# Shortest Reconfiguration of Sliding Tokens on a Caterpillar

Takeshi Yamada[1] and Ryuhei Uehara[1]

School of Information Science, JAIST, Japan.
{tyama,uehara}@jaist.ac.jp

**Abstract.** For given two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a graph, the SLIDING TOKEN problem is to determine if there exists a sequence of independent sets which transforms $\mathbf{I}_b$ into $\mathbf{I}_r$ so that each independent set in the sequence results from the previous one by sliding exactly one token along an edge in the graph. The SLIDING TOKEN problem is one of the reconfiguration problems that attract the attention from the viewpoint of theoretical computer science. These problems tend to be PSPACE-complete in general, and some polynomial time algorithms are shown in restricted cases. Recently, the problems for finding a *shortest* reconfiguration sequence are investigated. For the 3SAT reconfiguration problem, a trichotomy for the complexity of finding the shortest sequence has been shown; it is in P, NP-complete, or PSPACE-complete in certain conditions. Even if it is polynomial time solvable to decide whether two instances are reconfigured with each other, it can be NP-complete to find a shortest sequence between them. We show nontrivial polynomial time algorithms for finding a shortest sequence between two independent sets for some graph classes. As far as the authors know, one of them is the first polynomial time algorithm for the SHORTEST SLIDING TOKEN PROBLEM that requires detours of tokens.

## 1 Introduction

Recently, the *reconfiguration problems* attract the attention from the viewpoint of theoretical computer science. The problem arises when we wish to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible and each step abides by a fixed reconfiguration rule. The reconfiguration problems have been studied extensively for several well-known problems, including SATISFIABILITY [7, 13], INDEPENDENT SET [8–11, 15], SET COVER, CLIQUE, MATCHING [10], and so on.

The reconfiguration problem can be seen as a natural "puzzle" from the viewpoint of recreational mathematics. The *15 puzzle* is one of the most famous classic puzzles, that had the greatest impact on American and European society of any mechanical puzzle the word has ever known in 1880 (Fig. 1; see [18] for its history). It is well known that it has a parity; for any two placements, we can decide if they are reconfigurable or not by the parity. Thus, we can solve the reconfiguration problem in linear time just by checking their parities. Moreover,

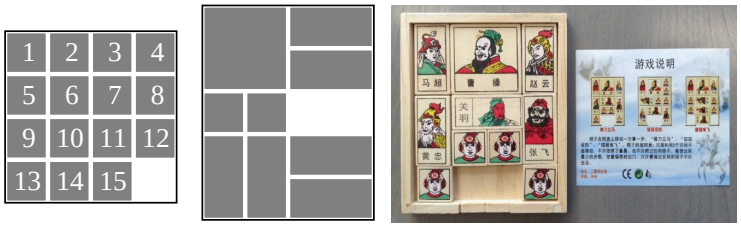


**Fig. 1.** The 15 puzzle, Dad's puzzle, and its Chinese variant.

the distance between any two reconfigurable placements is $O(n^3)$, i.e., we can reconfigure from one to the other in $O(n^3)$ sliding pieces, where the board is of size $n \times n$. However, surprisingly, for these two reconfigurable placements, finding a shortest path is NP-complete in general [17]. That is, although we know that it is $O(n^3)$, finding a shortest one is NP-complete. Another interesting property of the 15 puzzle is in another way of generalization. We have the other famous classic puzzles that can be seen as a generalization from this viewpoint. Namely, while every piece is a unit square in the 15 puzzle, when rectangles are allowed, we have the other classic puzzles, called "Dad puzzle" and its variants (Fig. 1). In 1964, Gardner said that "These puzzles are very much in want of a theory" [6], and Hearn and Demaine gave it after 40 years [8]; they proved that these puzzles are PSPACE-complete using their nondeterministic constraint logic model [9]. That is, the sliding block puzzle is PSPACE-complete in general decision problem, and it is linear time solvable for unit square pieces. However, finding a shortest reconfiguration for unit square pieces is NP-complete. In other words, we can characterize these three complexity classes using the model of the sliding block puzzle.

From the viewpoint of theoretical computer science, one of the most important problems is the 3SAT problem. Recently, for the 3SAT problem, a similar trichotomy for the complexity of finding a shortest sequence has been shown; that is, for the reconfiguration problem, finding a shortest sequence between two satisfiable assignments is in P, NP-complete, or PSPACE-complete in certain conditions [14]. In general, the reconfiguration problems tend to be PSPACE-complete, and some polynomial time algorithms are shown in restricted cases. However, finding a shortest sequence can be a new trend in theoretical computer science because it has a potential to characterize the class NP and gives us a new insight into this class.

Beside the 3SAT problem, one of the most important problems in theoretical computer science is the independent set problem. For this notion, the natural reconfiguration problem is called the SLIDING TOKEN problem introduced by Hearn and Demaine [8]: Suppose that we are given two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a graph $G = (V, E)$ and imagine that a token is placed on each vertex in $\mathbf{I}_b$. Then, the SLIDING TOKEN problem asks if there exists a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of $G$ such that (a) $\mathbf{I}_1 = \mathbf{I}_b$, $\mathbf{I}_\ell = \mathbf{I}_r$, and $|\mathbf{I}_b| = |\mathbf{I}_i|$ for all

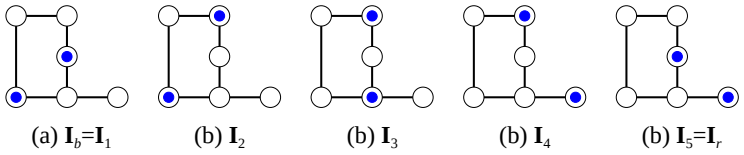


**Fig. 2.** A sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_5 \rangle$ of independent sets of the same graph, where the vertices in independent sets are depicted by small black circles (tokens).

$i$ with $1 \leq i \leq \ell$; and (b) for each $i$, $2 \leq i \leq \ell$, there is an edge $\{u, v\}$ in $E$ such that $\mathbf{I}_{i-1} \setminus \mathbf{I}_i = \{u\}$ and $\mathbf{I}_i \setminus \mathbf{I}_{i-1} = \{v\}$. Figure 2 illustrates a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_5 \rangle$ of independent sets which transforms $\mathbf{I}_b = \mathbf{I}_1$ into $\mathbf{I}_r = \mathbf{I}_5$. Hearn and Demaine proved that the SLIDING TOKEN problem is PSPACE-complete for planar graphs.

For the SLIDING TOKEN problem, some polynomial time algorithms are investigated as follows: Linear time algorithms have been shown for cographs [11] and trees [3]. Polynomial time algorithms are shown for bipartite permutation graphs [5] and claw-free graphs [1]. On the other hand, PSPACE-completeness is also shown for graphs of bounded tree-width [16] and planar graphs [9].

In this context, we investigate for finding a shortest sequence of the SLIDING TOKEN problem, which is called the SHORTEST SLIDING TOKEN problem defined as follows:

Input: A graph $G = (V, E)$ and two independent sets $\mathbf{I}_b, \mathbf{I}_r$ with $|\mathbf{I}_b| = |\mathbf{I}_r|$.
Output: A *shortest* reconfiguration sequence $\mathbf{I}_b = \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell = \mathbf{I}_r$ such that $\mathbf{I}_i$ can be obtained from $\mathbf{I}_{i-1}$ by sliding exactly one token on a vertex $u \in \mathbf{I}_{i-1}$ to its adjacent vertex $v$ along $\{u, v\} \in E$ for each $i$, $2 \leq i \leq \ell$.

We note that $\ell$ is not necessarily in polynomial of $|V|$; this is an issue how we formalize the problem, and if we do not know that $\ell$ is in polynomial or not. If the length $k$ is given as a part of input, we may be able to decide if $\ell \leq k$ in polynomial time even if $\ell$ itself is not in polynomial. However, if we have to output the sequence itself, it cannot be solved in polynomial time if $\ell$ is not in polynomial.

In this paper, we will show that the SHORTEST SLIDING TOKEN problem is solvable in polynomial time for the following graph classes:

*Proper interval graphs:* We first prove that every proper interval graph with two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ is a yes-instance if $|\mathbf{I}_b| = |\mathbf{I}_r|$. Furthermore, we can find the ordering of tokens to be slid in a shortest sequence in $O(n)$ time (implicitly), even though there exists an infinite family of independent sets on paths for which any sequence requires $\Omega(n^2)$ length.

*Trivially perfect graphs:* We then give an $O(n)$-time algorithm for trivially perfect graphs which actually finds a shortest sequence if such a sequence exists. In contrast to proper interval graphs, any shortest sequence is of length $O(n)$

for trivially perfect graphs. Note that trivially perfect graphs form a subclass of cographs, and hence its decision problem can be solved in polynomial time [11].

*Caterpillars:* We finally give an $O(n^2)$-time algorithm for caterpillars for the SHORTEST SLIDING TOKEN problem. To make self-contained, we first show a linear time algorithm for decision problem that asks if two independent sets can be transformed into each other. (We note that more general result for a tree has been shown [3].) For a yes-instance, we next show an algorithm that finds a shortest sequence between two independent sets.

We here remark that, since the problem is PSPACE-complete in general, an instance of the SLIDING TOKEN problem may require the exponential number of independent sets to transform. In such a case, tokens should make detours to avoid violating to be independent (as shown in Fig. 2). As we will see, caterpillars certainly require to make detours to transform. Therefore, it is remarkable that any yes-instance on a caterpillar requires a sequence of token-slides of polynomial length. This is still open even for a tree; in a tree, we can determine if two independent sets are reconfigurable in linear time due to [3], however, we do not know if the length of the sequence is in polynomial or not.

As far as the authors know, this is the first polynomial time algorithm for the shortest sliding token problem for a graph class that requires detours.

Due to the lack of space, some proofs are omitted, and available in a draft on arXiv [19].

## 2 Preliminaries

In this section, we introduce some basic terms and notations. In the SLIDING TOKEN problem, we may assume without loss of generality that graphs are simple and connected.

SLIDING TOKEN*:* For two independent sets $\mathbf{I}_i$ and $\mathbf{I}_j$ in a graph $G = (V, E)$, if there exists exactly one edge $\{u, v\}$ in $G$ such that $\mathbf{I}_i \setminus \mathbf{I}_j = \{u\}$ and $\mathbf{I}_j \setminus \mathbf{I}_i = \{v\}$, then we say that $\mathbf{I}_j$ can be obtained from $\mathbf{I}_i$ by *sliding* a token on the vertex $u \in \mathbf{I}_i$ to its adjacent vertex $v$ along the edge $\{u, v\}$, and denote it by $\mathbf{I}_i \vdash \mathbf{I}_j$.

A *reconfiguration sequence* between two independent sets $\mathbf{I}_1$ and $\mathbf{I}_\ell$ of $G$ is a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of $G$ such that $\mathbf{I}_{i-1} \vdash \mathbf{I}_i$ for $i = 2, 3, \ldots, \ell$. We denote by $\mathbf{I}_1 \vdash^* \mathbf{I}_\ell$ if there exists a reconfiguration sequence between $\mathbf{I}_1$ and $\mathbf{I}_\ell$. We note that a reconfiguration sequence is *reversible*, that is, we have $\mathbf{I}_1 \vdash^* \mathbf{I}_\ell$ iff $\mathbf{I}_\ell \vdash^* \mathbf{I}_1$. Thus we say that two independent sets $\mathbf{I}_1$ and $\mathbf{I}_\ell$ are *reconfigurable* into each other if $\mathbf{I}_1 \vdash^* \mathbf{I}_\ell$. The *length* of a reconfiguration sequence $\mathcal{S}$ is defined as the number of independent sets contained in $\mathcal{S}$. For example, the length of the reconfiguration sequence in Fig. 2 is 5.

The SLIDING TOKEN problem is to determine if two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ are reconfigurable into each other. We may assume without loss of generality that $|\mathbf{I}_b| = |\mathbf{I}_r|$; otherwise the answer is clearly "no." In this paper, we will consider the SHORTEST SLIDING TOKEN problem that computes the length

of a shortest reconfiguration sequence between two independent sets. Note that the length of a reconfiguration sequence may not be in polynomial of the size of the graph when the sequence may contain detours of tokens.

We always denote by $\mathbf{I}_b$ and $\mathbf{I}_r$ the initial and target independent sets of $G$, respectively, as an instance of the (SHORTEST) SLIDING TOKEN problem; we wish to slide tokens on the vertices in $\mathbf{I}_b$ to the vertices in $\mathbf{I}_r$. We sometimes call the vertices in $\mathbf{I}_b$ *blue*, and the vertices in $\mathbf{I}_r$ *red*; each vertex in $\mathbf{I}_b \cap \mathbf{I}_r$ is blue *and* red.

*Target-assignment:* We here give another notation of the SLIDING TOKEN problem to explain our algorithm. Let $\mathbf{I}_b = \{b_1, b_2, \ldots, b_k\}$ be an initial independent set of a graph $G$. For the sake of convenience, we label the tokens on the vertices in $\mathbf{I}_b$; let $t_i$ be the token placed on $b_i$ for each $i$, $1 \leq i \leq k$. Let $\mathcal{S}$ be a reconfiguration sequence between $\mathbf{I}_b$ and an independent set $\mathbf{I}$ of $G$, and hence $\mathbf{I}_b \vdash^* \mathbf{I}$. Then, for each token $t_i$, $1 \leq i \leq k$, we denote by $f_{\mathcal{S}}(t_i)$ the vertex in $\mathbf{I}$ on which the token $t_i$ is placed via the reconfiguration sequence $\mathcal{S}$. Notice that $\{f_{\mathcal{S}}(t_i) \mid 1 \leq i \leq k\} = \mathbf{I}$.

Let $\mathbf{I}_r$ be a target independent set of $G$, which is not necessarily reconfigurable from $\mathbf{I}_b$. Then, we call a mapping $g : \mathbf{I}_b \to \mathbf{I}_r$ a *target-assignment* between $\mathbf{I}_b$ and $\mathbf{I}_r$. The target-assignment $g$ is said to be *proper* if there exists a reconfiguration sequence $\mathcal{S}$ such that $f_{\mathcal{S}}(t_i) = g(b_i)$ for all $i$, $1 \leq i \leq k$. Note that there is no proper target-assignment between $\mathbf{I}_b$ and $\mathbf{I}_r$ if $\mathbf{I}_b \not\vdash^* \mathbf{I}_r$. Therefore, the SLIDING TOKEN problem can be seen as the problem of determining whether there exists at least one proper target-assignment between $\mathbf{I}_b$ and $\mathbf{I}_r$.

*Interval graphs and subclasses:* The *neighborhood* of a vertex $v$ in a graph $G = (V, E)$ is the set of all vertices adjacent to $v$, and denoted by $N(v) = \{u \in V \mid \{u, v\} \in E\}$. Let $N[v] = N(v) \cup \{v\}$. Two vertices $u$ and $v$ are called *strong twins* if $N[u] = N[v]$, and *weak twins* if $N(u) = N(v)$. We only consider the graphs without strong twins since only one of them can be used by a token for strong twins.

A graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ is an *interval graph* if there exists a set $\mathcal{I}$ of intervals $I_1, I_2, \ldots, I_n$ such that $\{v_i, v_j\} \in E$ iff $I_i \cap I_j \neq \emptyset$ for each $i$ and $j$ with $1 \leq i, j \leq n$.[1] We call the set $\mathcal{I}$ of intervals an *interval representation* of the graph, and sometimes identify a vertex $v_i \in V$ with its corresponding interval $I_i \in \mathcal{I}$. We denote by $L(I)$ and $R(I)$ the left and right endpoints of an interval $I \in \mathcal{I}$, respectively. That is, we always have $L(I) \leq R(I)$ for any interval $I = [L(I), R(I)]$.

We suppose that an interval graph $G = (V, E)$ is given as an input by its interval representation using $O(n)$ space, where $n = |V|$. (An interval representation of $G$ can be found in $O(n + m)$ time [12], where $m = |E|$.) More precisely, $G$ is given by a string of length $2n$ over alphabets $\{L(I_1), L(I_2), \ldots, L(I_n), R(I_1), R(I_2), \ldots, R(I_n)\}$.

[1] In this paper, a bold $\mathbf{I}$ denotes an "independent set," an italic $I$ denotes an "interval," and calligraphy $\mathcal{I}$ denotes "a set of intervals."

An interval graph is *proper* if it has an interval representation such that no interval properly contains another. An interval graph is *trivially perfect* if it has an interval representation such that the relationship between any two intervals is either disjoint or inclusion.

A *caterpillar* $G = (V, E)$ is a tree that consists of two subsets $S$ and $L$ of $V$ as follows. The vertex set $S$ induces a path $(s_1, \ldots, s_k)$ in $G$, each vertex $v$ in $L$ has degree 1, and its unique neighbor is in $S$. We call the path $(s_1, \ldots, s_k)$ *spine*, and each vertex in $L$ *leaf*. In this paper, without loss of generality, we assume that $k \geq 2$, $\deg(s_1) \geq 2$, and $\deg(s_k) \geq 2$. It is easy to see that the class of caterpillars is a proper subset of the class of interval graphs.

## 3 Proper Interval Graphs

We show the main theorem in this section for proper interval graphs. Firstly, the answer of SLIDING TOKEN is always "yes" for connected proper interval graphs. We give a constructive proof of the claim, and it certainly finds a shortest sequence in linear time.

**Theorem 1.** *For a connected proper interval graph $G = (V, E)$, any two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ with $|\mathbf{I}_b| = |\mathbf{I}_r|$ are reconfigurable into each other, i.e., $\mathbf{I}_b \vdash^* \mathbf{I}_r$. Moreover, the shortest reconfiguration sequence can be found in polynomial time.*

We give an algorithm which actually finds a shortest reconfiguration sequence between any two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a connected proper interval graph $G$. A connected proper interval graph $G = (V, E)$ has a unique interval representation (up to reversal), and we can assume that each interval is of unit length in the representation [4]. Therefore, by renumbering the vertices, we can fix an interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $G$ so that $L(I_i) < L(I_{i+1})$ (and $R(I_i) < R(I_{i+1})$) for each $i$, $1 \leq i \leq n-1$, and each interval $I_i \in \mathcal{I}$ corresponds to the vertex $v_i \in V$.

Let $\mathbf{I}_b = \{b_1, b_2, \ldots, b_k\}$ and $\mathbf{I}_r = \{r_1, r_2, \ldots, r_k\}$ be any given initial and target independent sets of $G$, respectively. W.l.o.g., we assume that the blue vertices $b_1, b_2, \ldots, b_k$ are labeled from left to right (according to the unique interval representation $\mathcal{I}$ of $G$), that is, $L(b_i) < L(b_j)$ if $i < j$; similarly, we assume that the red vertices $r_1, r_2, \ldots, r_k$ are labeled from left to right. Then, we define a target-assignment $g : \mathbf{I}_b \to \mathbf{I}_r$, as follows: for each blue vertex $b_i \in \mathbf{I}_b$

$$g(b_i) = r_i. \tag{1}$$

To prove Theorem 1, it suffices to show that $g$ is proper, and each token takes no detours.

*String representation:* By traversing the interval representation $\mathcal{I}$ of a connected proper interval graph $G$ from left to right, we can obtain a string $S = s_1 s_2 \cdots s_{2k}$ which is a superstring of both $b_1 b_2 \cdots b_k$ and $r_1 r_2 \cdots r_k$, that is, each letter $s_i$ in $S$ is one of the vertices in $\mathbf{I}_b \cup \mathbf{I}_r$ and $s_i$ appears in $S$ before $s_j$ if $L(s_i) < L(s_j)$. We

may assume without loss of generality that $s_1 = b_1$ since the reconfiguration rule is symmetric. If a vertex is in $\mathbf{I}_b \cap \mathbf{I}_r$ as $b_i$ and $r_j$, then we define that it appears as $b_i r_j$ in $S$. Then, for each $i$, $1 \leq i \leq 2k$, we define the *height* $h(i)$ *at* $i$ by the number of blue vertices appeared in the substring $s_1 s_2 \cdots s_i$ minus the number of red vertices appeared in $s_1 s_2 \cdots s_i$. For the sake of notational convenience, we define $h(0) = 0$. Then $h(i)$ can be recursively computed as follows:

$$h(i) = \begin{cases} 0 & \text{if } i = 0; \\ h(i-1) + 1 & \text{if } s_i \text{ is blue;} \\ h(i-1) - 1 & \text{if } s_i \text{ is red.} \end{cases} \tag{2}$$

Note that $h(2k) = 0$ for any string $S$ since $|\mathbf{I}_b| = |\mathbf{I}_r|$.

Using the notion of height, we split the string $S$ into substrings $S_1, S_2, \ldots, S_h$ at every point of height 0, that is, in each substring $S_j = s_{2p+1} s_{2p+2} \cdots s_{2q}$, we have $h(2q) = 0$ and $h(i) \neq 0$ for all $i$, $2p + 1 \leq i \leq 2q - 1$. Then, the substrings $S_1, S_2, \ldots, S_h$ form a partition of $S$, and each substring $S_j$ contains the same number of blue and red tokens. We call such a partition the *partition of* $S$ *at height* 0.

**Lemma 1.** *Let $S_j = s_{2p+1} s_{2p+2} \cdots s_{2q}$ be a substring in the partition of the string $S$ at height 0. Then, (a) the blue vertices $b_{p+1}, b_{p+2}, \ldots, b_q$ appear in $S_j$, and their corresponding red vertices $r_{p+1}, r_{p+2}, \ldots, r_q$ appear in $S_j$; (b) if $S_j$ starts with the blue vertex $b_{p+1}$, then each blue vertex $b_i$, $p + 1 \leq i \leq q$, appears in $S_j$ before its corresponding red vertex $r_i$; and (c) if $S_j$ starts with the red vertex $r_{p+1}$, then each blue vertex $b_i$, $p + 1 \leq i \leq q$, appears in $S_j$ after its corresponding red vertex $r_i$.*

*Algorithm:* Recall that we have fixed the unique interval representation $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of a connected proper interval graph $G$ so that $L(I_i) < L(I_{i+1})$ for each $i$, $1 \leq i \leq n - 1$, and each interval $I_i \in \mathcal{I}$ corresponds to the vertex $v_i \in V$. Since all intervals in $\mathcal{I}$ have unit length, the following proposition clearly holds.

**Proposition 1.** *For two vertices $v_i$ and $v_j$ in $G$ such that $i < j$, there is a path $P$ in $G$ which passes through only intervals (vertices) contained in $[L(I_i), R(I_j)]$. Furthermore, if $I_{i'} \cap I_i = \emptyset$ for some index $i'$ with $i' < i$, no vertex in $v_1, v_2, \ldots, v_{i'}$ is adjacent to any vertex in $P$. If $I_j \cap I_{j'} = \emptyset$ for some index $j'$ with $j < j'$, no vertex in $v_{j'}, v_{j'+1}, \ldots, v_n$ is adjacent to any vertex in $P$.*

Let $S$ be the string of length $2k$ obtained from two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a connected proper interval graph $G$, where $k = |\mathbf{I}_b| = |\mathbf{I}_r|$. Let $S_1, S_2, \ldots, S_h$ be the partition of $S$ at height 0. The following lemma shows that the tokens in each substring $S_j$ can always reach their corresponding red vertices. (Note that we sometimes denote simply by $S_j$ the set of all vertices appeared in the substring $S_j$, $1 \leq j \leq h$.)

**Lemma 2.** *Let $S_j = s_{2p+1} s_{2p+2} \cdots s_{2q}$ be a substring in the partition of $S$ at height 0. Then, there exists a reconfiguration sequence between $\mathbf{I}_b \cap S_j$ and $\mathbf{I}_r \cap S_j$*

*such that tokens are slid along edges only in the subgraph of $G$ induced by the vertices contained in $[L(s_{2p+1}), R(s_{2q})]$.*

**Proof of Theorem 1.** We now give an algorithm for sliding all tokens on the vertices in $\mathbf{I}_b$ to the vertices in $\mathbf{I}_r$. Recall that $S_1, S_2, \ldots, S_h$ are the substrings in the partition of $S$ at height 0. Intuitively, the algorithm repeatedly picks up one substring $S_j$, and slides all tokens in $\mathbf{I}_b \cap S_j$ to $\mathbf{I}_r \cap S_j$. By Lemma 2 it works locally in each substring $S_j$, but it should be noted that a token in $S_j$ may be adjacent to another token in $S_{j-1}$ or $S_{j+1}$ at the boundary of the substrings. To avoid this, we define a partial order over the substrings $S_1, S_2, \ldots, S_h$, as follows.

Consider any two consecutive substrings $S_j$ and $S_{j+1}$, and let $S_j = s_{2p+1}s_{2p+2} \cdots s_{2q}$. Then, the first letter of $S_{j+1}$ is $s_{2q+1}$. We first consider the case where both $s_{2q}$ and $s_{2q+1}$ are the same color. Then, since $s_{2q}$ and $s_{2q+1}$ are both in the same independent set of $G$, they are not adjacent. Therefore, by Proposition 1 and Lemma 2, we can deal with $S_j$ and $S_{j+1}$ independently. In this case, we do not define the ordering between $S_j$ and $S_{j+1}$. We next consider the case where $s_{2q}$ and $s_{2q+1}$ have different colors; in this case, we have to define their ordering. Suppose that $s_{2q}$ is blue and $s_{2q+1}$ is red; then we have $s_{2q} = b_q$ and $s_{2q+1} = r_{q+1}$. By Lemma 2 the token $t_q$ on $s_{2q}$ is slid to left, and the token $t_{q+1}$ will reach $r_{q+1}$ from right. Therefore, the algorithm has to deal with $S_j$ before $S_{j+1}$. Note that, after sliding all tokens $t_{p+1}, t_{p+2}, \ldots, t_q$ in $S_j$, they are on the red vertices $r_{p+1}, r_{p+2}, \ldots, r_q$, respectively, and hence the tokens in $S_{j+1}$ are not adjacent to any of them. By the symmetric argument, if $s_{2q}$ is red and $s_{2q+1}$ is blue, $S_{j+1}$ should be dealt with before $S_j$.

Such an ordering is defined only for two consecutive substrings $S_j$ and $S_{j+1}$, $1 \leq j \leq h-1$. Therefore, the partial order over the substrings is acyclic, and hence there exists a total order consistent with the partial order. The algorithm certainly slides all tokens from $\mathbf{I}_b$ to $\mathbf{I}_r$ according to this total order. Therefore, the target-assignment $g$ defined in Eq. (1) is proper, and hence $\mathbf{I}_b \vdash^* \mathbf{I}_r$.

We now discuss the length of reconfiguration sequences between $\mathbf{I}_b$ and $\mathbf{I}_r$, together with the running time of our algorithm.

**Proposition 2.** *For two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a connected proper interval graph $G$ with $n$ vertices, (1) the ordering of tokens to be slid in a shortest reconfiguration sequence between them can be computed in $O(n)$ time and $O(n)$ space, and (2) a shortest reconfiguration sequence between them can be output in $O(n^2)$ time and $O(n)$ space.*

This proposition also completes the proof of Theorem 1. □

It is remarkable that there exists an infinite family of instances for which any reconfiguration sequence requires $\Omega(n^2)$ length. Simple example is: $G$ is a path $(v_1, v_2, \ldots, v_{8k})$ of length $n = 8k$ for any positive integer $k$, $\mathbf{I}_b = \{v_1, v_3, v_5, \ldots, v_{2k-1}\}$, and $\mathbf{I}_r = \{v_{6k+2}, v_{6k+4}, \ldots, v_{8k}\}$. In this instance, each token $t_i$ must be slid $\Theta(n)$ times, and hence it requires $\Theta(n^2)$ time to output them all.

## 4 Caterpillars

The main result of this section is the following theorem.

**Theorem 2.** *The* SLIDING TOKEN *problem for a caterpillar $G = (V, E)$ and two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of $G$ can be solved in $O(n)$ time and $O(n)$ space, where $n = |V|$. Moreover, for a yes-instance, a shortest reconfiguration sequence between them can be output in $O(n^2)$ time and $O(n)$ space.*

Let $G = (S \cup L, E)$ be a caterpillar with spine $S$ which induces the path $(s_1, \ldots, s_m)$, and leaf set $L$. We assume that $m \geq 2$, $\deg(s_1) \geq 2$, and $\deg(s_m) \geq 2$. First we show that we can assume that each spine vertex has at most one leaf without loss of generality.

**Lemma 3.** *For any given caterpillar $G = (S \cup L, E)$ and two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ on $G$, there is a linear time reduction from them to another caterpillar $G' = (S' \cup L', E')$ and two independent sets $\mathbf{I}'_b$ and $\mathbf{I}'_r$ such that (1) $G$ with $\mathbf{I}_b$ and $\mathbf{I}_r$ is a yes-instance of the* SLIDING TOKEN *problem if and only if $G'$ with $\mathbf{I}'_b$ and $\mathbf{I}_r$ is a yes-instance of the* SLIDING TOKEN *problem, (2) the maximum degree of $G'$ is at most 3, and (3) $\deg(s_1) = \deg(s_m) = 2$, where $m = |S'|$. In other words, the* SLIDING TOKEN *problem on a caterpillar is sufficient to consider only caterpillars of maximum degree 3.*

Hereafter, we only consider the caterpillars stated in Lemma 3, and we denote the unique leaf of $s_i$ by $\ell_i$ if it exists. We here introduce a key notion of the problem on these caterpillars that is named *locked* path. Let $G$ and $\mathbf{I}$ be a caterpillar and an independent set of $G$, respectively. A path $P = (p_1, p_2, \ldots, p_k)$ on $G$ is *locked* by $\mathbf{I}$ iff (a) $k$ is odd and greater than 2, (b) $\mathbf{I} \cap P = \{p_1, p_3, p_5, \ldots, p_k\}$, (c) $\deg(p_1) = \deg(p_k) = 1$ (in other words, they are leaves), and $\deg(p_3) = \deg(p_5) = \cdots = \deg(p_{k-2}) = 2$. This notion is simplified version of a *locked* tree used in [3]. Using the discussion in [3], we obtain the condition for the immovable independent set on a caterpillar:

**Theorem 3 ([3]).** *Let $G$ and $\mathbf{I}$ be a caterpillar and an independent set of $G$, respectively. Then we cannot slide any token in $\mathbf{I}$ on $G$ at all if and only if there exist a set of locked paths $P_1, \ldots, P_h$ for some $h$ such that $\mathbf{I}$ is a union of them.*

The proof can be found in [3], and omitted here. Intuitively, for any caterpillar $G$ and its independent set $\mathbf{I}$, if $\mathbf{I}$ contains a locked path $P$, we cannot slide any token through the vertices in $P$. Therefore, $P$ splits $G$ into two subgraphs, and we obtain two completely separated subproblems. Therefore, we obtain the following lemma:

**Lemma 4.** *For any given caterpillar $G = (S \cup L, E)$ and two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ on $G$, there is a linear time reduction from them to another caterpillar $G' = (S' \cup L', E')$ and two independent sets $\mathbf{I}'_b$ and $\mathbf{I}'_r$ such that (1) $G$, $\mathbf{I}_b$, and $\mathbf{I}_r$ are a yes-instance of the* SLIDING TOKEN *problem if and only if $G'$, $\mathbf{I}'_b$, and $\mathbf{I}_r$ are a yes-instance of the* SLIDING TOKEN *problem, and (2) both of $\mathbf{I}'_b$ and $\mathbf{I}'_r$ contain no locked path.*

Hereafter, without loss of generality, we assume that the caterpillar $G$ with two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ satisfies the conditions in Lemmas 3 and 4. That is, each spine vertex $s_i$ has at most one leaf $\ell_i$, $s_1$ and $s_m$ have one leaf $\ell_1$ and

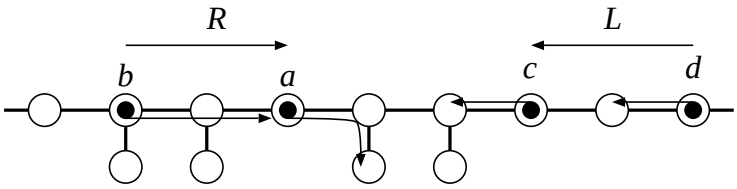


**Fig. 3.** The most right R token $a$ has to precede the most left L token $c$.

$\ell_m$, respectively, both of $\mathbf{I}_b$ and $\mathbf{I}_r$ contain no locked path, and $|\mathbf{I}_b| = |\mathbf{I}_r|$. Then, by the result in [3], this is a yes-instance. Thus, it is sufficient to show an $O(n^2)$ time algorithm that finds a shortest reconfiguration sequence between $\mathbf{I}_b$ and $\mathbf{I}_r$.

It is clear that each pair $(s_i, \ell_i)$ can have at most one token. Therefore, without loss of generality, we can assume that the blue vertices $b_1, b_2, \ldots, b_k$ in $\mathbf{I}_b$ (and the red vertices $r_1, r_2, \ldots, r_k$) are labeled from left to right according to the order $(s_1, \ell_1)$, $(s_2, \ell_2)$, …, $(s_m, \ell_m)$ of $G$. Then, by a similar argument for the proper interval graphs, we have $g(b_i) = r_i$ for each $i$. To prove Theorem 2, it suffices to show that we can move tokens with fewest detours by case analysis.

Now we introduce *direction* of a token $t$ denoted by $dir(t)$ as follows: when $t$ moves from $v_i \in \{s_i, \ell_i\}$ in $\mathbf{I}_b$ to $v_j \in \{s_j, \ell_j\}$ in $\mathbf{I}_r$ with $i < j$, the direction of $t$ is said to be $R$ and denoted by $dir(t) = R$. If $i > j$, it is said to be $L$ and denoted by $dir(t) = L$. If $i = j$, the direction of $t$ is said to be $C$ and denoted by $dir(t) = C$.

We first consider a simple case: all directions are either R or L. In this case, we can use the same idea appearing in the algorithm for a proper interval graph in Section 3. We can introduce a partial order over the tokens, and move them straightforwardly using the same idea in Section 3.Intuitively, a sequence of R tokens are moved from left to right, and a sequence of L tokens are moved from right to left, and we can define a partial order over the sequences of different directions. The only additional considerable case is shown in Fig. 3. That is, when the token $a$ moves to $\ell_i$ from left and the other token $c$ moves to $s_{i+1}$ from right, $a$ should precede $c$. It is not difficult to see that this (and its symmetric case) is the only exception than the algorithm in Section 3when all tokens move to right or left.

We next suppose that $\mathbf{I}_b$ (and hence $\mathbf{I}_r$) contains some token $t$ with $dir(t) = C$. In other words, $t$ is put on $s_i$ or $\ell_i$ for some $i$ in both of $\mathbf{I}_b$ and $\mathbf{I}_r$. We have five cases. Among them, here we consider the most complicated case that $t$ is put on $s_i$ in $\mathbf{I}_b$ and $\mathbf{I}_r$, and $\ell_i$ does not exist (the other cases are simpler, and omitted here). By assumption, $1 < s < m$ (since $\ell_1$ and $\ell_m$ exist). Without loss of generality, we suppose $t$ is the leftmost spine with the condition. We first observe that $|\mathbf{I}_b \cap \{s_{i-1}, \ell_{i-1}, s_{i+1}, \ell_{i+1}\}|$ is at most 1. Clearly, we have no token on $s_{i-1}$ and $s_{i+1}$. When we have two tokens on $\ell_{i-1}$ and $\ell_{i+1}$, the path $(\ell_{i-1}, s_{i-1}, s_i, s_{i+1}, \ell_{i+1})$ is a locked path, which contradicts the assumption. We also have $|\mathbf{I}_r \cap \{s_{i-1}, \ell_{i-1}, s_{i+1}, \ell_{i+1}\}| \leq 1$ by the same argument.

Now we consider the most serious case since the other cases are simpler and easier than this case. The most serious case is that $\mathbf{I}_b$ contains $\ell_{i-1}$ and $\mathbf{I}_r$ contains $\ell_{i+1}$. Since any token cannot bypass the other, $\mathbf{I}_b$ contains an L token on $\ell_{i-1}$, and $\mathbf{I}_r$ contains an L token on $\ell_{i+1}$. In this case, by the L token on $\ell_{i-1}$, first, $t$ should make a detour to right, and by the L token in $\mathbf{I}_r$, $t$ next should make a detour to left twice after the first detour. It is clear that this three slides should not be avoided, and this ordering of three slides cannot be violated. Therefore, $t$ itself should slide at least four times to return to the original position, and $t$ can done it in four slides. During this slides, since $t$ is the leftmost spine with this condition, the tokens on $s_1, \ell_1, s_2, \ell_2, \ldots, s_{i-1}, \ell_{i-1}$ do not make any detours. Thus we focus on the tokens on $s_{i+1}, \ell_{i+1}, \ldots$. Let $t'$ be the token that should be on $\ell_{i+1}$ in $\mathbf{I}_r$. Since $t$ is on $s_i$, $t'$ is not on $\{s_{i+1}, \ell_{i+1}\}$. If $t'$ is on one of $\ell_{i+2}, s_{i+3}, \ell_{i+3}, s_{i+4}, \ldots$ in $\mathbf{I}_b$, we have nothing to do; just make a detour for only $t$. The problem occurs when $t'$ is on $s_{i+2}$ in $\mathbf{I}_b$. If there exists $\ell_{i+2}$, we first slide $t'$ to it, and it is not difficult to see that this detour for $t'$ is unavoidable. If $\ell_{i+2}$ does not exist, we have to slide $t'$ to $s_{i+3}$ before slide of $t$. This can be done immediately except the similar situation that the only considerable case is that we have another L or S token $t''$ on $s_{i+3}$. We can repeat this process and confirm that each detour is unavoidable. Since $G$ with $\mathbf{I}_b$ and $\mathbf{I}_r$ contains no locked path, this process will halts. (More precisely, this process will be stuck if and only if these sequence of tokens forms a locked path on $G$, which contradicts the assumption.) Therefore, traversing this process, we can construct the shortest reconfiguration sequence.

**Proof of Theorem 2.** Using the algorithm in [3], we can decide if the input is a yes-instance. For a yes-instance, a shortest sequence can be constructed from the case analysis above. For each token, the number of detours made by the token is bounded above by $O(n)$, the number of slides of the token is also bounded above by $O(n)$, and the computation for the token can be done in $O(n)$ time. Therefore, the algorithm runs in $O(n^2)$ time, and the length of the shortest sequence is $O(n^2)$. (We note that, as shown before, there is a simple instance of the problem that requires a shortest sequence of length $\Theta(n^2)$.) □

## 5 Concluding Remarks

In this paper, we showed that the SHORTEST SLIDING TOKEN problem can be solved in polynomial time for three subclasses of interval graphs. The computational complexity of the problem for chordal graphs, interval graphs, and trees are still open. Especially, tree seems to be the next target from the viewpoint of finding a shortest sequence. We can decide if two independent sets are reconfigurable in linear time [3], then can we find a shortest sequence for a yes-instance? As in the 15-puzzle, finding a shortest one can be NP-hard even if the decision problem is polynomial time solvable. (In fact, the exact analysis for the 15-puzzle has been done up to $4 \times 4$ so far.) From the viewpoint of finding any sequence, the next target can be interval graphs. In general, it is an interesting open ques-

tion whether there is any instance on some graph classes whose reconfiguration sequence requires super-polynomial length.

## References


1. Bonsma, P., Kamiński, M., Wrochna M.: Reconfiguration Independent Sets in Claw-Free Graphs arXiv:1403.0359, 2014.
2. Brandstädg, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey, SIAM (1999)
3. Demaine, E.D., Demaine, M.L., Fox-Epstein, E., Hoang, D.A., Ito T., Ono, H., Otachi, Y., Uehara, R., Yamada, T.: Linear-Time Algorithm for Sliding Tokens on Trees. Theoretical Computer Science 600, pp. 132–142 (2015)
4. Deng, X., Hell, P., Huang., J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. SIAM J. Computing 25, pp. 390–403 (1996)
5. Fox-Epstein, E., Hoang, D.A., Otachi, Y., Uehara, R.: Sliding Token on Bipartite Permutation Graphs. The 26th International Symposium on Algorithms and Computation (ISAAC), accepted, 2015.
6. Gardner, M.: The Hypnotic Fascination of Sliding-Block Puzzles. Scientific American 210, pp. 122–130 (1964).
7. Gopalan, P., Kolaitis, P.G., Maneva, E.N., Papadimitriou, C.H.: The connectivity of Boolean satisfiability: computational and structural dichotomies. SIAM J. Computing 38, pp. 2330–2355 (2009)
8. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoretical Computer Science 343, pp. 72–96 (2005)
9. Hearn, R.A., Demaine, E.D.: Games, Puzzles, and Computation. A K Peters (2009)
10. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. Theoretical Computer Science 412, pp. 1054–1065 (2011)
11. Kamiński, M., Medvedev, P., Milanič, M.: Complexity of independent set reconfigurability problems. Theoretical Computer Science 439, pp. 9–15 (2012)
12. Korte, N., Möhring, R.: An incremental linear-time algorithm for recognizing interval graphs. SIAM J. Computing 18, pp. 68–81 (1989)
13. Makino, K., Tamaki, S., Yamamoto, M.: An exact algorithm for the Boolean connectivity problem for $k$-CNF. Theoretical Computer Science 412, pp. 4613–4618 (2011)
14. Mouawad, A.E., Nishimura, N., Pathak, V., Raman, V.: Shortest Reconfiguration Paths in the Solution Space of Boolean Formulas. In Proc. of ICALP 2015, LNCS 9134, pp. 985–996 (2015)
15. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the parameterized complexity of reconfiguration problems. In Proc. of IPEC 2013, LNCS 8296, pp. 281–294 (2013)
16. Mouawad, A.E., Nishimura, N., Raman, V., Wrochna, M.: Reconfiguration over tree decompositions. In Proc. of IPEC 2014, LNCS 8894, pp. 246–257 (2014)
17. Ratner, R., Warmuth, M.: Finding a shortest solution for the $N \times N$-extension of the 15-puzzle is intractable. J. Symb. Comp., Vol. 10, pp. 111–137, 1990.
18. Slocum, J.: The 15 Puzzle Book: How it Drove the World Crazy. Slocum Puzzle Foundation, 2006.
19. Yamada, T., Uehara, R.: Shortest Reconfiguration of Sliding Tokens on a Caterpillar. arXiv:1511.00243, November 1, 2015.