Reconfiguration problem, we are given a graph and two independent sets A and B, and wish to decide we can "reconfigure" A to B via a "valid" sequence of independent sets $A, I_1, \dots, I_{\ell-1}, B$. Suppose that we represent the current independent set by placing a token on each vertex. We can move between two independent sets by moving a single token. We consider two well-studied rules for deciding how we can move the tokens.

- Token jumping (TJ): we can "jump" a single token to any vertex that does not yet contain a token.
- Token sliding (TS): we can "slide" a single token to an adjacent vertex that does not yet contain a token.

Independent Set Reconfiguration is PSPACE-complete for both rules [9], but their complexities may be different when restricting to specific graph classes. For example, Independent Set Reconfiguration is NP-complete on bipartite graphs under the token jumping rule, but remains PSPACE-complete under the token sliding rule [11].

There is a third rule which has been widely studied, called the token addition-removal rule, but this rule is equivalent to the token jumping rule for our purposes (see e.g. [9, Theorem 1]). As further explained later, we show that the token jumping and the token sliding rule are also equivalent in some sense (which is much weaker but does allow us to control all the parameters that we care about). We will therefore not explicitly mention the specific rule under consideration below.

Throughout this paper, our reconfiguration problems are parameterized by the number of tokens (the size of the independent set). Independent Set Reconfiguration is W[1]-hard [7], but the problem is not known to be in W[1]. We show that in fact it is complete for the class XL, consisting of the parameterized problems that can be solved by a deterministic algorithm that uses $f(k) \log n$ space, where k is the parameter, n the input size and f any computable function.

▶ **Theorem 1.** *Independent Set Reconfiguration is* XL*-complete.*

In the Timed Independent Set Reconfiguration, we are given an integer ℓ in unary and two independent sets A and B in a graph G, and need to decide whether there is a reconfiguration sequence from A to B of length at most ℓ. We again parameterize it by the number of tokens. The following result has been shown by the authors and Nederlof [1].

▶ **Theorem 2** ([1]). *Timed Independent Set Reconfiguration is* XNLP*-complete.*

The class XNLP (also denoted $N[f \text{ poly}, f \log]$ by Elberfeld et al. [5]) is the class of parameterized problems that can be solved with a non-deterministic algorithm with simultaneously, the running time bounded by $f(k)n^c$ and the space usage bounded by $f(k) \log n$, with k the parameter, n the input size, c a constant, and f a computable function. This is a natural subclass of the class XNL, which consists of the parameterized problems that can be solved by a nondeterministic algorithm that uses $f(k) \log n$ space. Amongst others, XNL was studied by Chen et al. [3].

The classes XL, XNL, XSL, XP can be seen as the parameterized counterparts of L, NL, SL, P respectively. Although no explicit time bound is given, we can freely add a time bound of $2^{f(k) \log n}$, and thus XNL is a subset of XP. We remark that XL=XSL[1] [15] (just as L=SL), XL⊆XNL and XNLP⊆XNL.

[1] A proof can be found in Appendix A of the full version [2]

Table 1 The table shows the parameterized complexities of the independent set and dominating set reconfiguration problems, parameterized by the number of tokens, depending on the treatment of the bound ℓ on the length of the reconfiguration sequence.

Sequence length ℓ	Independent Set	Dominating Set	Sources
not given	XL-complete	XL-complete	Theorems 1, 5
parameter	W[1]-complete	W[2]-complete	Theorems 4, 5 and [12]
unary input	XNLP-complete	XNLP-complete	[1]
binary input	XNL-complete	XNL-complete	Theorems 3, 5

In Binary Timed Independent Set Reconfiguration, the bound ℓ on the length of the sequence is given in binary[2]. Interestingly, this slight adjustment to Timed Independent Set Reconfiguration is complete for XNL instead.

▶ **Theorem 3.** *Binary Timed Independent Set Reconfiguration is* XNL*-complete.*

Finally, we consider what happens when we consider ℓ to be a parameter instead. If Timed Independent Set Reconfiguration (or equivalently Binary Timed Independent Set Reconfiguration) is parameterized by the size of the independent set and the length of the sequence[3], then it is W[1]-hard [12][4]. We show that in this case, W[1] is the "correct class".

▶ **Theorem 4.** *Timed Independent Set Reconfiguration is in* W[1] *when parameterized by the size of the independent set and the length of the sequence.*

Dominating set reconfiguration

The dominating set reconfiguration problem is similar to the independent set reconfiguration problem, but in this case all sets in the sequence must form a dominating set in the graph. This again gives a PSPACE-complete problem [6]. We define the parameterized problems Dominating Set Reconfiguration, Timed Dominating Set Reconfiguration and Binary Timed Dominating Set Reconfiguration similarly as their independent set counterparts, again parameterized by the number of tokens. Since Dominating Set is W[2]-complete and Independent Set is W[1]-complete (parameterized by "the number of tokens"), it may be expected that the reconfiguration variants also do not have the same parameterized complexity. Indeed, Timed Dominating Set Reconfiguration is W[2]-hard when it is moreover parameterized by the length of the sequence [12].

We show that the picture is otherwise the same as for independent set.

▶ **Theorem 5.** *Dominating Set Reconfiguration is* XL*-complete. Binary Timed Dominating Set Reconfiguration is* XNL*-complete. Timed Dominating Set Reconfiguration is* W[2]*-complete when moreover parameterized by the length of the sequence.*

It was already known that Timed Dominating Set Reconfiguration is XNLP-complete [1]. The proof Theorem 5 can be found in Appendix C of the the full version of this paper [2]. A summary of our results can be found in Table 1.

[2] Giving ℓ in binary implies that it contributes $\log_2(\ell)$ to the size of an instance of Binary Timed Independent Set Reconfiguration.
[3] We can also consider it to be parameterized by the sum of the two parameters.
[4] Mouawad et al. [12] only studied the token jumping variant, but Theorem 6 implies the hardness also holds for token sliding.

Many other types of reconfiguration problems have been studied as well, and we refer the reader to the surveys by Van den Heuvel [14] and Nishimura [13] for further background.

Equivalences between token jumping and token sliding

In Appendix D of the full version [2], we introduce partitioned variants of token sliding and token jumping in which the tokens need to stay within specified token sets. We prove the theorem below by giving reductions from and to the independent set reconfiguration problems (with the four rules: (partitioned) token sliding and (partitioned) token jumping) that control the number of tokens and the length of the reconfiguration sequence. We give similar reductions for the dominating set reconfiguration problems.

▶ **Theorem 6.** *For the following parameterized problems, their variant with the token jumping rule is equivalent under pl-reductions and fpt-reductions to their variant with the token sliding rule:* Independent Set Reconfiguration*,* Timed Independent Set Reconfiguration*,* Binary Independent Set Reconfiguration *and* Timed Independent Set Reconfiguration *when moreover parameterized by the length of the sequence. The same holds for the dominating set variants.*

2 Preliminaries

We write $\mathbf{N}$ for the set of integers $0, 1, 2, \ldots$ and write $[a, b]$ for the set of integers x with $a \leq x \leq b$. All logs in this paper are base 2.

Parameterized reductions

A *parameterized reduction* from a parameterized problem $Q_1 \subseteq \Sigma_1^* \times \mathbf{N}$ to a parameterized problem $Q_2 \subseteq \Sigma_2^* \times \mathbf{N}$ is a function $f : \Sigma_1^* \times \mathbf{N} \to \Sigma_2^* \times \mathbf{N}$, such that the following holds.

1. For all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, $(x, k) \in Q_1$ if and only if $f((x, k)) \in Q_2$.
2. There is a computable function g, such that for all $(x, k) \in \Sigma_1^* \times \mathbf{N}$, if $f((x, k)) = (y, k')$, then $k' \leq g(k)$.

A *parameterized logspace reduction* or *pl-reduction* is a parameterized reduction for which there is an algorithm that computes $f((x, k))$ in space $\mathcal{O}(g(k) + \log n)$, with g a computable function and $n = |x|$ the number of bits to denote x.

Symmetric Turing Machine

A Symmetric Turing Machine (STM) is a Nondeterministic Turing Machine (NTM), where the transitions are symmetric. That means that for any transition, we can also take its inverse back. More formally, a Symmetric Turing Machine with one work tape is a 5-tuple $(\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$, where $\mathcal{S}$ is a finite set of *states*, Σ is the *alphabet*, $\mathcal{T}$ is the set of *transitions*, s_{start} is the *start state* and $\mathcal{A}$ is the set of *accepting states*. A transition $\tau \in \mathcal{T}$ is a tuple of the form (p, Δ, q) describing a transition the STM may take, where $p, q \in \mathcal{S}$ are states and Δ is a *tape triple*. A tape triple is equal to either (ab, δ, cd), where $a, b, c, d \in \Sigma$ and $\delta \in \{-1, 1\}$, or $(a, 0, b)$, where $a, b \in \Sigma$. For example, the transition $(p, (ab, 1, cd), q)$ describes that if the STM is in state p, reads a and b on the current work tape cell and the cell directly right of it, then it can replace a with c, b with d, moving the head to the right and going to state q.

Let $\Delta = (ab, \delta, cd)$ be a state triple, then its inverse is defined as $\Delta^{-1} = (cd, -\delta, ab)$. The inverse of $\Delta = (a, 0, b)$ is defined as $\Delta^{-1} = (b, 0, a)$. By definition of the Symmetric Turing Machine, for any $\tau \in \mathcal{T}$, there is an *inverse* transition $\tau^{-1} \in \mathcal{T}$, i.e. if $\tau = (p, \Delta, q) \in \mathcal{T}$, then $\tau^{-1} = (q, \Delta^{-1}, p) \in \mathcal{T}$.

We say that STM $\mathcal{M}$ *accepts* if there is a computation of $\mathcal{M}$ that ends in an accepting state. We remark that the Turing Machines in this paper do not have an input tape, as it is hidden in the states (see Appendix A of the full version [2]). For a more formal definition of Symmetric Turing Machines we would like to refer to the definition from Louis and Papadimitriou in [10].

Note that we may assume that there is only one accepting state $s_{\mathsf{acc}} \in \mathcal{A}$, by creating this new state s_{acc} and adding a transition to s_{acc} from any original accepting state. We may also assume all transitions to move the tape head to the left or right. This can be accomplished by replacing each transition $\tau = (p, (a, 0, b), q)$ with $2|\Sigma|$ transitions as follows. For all $\sigma \in \Sigma$, we create a new state s_σ and two new transitions $\tau_\sigma^1 = (p, (a\sigma, 1, b\sigma), s_\sigma)$ and $\tau_\sigma^2 = (s_\sigma, (b\sigma, -1, b\sigma), q)$.

The following problem will be used in the reductions of Section 3.

Accepting Log-Space Symmetric Turing Machine
Given: A STM $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ with $\Sigma = [1, n]$ and a work tape with k cells.
Parameter: k.
Question: Does $\mathcal{M}$ accept?

We define Accepting Log-Space Nondeterministic Turing Machine to be the Nondeterministic Turing Machine analogue of Accepting Log-Space Symmetric Turing Machine.

▶ **Theorem 7.** Accepting Log-Space Symmetric Turing Machine *is* XL*-complete and* Accepting Log-Space Nondeterministic Turing Machine *is* XNL*-complete.*

A proof of Theorem 7 can be found in Appendix A of the full version [2].

In our reductions we use the notion of a *configuration*, describing exactly in what state an NTM (and therefore an STM) and its tape are.

▶ **Definition 8.** *Let $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ be an NTM with $\Sigma = [1, n]$ and k cells on the work tape and let $\alpha \in \Sigma^*$ be the input. A* configuration *of $\mathcal{M}$ is a $k+2$ tuple $(p, i, \sigma_1, \dots, \sigma_k)$ where $p \in \mathcal{S}$, $i \in [1, k]$ and $\sigma_1, \dots, \sigma_k \in \Sigma$, describing the state, head position and content of the work tape of $\mathcal{M}$ respectively.*

3 Proof of Theorem 1: XL-completeness

By Theorem 6, it suffices to show that the following problem is XL-complete.

Partitioned TS-Independent Set Reconfiguration
Given: Graph $G = (V, E)$; independent sets $I_{\mathsf{init}}, I_{\mathsf{fin}}$ of size k; a partition $V = \sqcup_{i=1}^k P_i$ of the vertex set.
Parameter: k.
Question: Does there exist a sequence $I_{\mathsf{init}} = I_0, I_1, \dots, I_T = I_{\mathsf{fin}}$ of independent sets of size k for some T, with $|I_t \cap P_i| = 1$ for all $t \in [0, T]$ and $i \in [1, k]$, such that for all $t \in [1, T]$, $I_t = I_{t-1} \setminus \{u\} \cup \{v\}$ for some $uv \in E(G)$ with $u \in I_{t-1}$ and $v \notin I_{t-1}$?

Theorem 6 then implies the XL-completeness results for the other variants of Independent Set Reconfiguration.

The XL-completeness proof for Partitioned TS-Dominating Set Reconfiguration is similar and given in Appendix C of the full version [2].

▶ **Theorem 9.** Partitioned TS-Independent Set Reconfiguration *is* XL*-complete.*

Proof. By Theorem 7, it suffices to give pl-reductions to and from Accepting Log-Space Symmetric Turing Machine.

The problem is in XL (=XSL) as it can be simulated with a Symmetric Turing Machine with $\mathcal{O}(k \log n)$ space as follows. We store the current independent set of size k on the work tape, which takes about $k \cdot \log n$ bits space. We use the transitions of the STM to model the changes of one of the vertices in the independent set. For all vertices $u, v \in V$, we have a sequence of states and transitions that allows you to remove u and add v to the independent set currently stored on the work tape, if the following assumptions are met: $u \in I$, $v \notin I$, $uv \in E(G)$, u, v are part of the same token set (part of the partition) and I' is an independent set. This gives a total of $O(n^2k^2)$ states. There is one accepting state, reachable via a sequence of states and transitions that verifies that the current independent set is the final independent set. All transitions are symmetric.

We prove the problem to be XL-hard by giving a reduction from Accepting Log-Space Symmetric Turing Machine. Let $\mathcal{M} = (\mathcal{S}, \Sigma, \mathcal{T}, s_{\mathsf{start}}, \mathcal{A})$ be the STM of a given instance, with $\mathcal{A} = \{s_{\mathsf{acc}}\}$, $\Sigma = [1, n]$ and a work tape of k cells. We may assume that $\mathcal{M}$ only accepts if the symbol 1 is on every cell of the work tape and the head is at the first position. This can be done by creating a new accepting state and adding $\mathcal{O}(k)$ transitions from s_{acc} to this new state, which set only 1's on the work tape and move the head to the first position. We create an instance Γ of Partitioned TS-Independent Set Reconfiguration with $k' = k + 1$ tokens. These tokens will simulate the configuration of $\mathcal{M}$: k tape-tokens modeling the work tape cells and one state token describing the current state and tape head position.

Tape gadgets. For each work tape cell $i \in [1, k]$, we create a *tape gadget* consisting of $n + 1$ vertices as follows. We add a vertex v^i_σ for all $\sigma \in \Sigma = [1, n]$ and a vertex y^i, connected to v^i_σ for all $\sigma \in \Sigma$. The vertices in a tape gadget form a token set (a part of the partition, i.e. exactly one of these $n + 1$ vertices is in the independent set at any given time). The symbol σ that is on the ith work tape cell of $\mathcal{M}$ is simulated by which v^i_σ is in the independent set.

State vertices. The last token set is the set of all transition and state vertices (defined below), meaning that exactly one of these vertices is in the independent set at any given time. The token of this set (called the "state token") simulates the state of $\mathcal{M}$, the position of the head, and the transition $\mathcal{M}$ takes.

We create a vertex p^i for each state $p \in \mathcal{S}$ and all head positions $i \in [1, k]$. We add edges $p^i y^{i'}$ for all $i' \in [1, k]$. These vertices will simulate the current state of $\mathcal{M}$ and the position i of the tape head.

Transition vertices. To go from one state vertex to another, we create a path of three transition vertices, to checking whether the work tape agrees with the transition before and afterwards, and one allowing moving some tokens of the tape gadget. To control when we can move tokens in the tape gadgets, we put edges between y^i and all state and transition vertices (for each $i \in [1, k]$), unless specified otherwise. We further outline which edges are present below, and give an example in Figure 1.

Recall that we may assume that head always moves left or right. Consider first a transition $\tau \in \mathcal{T}$ that moves the head to the right, say $\tau = (p, (ab, 1, cd), q)$. For all $i \in [1, k - 1]$, we create a path between state vertices p^i and q^{i+1} consisting of three "transition" vertices: τ^i_{ab}, τ^i_{shift} and τ^i_{cd}. In order to ensure a token can only be on τ^i_{ab} when the token on the ith tape gadget represents the symbol a, we add edges from τ^i_{ab} to v^i_σ for all $\sigma \in \Sigma \setminus \{a\}$. Similarly,

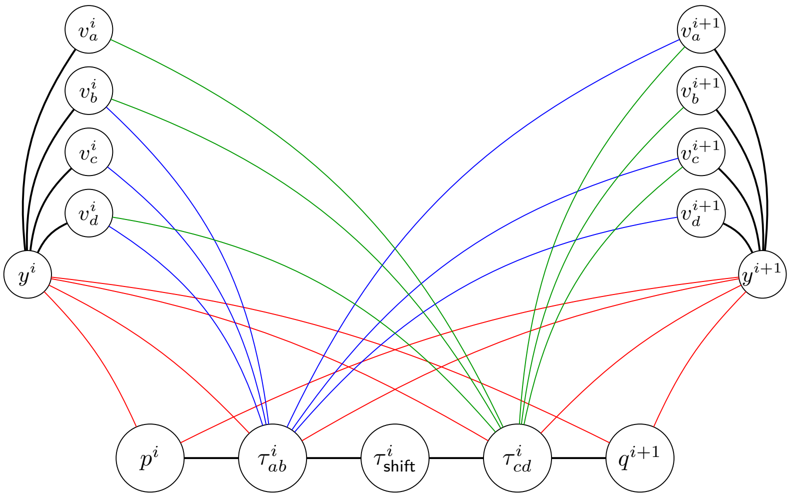

Figure 1 Sketch of part of the construction of Theorem 9. Given are the two tape gadgets for positions i and $i+1$, two state vertices and a transition path for transition $\tau = (p, (ab, 1, cd), q)$.

edges to v_σ^{i+1} are added for all $\sigma \in \Sigma \setminus \{b\}$, as well as edges between τ_{cd}^i and all v_σ^i and v_σ^{i+1} except for v_c^i and v_d^{i+1}. When the token is on the shift vertex τ_{shift}^i, the independent set is allowed to change the token in the ith and $(i+1)$th tape gadget. Therefore, we remove the edges between y^i and y^{i+1} and τ_{shift}^i.

Note that the constructed paths also handle transitions which move the tape head to the left, as we can transverse the constructed paths in both directions. We omit the details.

Initial and final independent sets. Recall that s_{start} is the starting state of $\mathcal{M}$. We let the initial independent set be $I_{\mathsf{init}} = \{s_{\mathsf{start}}^1\} \cup \left(\bigcup_{i\in[1,k]}\{v_1^i\}\right)$, corresponding to the initial configuration of $\mathcal{M}$. Let the final independent set be $I_{\mathsf{fin}} = \{s_{\mathsf{acc}}^1\} \cup \left(\bigcup_{i\in[1,k]}\{v_1^i\}\right)$, where s_{acc} is the accepting state of $\mathcal{M}$. We note that both I_{init} and I_{fin} are independent sets.

Let Γ be the created instance of Partitioned TS-Independent Set Reconfiguration. We prove that Γ is a yes-instance is and only if $\mathcal{M}$ accepts. We use the following function, hinting at the equivalence between configurations of $\mathcal{M}$ and certain independent sets of Γ.

▶ **Definition 10.** *Let $C = (p, i, \sigma_1, \ldots, \sigma_k)$ be a configuration of $\mathcal{M}$. Then let $I(C)$ be the corresponding independent set in Γ:*

$$I(C) = \{p^i\} \cup \left(\bigcup_{j=1}^{k} \left\{v_{\sigma_j}^j\right\}\right)$$

▷ Claim 11. Γ is a yes-instance if $\mathcal{M}$ accepts.

Proof. Let $C_1, \ldots, C_\ell$ be the sequence of configurations such that $\mathcal{M}$ accepts. We note that $I(C_1) = I_{\mathsf{init}}$. For each $t \in [1, \ell-1]$, we do the following. Let $C_t = (p, i, \sigma_1, \ldots, \sigma_k)$. Let τ be the transition $\mathcal{M}$ takes to the next configuration. Assume $\tau = (p, (ab, 1, cd), q)$, the case where τ moves the head to the left will be discussed later, but is similar. Take the following sequence of independent sets, where $I(C_t) = I_0$ is the current independent set:

$$I_1 = I_0 \setminus \{p^i\} \cup \{\tau^i_{ab}\} \qquad I_5 = I_4 \setminus \{v^{i+1}_b\} \cup \{y^{i+1}\}$$
$$I_2 = I_1 \setminus \{\tau^i_{ab}\} \cup \{\tau^i_{\mathsf{shift}}\} \qquad I_6 = I_5 \setminus \{y^{i+1}\} \cup \{v^{i+1}_d\}$$
$$I_3 = I_2 \setminus \{v^i_a\} \cup \{y^i\} \qquad I_7 = I_6 \setminus \{\tau^i_{\mathsf{shift}}\} \cup \{\tau^i_{cd}\}$$
$$I_4 = I_3 \setminus \{y^i\} \cup \{v^i_c\} \qquad I_8 = I_7 \setminus \{\tau^i_{cd}\} \cup \{q^{i+1}\}$$

Notice that this sequence of independent sets is allowed, as all sets are independent, each token stays within its token set and each next independent set is a slide away from its previous. Also, we see that $I(C_{t+1}) = I_8$. For transition $\tau = (p, (ab, -1, cd), q)$, where the tape head moves to the left, we do the following. Let $\tau^{-1} = (q, (cd, 1, ab), p)$ be the inverse. We take the sequence that belongs to τ^{-1} backwards, i.e. if $I_0, \ldots, I_8$ was the sequence of independent sets as described for τ^{-1}, then take the sequence $I_8, \ldots, I_0$.

We note that $I(C_\ell) = I_{\mathsf{fin}}$ is the final independent set, as we assumed the machine only to accept with $\sigma_i = 1$ for all $i \in [1, k]$ and the head at the first position. Therefore, we find that this created sequence of independent sets is a solution to Γ. ◁

We now prove the other direction.

▷ Claim 12. $\mathcal{M}$ accepts if Γ is a yes-instance.

Proof. Let $I_{\mathsf{init}} = I_1, \ldots, I_{\ell-1}, I_\ell = I_{\mathsf{fin}}$ be the sequence of independent sets that is a solution to Γ. We assume this sequence to be minimal, implying that no independent set can occur twice.

The state token should always be on either a state or transition vertex, because of its token set. Let $I'_1, \ldots, I'_{\ell'}$ be the subsequence of $I_1, \ldots, I_\ell$ of independent sets that include a state vertex. We will prove that the configurations of $\mathcal{M}$, simulated by this subsequence, is a sequence of configurations that leads to the accepting state s_{acc}. To do this, first we note some general facts about I_t for $t \in [1, \ell]$.

If the state token of I_t is on a state vertex p^i, then I_{t+1} slides the state token to a neighbor of p^i. This is because all $y^{i'}$ for $i' \in [1, k]$ are neighbors of p^i, hence the tokens in the tape gadgets are on some $v^{i'}_\sigma$ and cannot move. The same holds for transition vertices of the form τ^i_{ab}. If $\tau^i_{\mathsf{shift}} \in I_t$, then y^i and y^{i+1} are not neighbors of the state token. Therefore, the ith and $i + 1$th tape gadgets token can now slide. If $\tau^i_{ab} \in I_t$, then $v^i_a \in I_t$ and $v^{i+1}_b \in I_t$. This is because all other vertices of the ith and $i + 1$th tape gadgets are neighbors of τ^i_{ab}.

Recall that $I'_1, \ldots, I'_{\ell'}$ is the sequence of independent sets with the state token on a state vertex. For any I'_t with $t \in [1, \ell']$, let C_t be the unique configuration of $\mathcal{M}$ such that $I(C_t) = I'_t$. We prove that $C_1, \ldots, C_{\ell'}$ is an allowed sequence of configurations for $\mathcal{M}$. Note that this implies that $\mathcal{M}$ accepts as $C_{\ell'}$ is the accepting configuration.

We fix $t \in [1, \ell']$ and focus on the transition between C_t and C_{t+1}. Let $A_1, \ldots, A_R$ be the sequence of independent sets in the solution of Γ, that are visited between I'_t and I'_{t+1}. By definition of I'_t and I'_{t+1}, A_r does not contain a state vertex for all $r \in [1, R]$, therefore each A_r must have its state token on a transition vertex. Each such transition vertex corresponds to the same transition $\tau = (p, \Delta, q)$, as this is the only path the state token can take. We assume that $\Delta = (ab, 1, cd)$, the case $\Delta = (ab, -1, cd)$ can be proved with similar arguments. The set A_1 contains transition vertex τ^i_{ab} and therefore I'_t contains v^i_a and v^{i+1}_b. Also, A_R contains τ^i_{cd}, implying that $v^i_c, v^{i+1}_d \in I'_{t+1}$. We note that $A_2, \ldots, A_{R-1}$ must contain τ^i_{shift}: only the ith and $i + 1$th tape gadget tokens can shift when the state token is on τ^i_{shift}. So if the state token would be on τ^i_{ab} or τ^i_{cd} twice in $A_1, \ldots, A_R$, the independent sets would be equal, contradicting the minimal length of the sequence.

Combining this all, we conclude that if $I(C_t) = I'_t$ and $I(C_{t+1}) = I'_{t+1}$, there is an allowed sequence of independent set, traversing the path belonging to a transition $\tau = (p, (ab, 1, cd), q)$. Therefore, $I_{t+1} = I_t \setminus \{v_a^i, v_b^{i+1}, p^i\} \cup \{v_c^i, v_d^{i+1}, q^{i+1}\}$ and we are allowed to take transition τ from C_t to end up in configuration C_{t+1}. ◁

Hence, Γ is a yes instance if and only if $\mathcal{M}$ accepts and we find that the given reduction is correct. This concludes the proof of Theorem 9. ◀

4 Proof of Theorem 3: XNL-completeness

In this section we prove Theorem 3 by showing that the following problem is XNL-complete.

Binary Timed Partitioned TS-Independent Set Reconfiguration
Given: Graph $G = (V, E)$; independent sets $I_{\mathsf{init}}, I_{\mathsf{fin}}$ of size k; integer ℓ given in binary; a partition $V = \sqcup_{i=1}^k P_i$ of the vertex set.
Parameter: k.
Question: Does there exist a sequence $I_{\mathsf{init}} = I_0, I_1, \ldots, I_T = I_{\mathsf{fin}}$ of independent sets of size k with $T \leq \ell$ and $|I_t \cap P_i| = 1$ for all $t \in [0, T]$ and $i \in [1, k]$, such that for all $t \in [1, T]$, $I_t = I_{t-1} \setminus \{u\} \cup \{v\}$ for some $uv \in E(G)$ with $u \in I_{t-1}$ and $v \notin I_{t-1}$?

To prove XNL-hardness, we introduce a variant of CNF-SAT. The following is a "long chain"-variant of the XNLP-complete problems "chained CNF-Satisfiability" introduced by [1].

Long Partitioned Positive Chain Satisfiability
Input: Integers $k, q, r \in \mathbf{N}$ with r given in binary and $r \leq q^k$; Boolean formula F, which is an expression on $2q$ positive variables and in conjunctive normal form; a partition of $[1, q]$ into k parts $P^1, \ldots, P^k$.
Parameter: k.
Question: Do there exist variables $x_j^{(t)}$ for $t \in [1, r]$ and $j \in [1, q]$, such that we can satisfy the formula

$$\bigwedge_{1 \leq t \leq r-1} F(x_1^{(t)}, \ldots, x_q^{(t)}, x_1^{(t+1)}, \ldots, x_q^{(t+1)})$$

by setting, for $i \in [1, k]$ and $t \in [1, r]$, exactly one variable from the set $\{x_j^{(t)} : j \in P_i\}$ to true and all others to false?

We remark that all XNLP-complete "chained satisfiability" variant of [1] have an XNL-complete analogue, but we decided to only present the form we need for this section. In Appendix B of the full version [2], we prove the following result.

▶ **Theorem 13.** Long Partitioned Positive Chain Satisfiability *is* XNL-*complete.*

From this, we derive the following result.

▶ **Theorem 14.** Binary Timed Partitioned TS-Independent Set Reconfiguration *is* XNL-*complete.*

Recall that Theorem 6 then implies the XNL-completeness results for the other variants of Binary Timed Independent Set Reconfiguration. A similar proof for the dominating set variant can be found in Appendix C.2 of the full version [2].

Proof of Theorem 14. We first show that Binary Timed Partitioned TS-Independent Set Reconfiguration is in XNL, that is, it can be modelled by a Nondeterministic Turing Machine using a work tape of size $\mathcal{O}(k \log n)$. One can store the current independent set of size k on the work tape and allow only transitions between an independent set I to an independent set $I' = I \setminus \{v\} \cup \{w\}$ if $vw \in E$, $v \in I$ and $w \notin I$. We can generate the possible independent sets adjacent to a given independent set I and keep track of the number of moves on a work tape of size $\mathcal{O}(k \log n)$. Since the number of independent sets of size k is at most n^k, and a shortest sequence consists of distinct independent sets, we may assume that $\ell \leq n^k$.

To prove that Binary Timed Partitioned TS-Independent Set Reconfiguration is XNL-hard, we give a reduction from Long Partitioned Positive Chain Satisfiability. The construction is similar (but more cumbersome) way as the one in [1, Theorem 4.11].

Let $(q, r, F, P^1, \ldots, P^k)$ be an instance of Long Partitioned Positive Chain Satisfiability. We will create an instance Γ of Binary Timed Partitioned TS-Independent Set Reconfiguration with $3k + 1$ token sets. The idea is to represent the choice of which variables $x_j^{(t)}$ are set to true with variable gadgets, and to create a clause checking gadget that verifies that $F(x_1^{(t)}, \ldots, x_q^{(t)}, x_1^{(t+1)}, \ldots, x_q^{(t+1)})$ is true. The time counter gadget has k tokens, which together represent the integer t. Using the time constraint, we ensure that we have to follow a very specific sequence of moves, and can therefore not change which $x_j^{(t)}$ is true after we passed an independent set that made a choice for this.

Time counter gadget. We create k *time tokens* who have its token set within the time counter gadget, where the positions of these tokens represent an integer $t \in [1, r]$ with $r \leq q^k$. We create k timers, consisting each of $4q$ vertices. For $i \in [1, k]$, the timer t^i is a cycle on vertices $t_0^i, \ldots, t_{4q-1}^i$, which forms a token set for one of the time tokens. If the time tokens are on the vertices $t_{\ell_1}^1, \ldots, t_{\ell_k}^k$, then this represents the current time as

$$t = \sum_{i=1}^{k} (\ell_i \bmod q) q^{i-1}.$$

Henceforth, we will silently assume t to be given by the position of the time tokens as specified above. How these timers are connected such that they work as expected will be discussed later.

Variable gadget. We create four sets $A = \{a_1, \ldots, a_q\}$, $B = \{b_1, \ldots, b_q\}$, $C = \{c_1, \ldots, c_q\}$ and $D = \{d_1, \ldots, d_q\}$ that all contain q vertices. These sets will be used to model which variables $x_j^{(t)}$ are chosen to be true. We partition the sets in the same way as the variables, setting $A^i = \{a_j : j \in P^i\}$ for all $i \in [1, k]$ and defining B^i, C^i and D^i similarly.

For all $i \in [1, k]$, we make (A^i, B^i) and (C^i, D^i) complete bipartite graphs, adding the edges $a_j b_{j'}$ and $c_j d_{j'}$ for all $j, j' \in P^i$. We specify $A^i \cup B^i$ and $C^i \cup D^i$ as token sets, and refer to the corresponding $2k$ tokens as *variable tokens*. The first set is used to model the choice of the true variable $x_j^{(t)}$ for $j \in P^i$ for all odd t, whereas the second partition models the same for any even t.

We will enforce the following. Whenever we check whether all the clauses are satisfied, we will either restrict all tokens of $A \cup B$ to be in A, or restrict all to be in B. Whenever we have to choose a new set of true variables for t odd, we move all tokens from A to B (or the other way around). This movement takes exactly k steps. The same holds for even t and the sets C and D.

Clause checking gadget. The clause checking gadget exists of four parts, called AC, BC, BD and AD, named after which pair of sets we want the variable tokens to be in. All the vertices of the clause checking gadget form a token set, and we refer to the corresponding token as the *clause token*. The token will traverse the gadget parts in the order $AC \to BC \to BD \to AD \to AC \to \dots$. If the token is on AC, we require the variable tokens to be in A and C and we then check whether the clauses hold for the given choice of variables. The other parts are constructed likewise. For an example we refer to Figure 2.

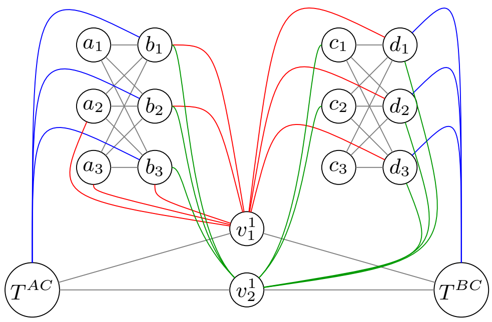

Figure 2 Sketch of part of the construction of Theorem 14. Given are the two variable gadgets for $A^i \cup B^i$ and $C^i \cup D^i$, the AC part of the clause checking gadget with one clause: $C_1 = (x_1^{(t)}, x_3^{(t+1)})$, where t is odd. Hence v_1^1 checks whether a_1 is set to true and v_2^1 checks whether c_3 is set to true.

We now give the construction of this gadget. We create a vertex, T^{AC} that is connected to all $b \in B$. This ensures that if the clause token is on T^{AC}, all tokens from $A \cup B$ are on vertices in A, yet tokens will be able to move from vertices in D to vertices in C.

Suppose $F = C_1 \wedge \dots \wedge C_S$ with each C_i a disjunction of literals. Let $s \in [1, S]$ and let $C_s = y_1 \vee \dots \vee y_{H_s}$ be the sth clause. We create a vertex v_h^s for all $h \in [1, H_s]$. All v_h^s are connected to all vertices in B and D, which ensures that whenever the clause token is on some v_h^s, all variable tokens to be on vertices in A and C and prohibits these variable tokens to move.

Let $h \in [1, H_s]$ and let $j \in [1, q]$ be such that y_h is the jth variable. We ensure that the clause token can only be on v_h^s if the corresponding $x_j^{(t)}$ is modelled as true, that is, the corresponding variable token is on the vertex a_j or c_j (depending on the parity of t). To ensure this, we connect v_h^s to all variables in $A^i \setminus \{a_j\}$ if t is odd and to all variables in $C^i \setminus \{c_j\}$ if t is even, where $i \in [1, k]$ satisfies $j \in P^i$.

We add edges such that $(\{v_h^s\}_{h \in [1, H_s]}, \{v_h^{s+1}\}_{h \in [1, H_{s+1}]})$ forms a complete bipartite graph for all $s \in [1, S-1]$. We connect T^{AC} to all v_h^1 and we connect all v_h^S to T^{BC}, the first vertex of the next gadget. Whenever we move the clause token from T^{AC} to T^{BC}, we have to traverse a vertex v_h^s for each clause C_s, which ensures that the literal y_h in the clause C_s is set to true according to the variable tokens.

The gadget parts for BC, BD and AD are constructed likewise. We omit the details.

Connecting the time counter gadget. We now describe how to connect the vertices in the time counter gadget to those in the clause checking gadget. In the first timer, we create the following edges for $z \in [0, 4q-1]$:

$$
\begin{aligned}
&T^{AC} t_z^1 && \text{when } z \equiv 2 \text{ or } z \equiv 3 \bmod 4,\\
&T^{BC} t_z^1 && \text{when } z \equiv 3 \text{ or } z \equiv 0 \bmod 4,\\
&T^{BD} t_z^1 && \text{when } z \equiv 0 \text{ or } z \equiv 1 \bmod 4,\\
&T^{AD} t_z^1 && \text{when } z \equiv 1 \text{ or } z \equiv 2 \bmod 4.
\end{aligned}
$$

This ensures that we can only move the first time token from t_0^1 to t_1^1 if the clause token is on T^{AC}, and that we cannot put the clause token on T^{BC} before having moved the time token. To enforce that the time token moves when the clause token is at T^{AC}, we add edges between any v_h^s vertex in this path and all t_z^1 with $z \not\equiv 1 \mod 4$. The edges are created in a similar manner for the paths following T^{BC}, T^{BD} and T^{AD}.

When the first time token has made q steps, we allow the second time token to move 1 step forward. For $i \in [2, k]$ we add all edges $t_z^i t_y^{i+1}$, *except* for the following $y, z \in [0, 4q-1]$:

$y \equiv 0 \bmod 4$ and $z \in [0, q]$,

$y \equiv 1 \bmod 4$ and $z \in [q, 2q]$,

$y \equiv 2 \bmod 4$ and $z \in [2q, 3q]$,

$y \equiv 3 \bmod 4$ and $z \in [3q, 4q-1] \cup \{0\}$.

This ensures, for example, that the $(i+1)$th gadget token can move from t_0^{i+1} to t_1^{i+1} if and only if the ith time gadget token is on t_q^i.

Finally, we add two sets V_{init} and V_{fin} of $2k$ vertices, and add the first set to the initial independent set I_{init} and the second to the final independent set I_{fin}. Each vertex of V_{init} is added to the token set of $A^i \cup B^i$ or $C^i \cup D^i$ for some $i \in [1, 2k]$, adding exactly one vertex to each token set, and similarly for V_{fin}.

We create edges uv for all $u \in V_{\mathsf{init}} \cup V_{\mathsf{fin}}$ and v in the clause checking gadget. We also create two vertices c_{init} and c_{fin} that are added to the initial and final independent set respectively, and to the token set of the clause token. We make c_{init} adjacent to T^{AC} and c_{fin} to a vertex T^{XY}, where X, Y depend on the value of r modulo 4.

The vertices c_{init} and c_{fin} are adjacent to all vertices in the time gadgets except for those representing the time 0 and r respectively. The initial independent set also contains the vertices in the time gadget that represent $t = 0$ and similarly I_{fin} contains the vertices that represent r.

Bounding the sequence length. We give a bound ℓ on the length of the reconfiguration sequence, to ensure that only the required moves are made. Before moving the time token, we first move the $2k$ variable tokens into position. We can then move the clause token to T^{AC}, move the first time token so that the time represents 1 and after that take $S+1$ steps to reach T^{BC} (with S the number of clauses in F), at which point we can move the first time token one step forward, and we need to move k variable tokens from A to B. Because we check exactly $r-1$ assignments, we need to move the ith time counter token exactly $\lfloor (r-1)/q^{k-(i-1)} \rfloor$ times. As a last set of moves, we need to move the variable tokens to the set V_{fin}, and the clause token to c_{fin} taking another $2k+1$ steps. Hence, we set the maximum length of the sequence ℓ (from the input of our instance of Binary Timed Partitioned TS-Independent Set Reconfiguration) to

$$4k + 2 + (r-1)(S + k + 1) + \sum_{i=1}^{k} \lfloor (r-1)/q^{k-(i-1)} \rfloor.$$

We claim that there is a satisfying assignment for our instance of Long Partitioned Positive Chain Satisfiability if and only if there is a reconfiguration sequence from I_{init} to I_{fin} of length at most ℓ. It is not too difficult to see that a satisfying assignment leads to a reconfiguration sequence (by moving the variable tokens such that they represent the chosen true variables $x_j^{(t)}$ when the time tokens represent time t).

Vice versa, suppose that there is a reconfiguration sequence of length ℓ. This is only possible if the sequence takes a particular form: we need to move the time tokens for $\sum_{i=1}^{k} \lfloor (r-1)/q^{k-(i-1)} \rfloor$ steps, and can only do this if we can move the clause token $(r-1)(S+1)+2$ steps. The moves of the clause token forces us to move k variable tokens between A and B and between C and D a total of $(r-1)$ times, and we need a further $2k$ moves to get these from V_{init} and to V_{fin}. In particular, there is no room for moving a variable token from one position in A to another position in A, without the "time" having moved 4 places. Therefore, for each $i \in [1,k]$ and $t \in [1,r]$, there is a unique j for which we find a variable token on $a_j \in A^i$, $b_j \in B^i$, $c_j \in C^i$ or $d_j \in D^i$ (which letter a, b, c or d we search for depends on the value of t modulo 4) when the time tokens represent time t. This is the variable $x_j^{(t)}$ that we set to true from the tth variable set in partition P^i. ◀

5 Proof of Theorem 4: W[1]-membership

We formulate Timed TJ-Independent Set Reconfiguration with the number of tokens and length of the reconfiguration sequence as combined parameter as an instance of Weighted 3-CNF-Satisfiability.

Weighted 3-CNF-Satisfiability
Given: Boolean formula F on n variables in conjunctive normal form such that each clause contains at most 3 literals; integer K.
Parameter: K.
Question: Can we satisfy F by setting exactly K variables to true?

This proves Theorem 4 since Weighted 3-CNF-Satisfiability is W[1]-complete [4]. We explain how to adjust it to W[2]-membership for the dominating set variant in Appendix C of the full version [2]; the main idea of our proof can be applied for several other reconfiguration problems (all that is needed is that the property of the solution set can be expressed as a CNF formula).

Proof of Theorem 4. Let $(G=(V,E), I_{\mathsf{init}}, I_{\mathsf{fin}}, k, \ell)$ be an instance of Timed TJ-Independent Set Reconfiguration. We set $C=(k+1+\ell)^2$ and $K=\ell(C+1)+(\ell+1)k$. We add the following variables to our Weighted 3-CNF-Satisfiability instance for all $t \in [0,\ell]$:

- $s_{t,v}$, for each vertex $v \in V$. This should be set to true if and only if v has a token at time t.
- $m^{(i)}_{t,v,w}$, for each pair of distinct vertices $v, w \in V$ and for all $i \in [1,C]$. This should be set to true if and only if we move a token from v to w from time $t-1$ to time t.
- $m^{(i)}_{t,\emptyset}$, for all $i \in [1,C]$. This is set to true if no token is moved at from time $t-1$ to time t.
- $a_{t,v}$ for all $v \in V$. This is set to true if and only if v received a token from time $t-1$ to time t.
- $a_{t,\emptyset}$. This is set to true if no vertex received a token from time $t-1$ to time t.

We add clauses that are satisfied if and only if the set of true variables corresponds to a correct TJ-reconfiguration sequence from I_{init} to I_{fin}.

- We have clauses with one literal that ensure that at time 0, we have the initial configuration: for each $v \in I_{\mathsf{init}}$, we have a clause $s_{0,v}$ and for each $v \notin I_{\mathsf{init}}$, we have a clause $\neg s_{0,v}$.
- Similarly, we have clauses that ensure that at time ℓ, we have the final configuration: for each $v \in I_{\mathsf{fin}}$, we have a clause $s_{\ell,v}$ and for each $v \notin I_{\mathsf{fin}}$, we have a clause $\neg s_{\ell,v}$.

- All $m^{(i)}_{t,\star}$ are equivalent: for all distinct $i, j \in [1, C]$, for all $t \in [1, \ell]$ and for all distinct $v, w \in V$, we add the clauses $\neg m^{(i)}_{t,v,w} \vee m^{(j)}_{t,v,w}$ and $m^{(i)}_{t,v,w} \vee \neg m^{(j)}_{t,v,w}$. For all distinct $i, j \in [1, C]$, for all $t \in [1, \ell]$, we add the clauses $\neg m^{(i)}_{t,\emptyset} \vee m^{(j)}_{t,\emptyset}$ and $m^{(i)}_{t,\emptyset} \vee \neg m^{(j)}_{t,\emptyset}$.
- We have clauses that ensure that at each time $t \in [1, \ell]$, at most one move is selected: for any two distinct pairs of distinct vertices (v, w) and (v', w'), we add the clauses $\neg m^{(1)}_{t,v,w} \vee \neg m^{(1)}_{t,v',w'}$ and $\neg m^{(1)}_{t,v,w} \vee \neg m^{(1)}_{t,\emptyset}$.
- For $t \in [1, \ell]$, if the move $m^{(1)}_{t,v,w}$ is selected, then v lost a token and w obtained a token from time $t - 1$ to time t: $\neg m^{(1)}_{t,v,w} \vee s_{t-1,v}$, $\neg m^{(1)}_{t,v,w} \vee \neg s_{t-1,w}$, $\neg m^{(1)}_{t,v,w} \vee \neg s_{t,v}$ and $\neg m^{(1)}_{t,v,w} \vee s_{t,w}$.
- For $t \in [1, \ell]$, tokens on vertices not involved in the move remain in place. For all distinct $v, w, u \in V$, we add the clauses

$$\neg m^{(1)}_{t,\emptyset} \vee \neg s_{t-1,v} \vee s_{t,v},$$
$$\neg m^{(1)}_{t,\emptyset} \vee s_{t-1,v} \vee \neg s_{t,v},$$
$$\neg m^{(1)}_{t,v,w} \vee \neg s_{t-1,u} \vee s_{t,u} \text{ and}$$
$$\neg m^{(1)}_{t,v,w} \vee s_{t-1,u} \vee \neg s_{t,u}.$$

- We record if a token was added to a vertex: for all $t \in [1, \ell]$ and $v \in V$, we add the clause $s_{t-1,v} \vee \neg s_{t,v} \vee a_{t,v}$. This in particular ensures that $a_{t,v}$ is true when $m^{(1)}_{t,v,w}$ is true for some vertex $w \neq v$.
- No move happened if and only if no token was added: for all $t \in [1, \ell]$ we add the clauses $\neg m^{(1)}_{t,\emptyset} \vee a_{t,\emptyset}$ and $\neg a_{t,\emptyset} \vee m^{(1)}_{t,\emptyset}$.
- At most one $a_{t,\star}$ is set to true, implying that at most one taken gets added at each time step: for all $t \in [1, \ell]$ and distinct $v, w \in V$, we add the clauses $\neg a_{t,v} \vee \neg a_{t,w}$ and $\neg a_{t,v} \vee \neg a_{t,\emptyset}$.
- Finally, we check whether the current set forms an independent set: for all edges $vw \in E(G)$ and $t \in [0, \ell]$, we add the clause $\neg s_{t,v} \vee \neg s_{t,w}$.

If there is a TJ-independent set reconfiguration sequence $I_{\mathsf{init}} = I_0, \dots, I_T = I_{\mathsf{fin}}$ with $T \leq \ell$, then we set $s_{t,v}$ to true if and only if $v \in I_t$ for $t \in [0, T]$. For all $t \in [T, \ell]$, we set $s_{t,v}$ to true if and only if $v \in I_T$.

Let $t \in [1, \ell]$. If $I_t = I_{t-1}$, we set $a_{t,\emptyset}$ to true and $m^{(i)}_{t,\emptyset}$ to true for all $i \in [1, C]$. Otherwise, we find $I_t = I_{t-1} \setminus \{v\} \cup \{w\}$ for some $v, w \in V$ and we set $m^{(i)}_{t,v,w}$ and $a_{t,v}$ to true for all $i \in [1, C]$. All other $m^{(i)}_{t,\star}$ are set to false. This gives a satisfying assignment with exactly $\ell(C + 1) + (\ell + 1)k = K$ variables set to true.

Suppose now that there is a satisfying assignment with K variables set to true. At most one $a_{t,v}$ variable can be true for each $t \in [1, \ell]$. Exactly k variables of the form $s_{0,v}$ are set to true by the initial condition. If there are k' tokens true at time t, then there are at most $k' + 1$ tokens true at time $t + 1$ and so the $s_{t,v}$ and $a_{t,v}$ variables together can constitute at most $((k + \ell) + 1)\ell \leq C - 1$ true variables. Therefore, there must be strictly more than $(C + 1)(\ell - 1)$ variables of the form $m^{(i)}_{t,\star}$ that are set to true. Since $m^{(i)}_{t,\star}$ must take the same value as $m^{(j)}_{t,\star}$, there must be at least ℓ variables of the form $m^{(1)}_{t,\star}$ that are set to true. There can be at most one per time step t, and so there is exactly one per time step. We consider the TJ-independent set reconfiguration sequence $I_{\mathsf{init}} = I'_0, \dots, I'_\ell = I_{\mathsf{fin}}$ where for $t \in [1, \ell]$ we define $I'_t = I'_{t-1}$ if $m_{t,\emptyset}$ is true, and $I'_t = I'_{t-1} \setminus \{v\} \cup \{w\}$ if $m^{(1)}_{t,v,w}$ is true. The subsequence $I_{\mathsf{init}} = I_0, \dots, I_T = I_{\mathsf{fin}}$ obtained by removing I'_t if $I'_t = I'_{t-1}$, is now a valid TJ-independent set reconfiguration sequence. ◀

6 Conclusion

We showed that for independent set reconfiguration problems parameterized by the number of tokens, the complexity may vary widely depending on the way the length ℓ of the sequence is treated. If no bound is given, then we ask for the existence of an undirected path in the reconfiguration graph[5] and indeed the problem is XL-complete. If ℓ is given in binary, then we may in particular choose it larger than the maximum number of vertices in the reconfiguration graph, and so this problem is at least as hard as the previous. We show it to be XNL-complete. When ℓ is given in unary, it is easier to have a running time polynomial in ℓ, and indeed the problems becomes XNLP-complete. When ℓ is taken as parameter, the problem is $W[1]$-complete.

On the other hand, switching the rules of how the tokens may move does not affect the parameterized complexity, and the results for dominating set reconfiguration are also similar. It would be interesting to investigate for which graph classes switching between token jumping and token sliding does affect the parameterized complexities. We give an explicit suggestion below.

▶ **Problem 15.** *For which graphs H is TJ-Independent Set Reconfiguration equivalent to TS-Independent Set Reconfiguration under pl-reductions for the class of graphs with no induced H?*

The answer might also differ for Independent Set Reconfiguration and Dominating Set Reconfiguration. We remark that TJ-Clique Reconfiguration and TS-Clique Reconfiguration have the same complexity for all graph classes [8].

References

1 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized Problems Complete for Nondeterministic FPT time and Logarithmic Space. *arXiv:2105.14882*, 2021.

2 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. *CoRR*, abs/2106.15907, 2021. arXiv:2106.15907.

3 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 13–29. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214407.

4 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

5 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

6 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.

7 Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In T. V. Gopal, Manindra Agrawal, Angsheng Li, and S. Barry Cooper, editors, *Theory and Applications of Models of Computation*, pages 341–351, Cham, 2014. Springer International Publishing.

[5] The reconfiguration graph has the possible token configurations as vertex set, and there is an edge between two configurations if we can go from one to the other with a single move.

8 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. In *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation*, pages 212–223, 2015.

9 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

10 Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.

11 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1), 2018.

12 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.

13 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4), 2018.

14 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409:127–160, 2013.

15 Michael Wehar. *On the complexity of intersection non-emptiness problems.* PhD thesis, State University of New York at Buffalo, 2017.