Sliding Token on Bipartite Permutation Graphs

Eli Fox-Epstein[1], Duc A. Hoang[2], Yota Otachi[2], and Ryuhei Uehara[2]

[1] Brown University, USA. ef@cs.brown.edu
[2] JAIST, Japan. {hoanganhduc,otachi,uehara}@jaist.ac.jp

Abstract. Sliding Token is a natural reconfiguration problem in which vertices of independent sets are iteratively replaced by neighbors. We develop techniques that may be useful in answering the conjecture that Sliding Token is polynomial-time decidable on bipartite graphs. Along the way, we give efficient algorithms for Sliding Token on bipartite permutation and bipartite distance-hereditary graphs.

1 Introduction

Reconfiguration problems have been subject to much recent attention and study. We focus on just one reconfiguration problem, Sliding Token, which is a natural reconfiguration problem over independent sets on graphs. Recall that an *independent set* of a graph is a subset of its vertices such that no two are adjacent. A vertex in an independent set is called a *token*. Intuitively, one "slides" tokens across edges to form new independent sets.

For independent sets I and J, we write $I \overset{G}{\leftrightarrow} J$ if $|I| = |J|$ and there exists an edge $uv \in E(G)$ where $I \triangle J = \{u, v\}$, where $\triangle$ denotes symmetric difference. A *reconfiguration sequence* is a sequence of independent sets $\langle I_1, I_2, \ldots, I_k \rangle$ such that $I_i \overset{G}{\leftrightarrow} I_{i+1}$ for all $1 \leq i < k$. For independent sets I and J on graph G, the binary relation $I \overset{G}{\leftsquigarrow} J$ denotes that a reconfiguration sequence containing both I and J exists. "$\overset{G}{\leftsquigarrow}$" partitions independent sets into equivalence classes: let $[I]_G = \{J \mid I \overset{G}{\leftsquigarrow} J\}$ be the equivalence class of I (with the subscript omitted when implied from context). A yes-instance of Sliding Token is a graph G and independent sets I and J where $I \overset{G}{\leftsquigarrow} J$.

Ito et al. [4] show Sliding Token is PSPACE-complete. Kamiński et al. give a linear-time algorithm for Sliding Token on cographs. There are also polynomial-time algorithms on trees and claw-free graphs for Sliding Token [2, 1]. On graphs of bounded bandwidth (and thus treewidth), Sliding Token remains PSPACE-complete [10]. Sliding Token is $W[1]$-hard parameterized only by the length of the reconfiguration sequence [7, 5].

1.1 Preliminaries

Let G be a graph with vertex set $V(G)$ (with $n = |V(G)|$) and edge set $E(G)$, and S a subset of its vertices. $G[S]$ is the subgraph induced by S: the graph

with vertex set S and edge set $E(G) \cap (S \times S)$. Define $G \setminus S$ as $G[V(G) \setminus S]$. $N_G(v)$ is the set of all vertices adjacent to v in G and $N_G[v] = N_G(v) \cup \{v\}$. $N_G[S] = \cup_{v \in S} N_G[v]$ for vertex-subset S. When the graph is unambiguous, it is omitted from the notation.

Let $R(G, I) = \{v \mid v \in \cap_{I' \in [I]_G} I'\}$ be the subset of I containing all of the tokens v such that $v \in I'$ for all $I' \in [I]_G$. Vertices in $R(G, I)$ are called *rigid* with respect to G and I. An independent set I is *unlocked* if $R(G, I) = \emptyset$.

Because we frequently form sets that are just slight modifications of others, we write $A + x$ to be $A \cup \{x\}$ and $A - x$ to be $A \setminus \{x\}$.

A graph is a *permutation* graph if and only if there is a bijection between the vertices and a set of line segments between two parallel vertical lines such that two vertices are adjacent if and only if their corresponding segments intersect. A *bipartite permutation* graph is a permutation graph that has no odd-length cycles.

Given an ordering $\langle v_1, \ldots, v_n \rangle$ of the vertices of a graph, let $N_G^+(v_i) = N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\}$. Similarly, define $N_G^-(v_i) = N_G(v_i) \cap \{v_1, \ldots, v_{i-1}\}$.

The following is easily derived from e.g. [8, 9]:

Proposition 1. *Each connected bipartite permutation graph G has an ordering $\langle v_1, v_2, \ldots, v_n \rangle$ to $V(G)$ such that*

1. *for all $j > 1$, $N(v_j) \not\subset N(v_1)$,*
2. *for all $i \leq j \leq k$, every path from v_i to v_k contains some vertex in $N_G[v_j]$*
3. *$v_2 \in N(v_1)$ if $n > 1$,*
4. *v_2 is a pendant only if $n = 2$,*
5. *for all i and j where $1 \leq i < j \leq n$, v_i's distance to v_1 is at most v_j's distance to v_1, and*
6. *for all i and j where $1 \leq i < j \leq n$ and v_i and v_j have equal distance to v_1, $N_G^-(v_j) \subseteq N_G^-(v_i)$ and $N_G^+(v_i) \subseteq N_G^+(v_j)$, and*
7. *$N_G^-(v_i) \neq \emptyset$ for all $1 < i \leq n$.*

Such an ordering can be found in linear time. □

Bipartite permutation graphs may seem somewhat arbitrary; however, their many definitions make them a compelling class to study. For example, they are also characterized as bipartite AT-free graphs, bipartite bounded tolerance graphs, bipartite tolerance graphs, bipartite trapezoid graphs, and unit interval bigraphs. They are well studied (see e.g. [8]) and Sliding Token is PSPACE-complete on some slight non-bipartite generalizations (e.g. AT-free, perfect [6]).

We present an algorithm to efficiently decide Sliding Token on bipartite permutation graphs. Our main theorem is:

Theorem 1. Sliding Token *can be decided in polynomial time on bipartite permutation graphs of n vertices.*

This result bounds the diameter of the "reconfiguration graph" for Sliding Token on a bipartite permutation graph; the algorithm produces a sequence of length quadratic in the number of tokens if any sequence exists. Because of this,

determining if there exists a reconfiguration sequence of length at most k is in NP.

To prove the main result, we first give some results about general and biparite graphs in Sections 2 and 3. We prove our main result in Section 4 and then briefly show how techniques developed within can be applied to other classes of bipartite graphs.

2 Coping with Rigid Tokens

In general, tokens may be confined to specific areas of the graph. For example, in the PSPACE-hardness reduction for Sliding Token given by Demaine and Hearn [3], no token can ever slide out of its specific gadget (see e.g. Theorem 23 in [3]). Rigidity is a much stricter form of confinement; easing proof of strong statements about it, and for the purposes of Sliding Token on bipartite permutation graphs, it is not too restrictive. Once identified, rigid vertices and their neighborhoods can be deleted. This allows algorithms to only consider instances without rigid vertices, which, in this case, significantly simplifies them.

Proposition 2. *If G' is an induced subgraph of G and $I \overset{G'}{\leftrightsquigarrow} J$, then $I \overset{G}{\leftrightsquigarrow} J$ via the same reconfiguration sequence.* □

Proposition 3. *$I \overset{G}{\leftrightsquigarrow} J$ if and only if $I - v \overset{G\setminus N[v]}{\leftrightsquigarrow} J - v$ for any $v \in R(G, I) \cap R(G, J)$.*

Proof. First, assume $I \overset{G}{\leftrightsquigarrow} J$. Fix a reconfiguration sequence $\langle I = I_0, I_1, \ldots, I_k = J\rangle$. $v \in I_j$ and $N(v) \cap I_j = \emptyset$ for $0 \leq j \leq k$. Therefore, simply remove v from all I_j, $0 \leq j \leq k$, and remove $N_G[v]$ from G: the sets remain independent and do not use deleted vertices.

Next, suppose $I-v \overset{G\setminus N[v]}{\leftrightsquigarrow} J-v$. Proposition 2 gives $I-v \overset{G}{\leftrightsquigarrow} J-v$. Modify the reconfiguration sequence by inserting v into each independent set. This maintains independence: no vertex in $N_G(v)$ is in the reconfiguration sequence as those vertices do not exist in the induced subgraph. □

Proposition 4. *$I \overset{G}{\leftrightsquigarrow} J$ if and only if $R(G, I) = R(G, J)$ and $I \setminus R(G, I) \overset{G\setminus N[R(G,I)]}{\leftrightsquigarrow} J \setminus R(G, I)$.*

Proof. By definition of rigidity, if $R(G, I) \neq R(G, J)$ then $J \notin [I]_G$. Repeated application of Proposition 3 implies the other direction. □

Proposition 5. *Let I be an independent set and $S \subseteq I$. If, for all $w \in N(S)$, $|N(w) \cap S| > 1$, then $S \subseteq R(G, I)$.* □

Algorithm 1: SwitchSides(A, B, E, I_0)

Input: Bipartite graph $G = (A \cup B, E)$, independent set I_0
Output: Reconfiguration sequence $\langle I_0, \ldots, I_k \rangle$ where $I_0 \cap I_k \cap A = R(G, I_0) \cap A$ and $k = |I_0| - |R(G, I_0) \cap A|$

$M \leftarrow \emptyset$ `// Will hold available slides`
$C \leftarrow$ table from vertices to subsets of vertices
`// Initialize` M
foreach *vertex* $u \in B$ **do**
    $C_u \leftarrow N(u) \cap I_0$
    if $|C_u| = 1$ **then**
        $M \leftarrow M \cup \{u\}$
$k \leftarrow 0$
while $|M| > 0$ **do**
    $k \leftarrow k + 1$
    $u \leftarrow$ remove an arbitrary element u from M `//` $u \in B$ `will be in` I_k
    $v \leftarrow$ remove the unique vertex v from C_u `//` $v \in I_{k-1}$
    $I_k \leftarrow I_{k-1} - v + u$
    foreach *vertex* $w \in N(v)$ **do**
        $C_w \leftarrow C_w - v$
        if $|C_w| = 1$ **then**
            $M \leftarrow M \cup \{w\}$
return $\langle I_0, I_1, \ldots, I_k \rangle$

Algorithm 2: Wiggle(A, B, E, I_0)

Input: Bipartite graph $G = (A \cup B, E)$, independent set I_0
Output: Reconfiguration sequence $\langle I_0, \ldots, I_k \rangle$ with $k \leq 4|I_0|$ such that for all $v \in I_0 \setminus R(G, I_0)$, there is some j where $I_j \setminus I_{j-1} = \{v\}$

$\langle I_0, \ldots, I_{k_1} \rangle \leftarrow$ SwitchSides$(A, B, E(G), I_0)$
$\langle I_0 = I'_0, \ldots, I'_{k_2} \rangle \leftarrow$ SwitchSides$(B, A, E(G), I_0)$
return $\langle I_0, \ldots, I_{k_1}, I_{k_1-1}, \ldots, I_0, I'_1, \ldots, I'_{k_2}, I'_{k_2-1}, \ldots, I_0 \rangle$

3 An Algorithm on Bipartite Graphs

In this section, we show that it is relatively straightforward to manipulate the tokens of an independent set in a bipartite graph in a number of ways to e.g. find rigid tokens. In general graphs, identifying $R(G, I)$ is PSPACE-complete.

Proposition 6. *Given a bipartite graph $G = (A \cup B, E)$ and an independent set I_0, in linear time a reconfiguration sequence $\langle I_0, \ldots, I_k \rangle$ can be computed where $I_0 \cap I_k \cap A = R(G, I_0) \cap A$ and $k = |I_0| - |R(G, I_0 \cap A)|$.*

Proof. We analyze Algorithm 1.

Runtime. The first loop, when processing u, charges its work to all the incident edges to u. Charge each iteration of the inner loop (lines 14–16) to the edge vw and charge the work on lines 9–12 to the vertex v. No edge or vertex is charged more than twice, and each charge takes $O(1)$ time.

Correctness. Let C_u^t (M^t) be the state of C_u (resp., M) at the top of the tth execution of the while loop (i.e. at line 9 when k is incremented to be t).

The while loop of Algorithm 1 maintains these properties going into the tth iteration: (P1) $C_u^t = N(u) \cap I_{t-1}$ for all vertices $u \in B$ and (P2) $M^t = \{u \in I_0 : |C_u^t| = 1\}$.

The output is a valid reconfiguration sequence because (1) I_k and I_{k-1} differ by adjacent vertices (line 12) and (2) P1 guarantees that each set is independent.

Next, we prove $I_0 \cap I_k \cap A = R(G, I_0) \cap A$. As only vertices in $I_0 \cap A$ are removed from an independent set during the reconfiguration sequence, both I_0 and I_k contain $I_0 \cap B$. Since it is a valid reconfiguration sequence, we know $R(G, I_0) \subseteq I_0 \cap I_k$. Thus, $R(G, I_0) \subseteq I_0 \cap I_k$ and it remains to be shown that no non-rigid vertices of $I_0 \setminus (R(G, I_0) \cap A)$ are in I_k. Since M is empty at the end of the algorithm, $|C_u| \neq 1$ for all $u \in I_k$. Consider $S = I_0 \cap I_k \cap A$. Any $w \in N(S)$ must have $|C_w| > 1$ by property (P1), so Proposition 5 with G and S shows $S \subseteq R(G, I_0)$. Thus, $I_0 \cap I_k \cap A = R(G, I_0) \cap A$.

Finally, we show that the length of the reconfiguration sequence, k, is as promised. For all $0 < j \leq k$, we have that $|I_j \cap I_0| = |I_{j-1} \cap I_0| - 1$, so $|I_0| - |R(G, I_0) \cap A|$ is an upper bound on k. To lower-bound k, it takes k slides to reconfigure k vertices out of I_0. □

Algorithm 2 applies Algorithm 1 twice to produce a sequence that starts and ends with the same sequence but ensures that each token not in $R(G, I)$ slides exactly twice.

Lemma 1. *Given bipartite graph $G = (A \cup B, E)$, and independent set I_0 in linear time Algorithm 2 finds a reconfiguration sequence of length at most $4|I_0|$ in which each token of $I_0 \setminus R(G, I_0)$ slides exactly twice.* □

Lemma 2. *Let $G = (A \cup B, E)$ be a bipartite graph and I be an independent set of G. In linear time, $R(G, I)$ can be computed.*

Proof. Invoke Algorithm 2. By the post-condition promises, the tokens that never slid in the output sequence are exactly $R(G, I)$. □

Lemma 3. *Let $G = (A \cup B, E)$ be a connected bipartite graph and I an unlocked independent set. Then for any $v \in V(G)$, in linear time, one can find a reconfiguration sequence $\langle I = I_0, I_1, \ldots, I_k = J \rangle$ where $v \in J$, $v \notin I_{k-1}$, and k is at most $|I|$ plus the distance between v and the closest token of I.*

Proof. We distinguish 3 cases:

(1) If $v \in I$, the entire sequence is just $\langle I \rangle$.

(2) If there is a unique closest token w in I to v, the reconfiguration sequence repeatedly replaces that token with a vertex that is one closer to v. Let u be any vertex in $N(v)$ where some shortest path from w to v passes through u. Since w is uniquely closest to v among all tokens in I, it must be the case that $N(u) \cap I = \{w\}$. So update construct $I' = I - w + u$; u is now uniquely closest in I' to v, so this process can be repeated.

(3) Otherwise, let S be the set of all closest vertices to v at distance d. Without loss of generality assume $S \subseteq A$. By the correctness of Algorithm 1, there is a $J \in [I]$ where $J \subseteq B$. Consider a reconfiguration sequence $\langle I = I_0, I_1, \ldots, I_k = J \rangle$ from I to J. There must be an index j, with $j \leq k \leq |I|$, where I_j has a unique closest token to v as either some token will first move to be distance $d-1$ away from v, or all but one token will slide to be at least distance $d+1$ away. Then, from I_j, the reconfiguration sequence is as described in case (2). □

We write I_v^G (with the graph usually omitted) to indicate an independent set resulting in invoking Lemma 3 on G and I to place a vertex on v. This produces some reconfiguration sequence of linear length from I to I_v, in which I_v is the only independent set containing v.

We are able to simplify instances with the following lemma:

Lemma 4. *Let I be an unlocked independent set in bipartite graph G.*
(1) If $N[v] \cap I = \emptyset$, then $R(G \setminus \{v\}, I) = \emptyset$.
(2) If $N[N[v]] \cap I \subseteq \{v\}$, then $R(G \setminus N[v], I - v) = \emptyset$.

Proof. Invoke Algorithm 2 on G. Since all tokens move no farther than to their neighbors, both cases immediately follow. □

Proposition 7. *Suppose $N_G(u) = N_G(v)$. For any unlocked independent sets I and J, $I \overset{G}{\leftrightsquigarrow} J$ if and only if $I_u^G \overset{G \setminus \{v\}}{\leftrightsquigarrow} J_u^G$.* □

4 Sliding Token on Bipartite Permutation Graphs

Throughout the section, let G be a bipartite permutation graph with vertices $\langle v_1, v_2, \ldots, v_n \rangle$ ordered as described previously.

Proposition 8. *Assume $R(G, I) = R(G, J)$. If $v_i \in R(G, I)$, then each component of $G \setminus N[v_i]$ is a bipartite permutation graph and $I \overset{G}{\leftrightsquigarrow} J$ if and only if, for each component C of $G \setminus N[v_i]$, we have $I \cap C \overset{G[C]}{\leftrightsquigarrow} J \cap C$.*

Proof. First, note that an induced subgraph of a bipartite permutation graph is still a bipartite permutation graph. Now, we appeal to Proposition 4. □

Lemma 2 locates rigid vertices in linear time and Proposition 8 permits treating each component independently after deleting rigid vertices and their neighborhoods. We assume $R(G, I) = \emptyset$ for the remainder of the section. Using Proposition 7 allows us to assume that each vertex has a distinct neighborhood.

In each equivalence class over $\overset{G}{\leftrightsquigarrow}$, we will pick a representative independent set by defining a injective function $f(\cdot)$ from independent sets to natural numbers: the representative will be the independent set in the equivalence class that minimizes the function. The function used is

$$f(I) = \sum_{v_i \in I} 2^i.$$

We write I_+ to indicate the representative of the equivalence class to which some independent set I belongs: $I_+ = \arg\min_{I' \in [I]_G} f(I')$. Then, deciding if $I \overset{G}{\leftrightsquigarrow} J$ is equivalent to determining if $I_+ = J_+$.

To give some intuition on why finding I_+ is nontrivial, Figure 1 illustrates two unlocked independent sets in different equivalence classes.

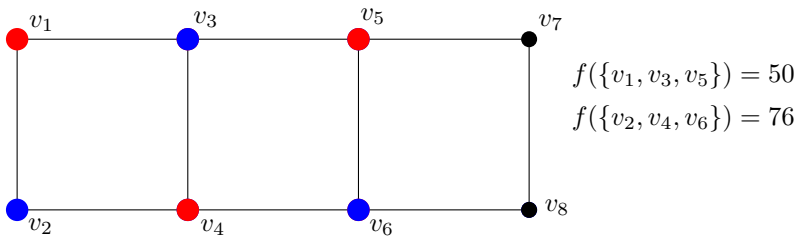

Fig. 1. Two unlocked independent sets in different equivalence classes: $\{v_1, v_3, v_5\}$ and $\{v_2, v_4, v_6\}$.

Fix some I and let w_j^+ be the jth least token of I_+. The algorithm relies on two vital observations: first, that there are only two possibilities for where the token of least index in I will reside in I_+ and second, that I_+ can be assembled one vertex at a time.

Proposition 9. $|\{v_1, v_2\} \cap I_+| = 1$. *If* $|I_+| \geq 2$ *and* $v_2 \in I_+$ *then* $|N(v_1) \cap I_+| \geq 2$.

Proof. First, we prove $|\{v_1, v_2\} \cap I_+| = 1$. Suppose not: that $w_1^+ = v_i$ for some $i > 1$. There are two cases to consider:

(1) Assume $v_i \in N(v_1)$. Use Lemma 3 to place a token on v_1 and obtain a reconfiguration sequence $\langle I_+ = I_0, I_1, \ldots, I_k \rangle$. Recall that $v_i \in I_j$ for all $j < k$. Consider the sequence $\langle I_0, \ldots, I_k, I_{k-1} - v_i + v_2, I_{k-1} - v_i + v_2, I_{k-2} - v_i +$

$v_2, \ldots, I_0 - v_i + v_2\rangle$. This sequence is valid, so $I_+ - v_i + v_2 \in [I_+]$. But $f(I_+ - v_i + v_2) < f(I_+)$, a contradiction.

(2) Now assume $v_i \notin N(v_1)$. Again use Lemma 3 to place a token on v_1. Similarly, the sequence can be unrolled in reverse, except this time leaving a token on v_1.

Now we prove if $v_2 \in I_+$ then $|N(v_1) \cap I_+| \geq 2$. Suppose not: that $v_2 \in I_+$ but $N(v_1) \cap I_+ = \{v_2\}$. Then $I_+ \overset{G}{\leftrightarrow} I_+ - v_2 + v_1$ is legal, which decreases f, a contradiction. □

Proposition 10. *If I is an unlocked independent set containing w_1^+ and w_2^+ then $R(G \setminus N[w_1^+], I - w_1^+) \subseteq \{w_2^+\}$.*

Proof. We assume $|I| > 2$ as the statement is otherwise trivial. Let $v_j = w_2^+$ for some $j > 2$. We proceed with a complicated case analysis:

1. Assume $v_1 \in I$.
   (a) Assume $N[N[v_1]] \cap I = \{v_1\}$. Then Lemma 4 applies to I.
   (b) Assume v_1 is a pendant. For v_1 to slide, at some set I' in the reconfiguration sequence given by Algorithm 2, $N(v_2) \cap I' = \{v_1\}$. Lemma 4 applies to I'.
   (c) Assume no neighbor of v_1 has v_1 as its only neighboring token. $N(v_1) \subseteq N(v_j)$ (otherwise, we fall into one of the previous cases) so the token on v_1 cannot slide until v_j slides. Once v_j slides, $N[N[v_1]] = \{v_1\}$ and Lemma 4 completes the proof.
   (d) Otherwise, observe that $N(v_2) \cap I = \{v_1\}$. Let L_i be the set of vertices distance i away from v_1.

   If any two vertices v_a, v_b in $I \cap L_2$ have $N^-(u) = N^-(v)$, then Algorithm 1 slides all vertices of L_2 with index at least b (assuming $a < b$) into L_3. Notice that it suffices to show that $a = j$.

   In I_+, there must be a k where $k > j$ and $N^-(v_k) = N^-(v_j)$; (otherwise $I_+ - v_1 + v_2 - v_j + v_i$ would improve on $f(\cdot)$ for some $i < j$). However, if $v_k \notin I$, more argument is required. Consider any reconfiguration sequence from I to I_+. Let I' be the last independent set in the sequence containing v_k. In I', the token of second-least index cannot be in $N^-(v_k)$ but must be in L_2. We show this gives a contradiction to I_+ minimizing f: since the token on v_k does not slide for the remainder of the reconfiguration sequence, the two first tokens are able to reconfigure from v_1 and v_j in I_+ to a configuration with smaller f-value.
2. Assume $v_2 \in I$. By Proposition 9, $v_j \in N(v_1)$. Thus, $N(v_2) \subseteq N(v_1)$. Consider a reconfiguration sequence in which v_2 eventually slides, e.g. the one generated by Lemma 3 to produce $I_{v_1}^G$. In this, v_j must slide before v_1. Let I' be the independent set immediately after v_j slides. $N[N[v_i]] \cap I' = \{v_i\}$, so Lemma 4 applies.

□

Proposition 11. *$I_+ - w_1^+$ is f-minimal on $G \setminus N[w_1^+]$.* □

We find a reconfiguration sequence between I and I_+ using dynamic programming over vertex index with a table $T[\cdot]$. For notational convenience, we define $J^{i,k} = \{v_j \in J \mid i \leq j \leq k\}$ for any independent set J. Let G_i be the unique component of $G \setminus N[v_i]$ containing vertices of higher index. $T[i]$ will be assigned some $J = \arg\max_{J \in [I]: J \ni v_i} |J^{0,i}|$. As a base case, set $T[0] = I$.

Define

$$W(i,j) = \begin{cases} T[j] & \text{if } v_i \in T[j] \\ T[j]^{0,k} \cup (T[j]^{j+1,n})_{v_i}^{G_j} & \text{if } R(G_j, T[j]^{j+1,n}) = \emptyset \\ \text{“invalid”} & \text{otherwise.} \end{cases}$$

(Recall, the notation in the middle case invokes Lemma 3.) Say $W(i,j)$ is *valid* if $0 \leq j < i$ and $W(i,j)$ is an independent set and not “invalid”. Among the valid $W(i,j)$ that maximize $|W(i,j)^{0,i}|$, set $T[i]$ to be the $W(i,j)$ where j is least.

Lemma 5. *If $v_i \in I_+$ then $T[i]^{0,i} = I_+^{0,i}$.*

Proof. Using Proposition 10 and Proposition 11, this follows from a simple induction on the size of I_+. □

Theorem 2. *Given a connected bipartite permutation graph G and an unlocked independent set I, there is a cubic-time algorithm to find I_+.*

Proof. Given the dynamic programming table $T[\cdot]$, find the least index i where $|T[i]^{0,i}| = |I|$ and report $I_+ = T[i]$; by Lemma 5, this is correct.

In total, $O(n^2)$ sets $W(i,j)$ are computed, each of which takes linear time, giving cubic runtime. □

Given this, proving the main theorem is straightforward:

Proof (of Theorem 1). As input, we are given a bipartite permutation graph G and two independent sets I and J. If $R(G,I) \neq R(G,J)$, then output “no”. Otherwise, form $G' = G \setminus N[R(G,I)]$. For each C component of G', find $I' = I \cap C$ and $J' = J \cap C$; then find I'^+ and J'^+ using Theorem 2. If in any component, I'^+ and J'^+ differ, then output “no”. Otherwise, it must be that $I \overset{G}{\leftrightsquigarrow} J$. □

5 Sliding Token on Bipartite Distance-Hereditary Graphs

In this section, we give an additional application of the techniques built in Section 3. A graph is *distance-hereditary* if the distance between two vertices in any connected induced subgraph is exactly the distance in the original graph. One characterization of bipartite distance-hereditary graphs is graphs obtainable from a single vertex by repeatedly picking a vertex v in the graph and then adding a new vertex w with either $N(w) = \{v\}$ (pendant) or $N(w) = N(v)$ (twin).

Theorem 3. *There is a polynomial-time algorithm to decide* Sliding Token *on bipartite distance-hereditary graphs.*

Proof. Let I_0 and J_0 be independent sets of the same cardinality on bipartite distance-hereditary graph G. We analyze the following algorithm.

We can assume, using Lemma 2 and Proposition 4 that $R(G, I) = R(G, J) = \emptyset$. Repeatedly:

1. If $N(v) = N(w)$ for any v, w, use Lemma 3 to place a token on v in I and in J, and then delete w.
2. Else, if there is a pendant v whose neighbor w has degree 2, use Lemma 3 to place a token from I and from J on v, then delete $N(v)$.
3. Otherwise, compute a sequence of operations used to construct the graph and look at the last twin operation used. At least one of the two involved vertices must have a pendant. Use Lemma 3 to place a token from I and from J on the pendant and delete it and its neighborhood.

Bipartite distance-hereditary graphs are closed under vertex deletion, so after each iteration the graph remains bipartite distance-hereditary. Suppose that before an iteration, $R(G, I) = \emptyset$. Let G', I', J' be the graph and independent sets after the iteration. We show that $R(G', I') = R(G', J') = \emptyset$.

In case (1), since $N_G(v) = N_G(w)$ and $v \in I' \cap J'$, we have $N_G(w) \cap I' = N_G(w) \cap J' = \emptyset$. Lemma 4 implies $R(G', I') = R(G', J') = \emptyset$. In cases (2) and (3), if there is a token on any neighbor u of w besides v in I', then after invoking Algorithm 2, there must be an intermediate independent set I'' where $v \in I''$ but $u \notin I''$. From I'', Lemma 4 completes the proof. □

6 Discussion

We show that Sliding Token can be efficiently decided on bipartite permutation graphs and bipartite distance-hereditary graphs. The results of [6] show that Sliding Token is PSPACE-hard on AT-free graphs, which are a natural generalization of bipartite permutation graphs to non-bipartite graphs. This suggests that bipartitedness is closely related to the complexity of Sliding Token.

The complexity of Sliding Token on bipartite graphs remains a compelling topic for future research; the tools developed here tackle rigidity but need strengthening to be able to decide Sliding Token when dynamic programming does not fit as naturally.

References

1. P. S. Bonsma, M. Kaminski, and M. Wrochna. Reconfiguring independent sets in claw-free graphs. In *Algorithm Theory - SWAT 2014*.
2. E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada. Polynomial-time algorithm for sliding tokens on trees. In *Algorithms and Computation*, volume 8889 of *Lecture Notes in Computer Science*, pages 389–400. Springer International Publishing, 2014.
3. R. A. Hearn and E. D. Demaine. PSPACE-completeness of Sliding-block Puzzles and Other Problems Through the Nondeterministic Constraint Logic Model of Computation. *Theor. Comput. Sci.*, 343(1-2):72–96, October 2005.

4. T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. In *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 28–39. Springer Berlin Heidelberg, 2008.
5. T. Ito, M. Kamiński, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka. On the parameterized complexity for token jumping on graphs. In *Theory and Applications of Models of Computation*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer International Publishing, 2014.
6. M. Kamiński, P. Medvedev, and M. Milani. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, June 2012.
7. A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Parameterized and Exact Computation*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer International Publishing, 2014.
8. J. Spinrad, A. Brandstädt, and L. Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.
9. A. P. Sprague. Recognition of bipartite permutation graphs. *Congressus Numerantium*, 62:151–161, 1995.
10. M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. *CoRR*, abs/1405.0847, 2014.