Reconfiguring k-Path Vertex Covers

Duc A. Hoang[1] and Akira Suzuki[2] and Tsuyoshi Yagita[3]

[1] Graduate School of Informatics, Kyoto University, Japan
hoang.duc.8r@kyoto-u.ac.jp
[2] Graduate School of Information Sciences, Tohoku University, Japan
akira@tohoku.ac.jp
[3] Graduate School of Computer Science and Systems Engineering,
Kyushu Institute of Technology, Japan
yagita.tsuyoshi307@mail.kyutech.jp

Abstract

A vertex subset I of a graph G is called a k-path vertex cover if every path on k vertices in G contains at least one vertex from I. The k-Path Vertex Cover Reconfiguration (k-PVCR) problem asks if one can transform one k-path vertex cover into another via a sequence of k-path vertex covers where each intermediate member is obtained from its predecessor by applying a given reconfiguration rule exactly once. We investigate the computational complexity of k-PVCR from the viewpoint of graph classes under the well-known reconfiguration rules: TS, TJ, and TAR. The problem for $k = 2$, known as the Vertex Cover Reconfiguration (VCR) problem, has been well-studied in the literature. We show that certain known hardness results for VCR on different graph classes can be extended for k-PVCR. In particular, we prove a complexity dichotomy for k-PVCR on general graphs: on those whose maximum degree is three (and even planar), the problem is PSPACE-complete, while on those whose maximum degree is two (i.e., paths and cycles), the problem can be solved in polynomial time. Additionally, we also design polynomial-time algorithms for k-PVCR on trees under each of TJ and TAR. Moreover, on paths, cycles, and trees, we describe how one can construct a reconfiguration sequence between two given k-path vertex covers in a yes-instance. In particular, on paths, our constructed reconfiguration sequence is shortest.

1 Introduction

Recently, a collection of problems called *Combinatorial Reconfiguration* has been extensively studied. Work in this research area specifically aims to model dynamic situations where one needs to transform one feasible solution of a computational problem into another by locally changing a solution while keeping its feasibility along the way. In a reconfiguration setting, two feasible solutions of a computational problem (e.g., Satisfiability, Independent Set, Vertex Cover, Dominating Set, etc.) are given,

along with a *reconfiguration rule* that describes an adjacency relation between solutions. A *reconfiguration problem* asks whether one feasible solution can be transformed into the other via a sequence of adjacent feasible solutions where each intermediate member is obtained from its predecessor by applying the given reconfiguration rule exactly once. Such a sequence, if exists, is called a *reconfiguration sequence*. One may recall the classic Rubik's cube puzzle as an example of a reconfiguration problem, where each configuration of the Rubik's cube corresponds to a feasible solution, and two configurations (solutions) are adjacent if one can be obtained from the other by rotating a face of the cube by either 90, 180, or 270 degrees. The question is whether one can transform an arbitrary configuration to the one where each face of the cube has only one color. For an overview of this research area, readers are referred to the recent surveys [17, 25, 24].

1.1 k-Path Vertex Cover Reconfiguration

Let $G = (V, E)$ be a simple graph. A *vertex cover* of G is a subset I of V where each edge contains at least one vertex from I. The Vertex Cover (VC) problem, which asks whether there is a vertex cover of G whose size is at most some positive integer s, is one of the classic NP-complete problems in the computational complexity theory [15].

Let $k \geq 2$ be a fixed positive integer. A subset I of V is called a *k-path vertex cover* if every path on k vertices in G contains at least one vertex from I. The k-Path Vertex Cover (k-PVC) problem asks if there is a k-path vertex cover of G whose size is at most some positive integer s. Motivated by the importance of a problem related to secure communication in wireless sensor networks, Brešar et al. initiated the study of k-PVC in [8] (as a generalized concept of *vertex cover*). It is known that k-PVC is NP-complete for every $k \geq 2$ [1, 8]. Subsequent work regarding the *maximum* variant [23] and *weighted* variant [9] of k-PVC has also been considered in the literature. Recently, the study of k-PVC and related problems has gained a lot of attention from both theoretical aspect [21, 26, 27] and practical application [3, 14].

In this paper, we initiate the study of k-PVC from the viewpoint of *combinatorial reconfiguration*. Given two distinct k-path vertex covers I and J of a graph G and a *single* reconfiguration rule, the k-Path Vertex Cover Reconfiguration (k-PVCR) problem asks whether there is a reconfiguration sequence between I and J. We study the computational complexity of k-PVCR with respect to different graph classes under the well-known reconfiguration rules: *Token Sliding*, *Token Jumping*, and *Token Addition or Removal*. They are informally defined as follows. Imagine that a token is placed at each vertex of a k-path vertex cover in G. For each of the following rules, a common requirement is that the resulting token-set forms a k-path vertex cover of G.

- Token Sliding (TS): A TS-step involves moving a token on some vertex v to one of its unoccupied neighbors.
- Token Jumping (TJ): A TJ-step involves moving a token on v to any unoccupied vertex.
- Token Addition or Removal (TAR): A TAR-step involves either adding or removing a single token such that the resulting token-set is of size at most given positive integer u. We sometimes write "TAR(u)" instead of "TAR" to emphasize

the upper bound u on the size of each token-set in a reconfiguration sequence under TAR.

1.2 Related Work

The *reoptimization* framework is closely related to *reconfiguration*. Roughly speaking, given an optimal solution of a problem instance $\mathcal{I}$, and some perturbations that change $\mathcal{I}$ into a new instance $\mathcal{I}'$, a reoptimization problem aims to find an optimal solution for the changed instance $\mathcal{I}'$. Recently, Kumar et al. [21] initiated the study of reoptimization problems for (both weighted and unweighted) k-PVC with $k \geq 3$, extending some known reoptimization paradigms for the well-known VC problem [2]. The perturbation they considered in [21] is changing the input graph of the current instance by inserting new vertices.

The Vertex Cover Reconfiguration (VCR) problem is one of the most well-studied reconfiguration problems, from both classical and parameterized complexity viewpoints (e.g., see [25] for a quick summary of known results). It is well-known that if I is a vertex cover of a graph $G = (V, E)$ then $V \setminus I$ is an *independent set* of G, i.e., a vertex subset whose members are pairwise non-adjacent. Consequently, from classical complexity viewpoint, results of Independent Set Reconfiguration (ISR) and Vertex Cover Reconfiguration are interchangeable.

We now mention some known complexity results of VCR (which are mostly interchanged with ISR) for some graph classes. It is well-known that VCR is PSPACE-complete under each of TS, TJ, and TAR for general graphs [18], planar graphs of maximum degree three [16, 19], perfect graphs [20], and bounded bandwidth graphs [28]. Even on bipartite graphs, VCR remains PSPACE-complete under TS, and NP-complete under each of TJ and TAR [22]. On chordal graphs (and even on split graphs), VCR is known to be PSPACE-complete under TS [4]. On the positive side, polynomial-time algorithms have been designed for VCR on even-hole-free graphs (and therefore chordal graphs) under each of TJ and TAR [20], on bipartite permutation graphs and bipartite distance-hereditary graphs [13] under TS, on cographs [6, 20], claw-free graphs [7], interval graphs [5, 20, 10], and trees [11, 20] under each of TS, TJ, and TAR.

1.3 Our Results

In this paper, we investigate the complexity of k-PVCR with respect to different input graphs (see Figure 1). More precisely, we show that:

- Several hardness results for VCR remain true for k-PVCR. More precisely, we show the PSPACE-completeness of k-PVCR on general graphs under each rule TS, TJ, and TAR using a reduction from a variant of VCR. As our reduction preserves some nice graph properties, we claim (as a consequence of our reduction) that the hardness results for VCR on several graphs (namely planar graphs, bounded bandwidth graphs, chordal graphs) can be converted into those for k-PVCR. Using a reduction from the Nondeterministic Constraint Logic [16, 29], we also show that k-PVCR remains PSPACE-complete even on planar graphs of

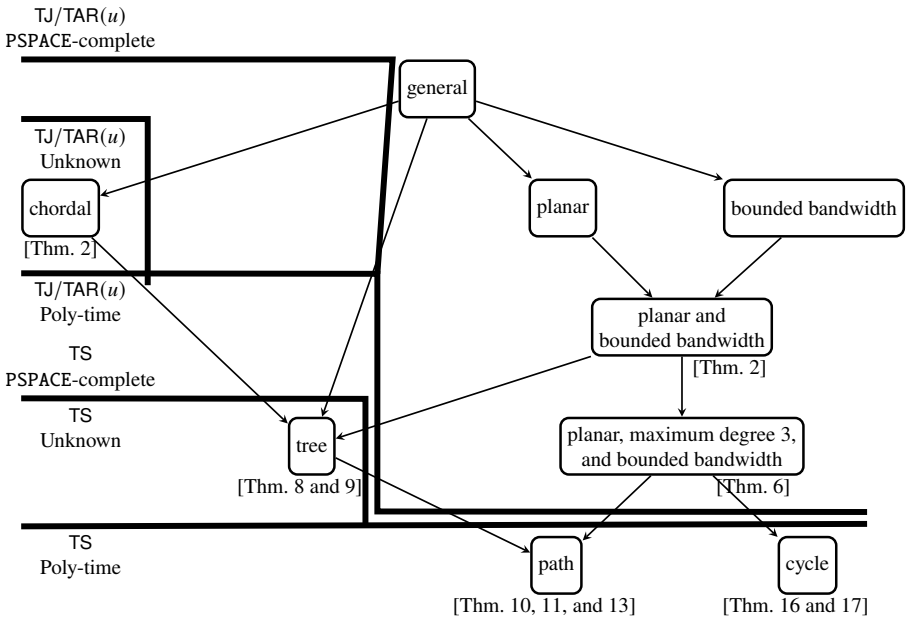


Figure 1: Computational complexity of k-PVCR on some graph classes, under each of TS, TJ, and TAR(u). Each arrow from A to B means B is a subclass of A.

bounded bandwidth and maximum degree three. (Our reduction from VCR does not preserve the maximum degree.)

- On the positive side, we design polynomial-time algorithms for k-PVCR on some restricted graph classes: trees (under each of TJ and TAR), paths and cycles (under each of TS, TJ, and TAR). Our algorithms are constructive, i.e., we explicitly show how a reconfiguration sequence can be constructed in a yes-instance. On paths, we claim that our algorithm constructs a *shortest* reconfiguration sequence. As a result, we obtain a complexity dichotomy for k-PVCR on (planar) graphs with respect to their maximum degree.

2 Preliminaries

In this section, we define some useful notation and terminology. For standard concepts on graphs, we refer readers to [12].

Let G be a simple graph with vertex-set $V(G)$ and edge-set $E(G)$. For two vertices u, v, we denote by $\mathrm{dist}_G(u, v)$ the *distance* between u and v in G, i.e., the number of edges in a shortest path between them. For a vertex $v \in V(G)$, we denote by $G - v$ the graph obtained from G by removing the vertex v and all incident edges. For two vertex subsets I and J, we denote by $G[I \Delta J]$ the subgraph of G induced by their *symmetric difference* $I \Delta J = (I \setminus J) \cup (J \setminus I)$. For a fixed integer $k \geq 2$, we say that a vertex v

covers a k-path (i.e., a path on k vertices) P_k in G if $v \in V(P_k)$. A vertex subset I is called a *k-path vertex cover* if every k-path in G contains at least one vertex from I. In other words, vertices of I cover all k-paths in G. We denote by $\psi_k(G)$ the size of a *minimum* k-path vertex cover of G. Trivially, for $n \geq k \geq 2$, $\psi_k(P_n) = \lfloor n/k \rfloor$ and $\psi_k(C_n) = \lceil n/k \rceil$ for a path P_n and a cycle C_n on n vertices.

Throughout this paper, we denote by (G, I, J, R) an instance of k-PVCR under a reconfiguration rule $\mathsf{R} \in \{\mathsf{TJ}, \mathsf{TS}, \mathsf{TAR}\}$, where I and J are two k-path vertex covers of G. We shall respectively call a reconfiguration sequence under each of TS, TJ, and TAR by a TS-*sequence*, TJ-*sequence*, and TAR(u)-*sequence*. Formally, let $S = \langle I_0, I_1, \ldots, I_\ell \rangle$ be an *ordered* sequence of k-path vertex covers of G. The *length* of S is defined as ℓ, i.e., if S is a reconfiguration sequence then its length is exactly the number of steps it performs under the given reconfiguration rule. Imagine that a token is placed at each vertex of a k-path vertex cover of G. We may sometimes identify a token with the vertex where it is placed and say "a token in a k-path vertex cover", and therefore use the terms "token-set" and "k-path vertex cover" interchangeably. We say that S is a TS-*sequence* between two k-path vertex covers I_0 and I_ℓ if for each $i \in \{0, \ldots, \ell - 1\}$, there exist two vertices x_i and y_i such that $I_i \setminus I_{i+1} = \{x_i\}$, $I_{i+1} \setminus I_i = \{y_i\}$, and $x_i y_i \in E(G)$. Roughly speaking, I_{i+1} is obtained from I_i by sliding the token placed on x_i to y_i along an edge $x_i y_i$. Similarly, we say that S is a TJ-*sequence* between I_0 and I_ℓ if for each $i \in \{0, \ldots, \ell - 1\}$, there exist two vertices x_i and y_i such that $I_i \setminus I_{i+1} = \{x_i\}$, $I_{i+1} \setminus I_i = \{y_i\}$. Intuitively, I_{i+1} is obtained from I_i by jumping the token placed on x_i to y_i. Now, if $\max\{|I_i| : 0 \leq i \leq \ell\} \leq u$ for some positive integer u, and for each $i \in \{0, \ldots, \ell - 1\}$, there exists a vertex x_i such that $I_i \Delta I_{i+1} = \{x_i\}$ then we say that S is a TAR(u)-sequence between I_0 and I_ℓ. Roughly speaking, I_{i+1} is obtained from I_i by either adding a token to x_i or removing a token from x_i. If a TS-, TJ-, or TAR(u)-sequence between two k-path vertex covers I and J exists, we say that I and J are *reconfigurable* under TS, TJ, or TAR, respectively.

Using a similar argument as in [20, Theorem 1], we can prove the following useful lemma.

Lemma 1. *There exists a* TJ*-sequence of length ℓ between two k-path vertex covers I, J of a graph G with $|I| = |J| = s$ if and only if there exists a* TAR($s+1$)*-sequence of length 2ℓ between them.*

A reconfiguration sequence of minimum length is called a *shortest* reconfiguration sequence. For a reconfiguration sequence $S = \langle I_0, I_1, \ldots, I_p \rangle$, we denote by $\mathrm{rev}(S)$ the *reverse* of S, i.e., the reconfiguration sequence $\langle I_p, \ldots, I_1, I_0 \rangle$. For two reconfiguration sequences $S = \langle I_0, I_1, \ldots, I_p \rangle$ and $S' = \langle I'_0, I'_1, \ldots, I'_q \rangle$ under the same reconfiguration rule, if $I_p = I'_0$ then we say that they can be *concatenated* and define their *concatenation* $S \oplus S'$ as the reconfiguration sequence $\langle I_0, I_1, \ldots, I_p, I'_1, \ldots, I'_q \rangle$. We assume for convenience that if S' is empty then $S \oplus S' = S' \oplus S = S$.

3 Hardness Results

3.1 Reduction from Vertex Cover Reconfiguration

In this section, we prove the following theorem using a polynomial-time reduction from VCR.

Theorem 2. *k-PVCR is* PSPACE*-complete under each of* TS*,* TJ*, and* TAR *even when the input graph is a planar graph of maximum degree four, or a bounded bandwidth graph. Additionally, k-PVCR is* PSPACE*-complete under* TS *on chordal graphs.*

The outline of our proof is as follows:

(1) In Lemma 3, using a reduction similar to that in [8], we show the PSPACE-completeness of k-PVCR under TJ.

(2) In Lemma 4, we combine (1), the known results for VCR, and Lemma 1 to show the hardness results on several graphs under each of TJ and TAR as mentioned in Theorem 2.

(3) Finally, in Lemma 5, we show that the hardness results under TS hold via the same reduction.

Lemma 3. *k-PVCR is* PSPACE*-complete under* TJ.

Proof. Given two distinct *minimum* k-path vertex covers I and J of a graph G and a single reconfiguration rule, the Minimum k-Path Vertex Cover Reconfiguration (Min-k-PVCR) problem asks whether there is a reconfiguration sequence between I and J. For $k = 2$, the Min-k-PVCR problem is also known as Minimum Vertex Cover Reconfiguration (Min-VCR).

Clearly, since k-Path Vertex Cover is in NP [8], it follows from [18] that k-PVCR is in PSPACE. Since k-PVCR is more general than Min-k-PVCR, in order to show the PSPACE-completeness of k-PVCR, it suffices to reduce from the Min-VCR problem (which is known to be PSPACE-complete [18]) to the Min-k-PVCR problem. More precisely, given an instance (G, I, J, TJ) of Min-VCR, we construct a corresponding instance $(G', I', J', \mathsf{TJ})$ of Min-k-PVCR as follows. Let G' be the graph obtained from G by joining each vertex x of G to an endpoint of a new path P^x on $\lfloor (k-1)/2 \rfloor$ vertices. We choose $I' = I$ and $J' = J$. Note that each vertex cover of G is also a k-path vertex cover of G', Moreover, for any minimum k-path vertex cover I' of G', if I' contains a new vertex y in a path P^x for some vertex x of G then $(I' \setminus \{y\}) \cup \{x\}$ is also a minimum k-path vertex cover of G', because any k-path covered by y must also be covered by x. Consequently, $(G', I', J', \mathsf{TJ})$ is an instance of Min-k-PVCR.

It is clear that this construction can be done in polynomial time. It remains to show that (G, I, J, TJ) is a yes-instance of Min-VCR if and only if $(G', I', J', \mathsf{TJ})$ is a yes-instance of Min-k-PVCR.

Assume that (G, I, J, TJ) is a yes-instance of Min-VCR, that is, there exists a TJ-sequence $\langle I = I_0, I_1, \ldots, I_p = J \rangle$ between I and J in G. Clearly, for any $i \in \{0, 1, \ldots, p\}$, the set I_i is also a minimum k-path vertex cover of G'. Then, $\langle I = I_0, I_1, \ldots, I_p = J \rangle$ is also a TJ-sequence between $I' = I$ and $J' = J$ in G'.

Now, assume that $(G', I', J', \mathsf{TJ})$ is a yes-instance of Min-k-PVCR in G', that is, there exists a TJ-sequence $S = \langle I' = I'_0, I'_1, \ldots, I'_q = J' \rangle$ between $I' = I$ and $J' = J$ in G'. We claim that (G, I, J, TJ) is also a yes-instance by constructing a TJ-sequence between I and J in G. For $i \in \{0, 1, \ldots, q\}$, let $I_i = I'_i \setminus \bigcup_{x \in V(G)} V(P^x) \cap \bigcup_{x \in V(G)} \{x : I'_i \cap V(P^x) \neq \emptyset\}$. Intuitively, I_i is obtained from I'_i by moving each token placed at some new vertex in P^x to x itself. Since any k-path covered by some vertex in P^x is also covered by x, and each I'_i is minimum, such moves are well-defined. Clearly, each I_i is a minimum vertex cover of G. For $i \in \{0, 1, \ldots, q-1\}$, let x'_i and y'_i be two distinct vertices of G' such that $I'_i \setminus I'_{i+1} = \{x'_i\}$ and $I'_{i+1} \setminus I'_i = \{y'_i\}$. Next, we will show that I_{i+1} can be obtained from I_i by performing at most one TJ-step in G.

- **Case 1:** $x'_i \in V(G)$ **and** $y'_i \in V(G)$. By definition, $I_i \setminus I_{i+1} = \{x'_i\}$ and $I_{i+1} \setminus I_i = \{y'_i\}$.
- **Case 2:** $x'_i \in V(G)$ **and** $y'_i \in V(G') \setminus V(G)$**.** Then, y'_i must belong to a new path P^y joined to some vertex $y \in V(G)$. By definition, $I_i \setminus I_{i+1} = \{x'_i\}$ and $I_{i+1} \setminus I_i = \{y\}$. Note that if $x'_i = y$, then $I_i = I_{i+1}$, and we are done. Moreover, as we consider minimum k-path vertex covers, $y \notin I'_i \setminus \{x'_i\}$ and therefore $y \notin I_i$; otherwise, we cannot move the token on x'_i to y'_i.
- **Case 3:** $x'_i \in V(G') \setminus V(G)$ **and** $y'_i \in V(G)$**.** As before, x'_i must belong to a new path P^x joined to some vertex $x \in V(G)$. By definition, $I_i \setminus I_{i+1} = \{x\}$ and $I_{i+1} \setminus I_i = \{y'_i\}$. Note that if $x = y'_i$, then $I_i = I_{i+1}$.
- **Case 4:** $x'_i \in V(G') \setminus V(G)$ **and** $y'_i \in V(G') \setminus V(G)$**.** As before, x'_i (resp. y'_i) must belong to a new path P^x (resp. P^y) joined to some vertex $x \in V(G)$ (resp. $y \in V(G)$). By definition, $I_i \setminus I_{i+1} = \{x\}$ and $I_{i+1} \setminus I_i = \{y\}$. Note that if $x = y$, then $I_i = I_{i+1}$.

Clearly, the sequence obtained from $\langle I_0, I_1, \ldots, I_q \rangle$ by removing redundant vertex covers (i.e., those equal to their predecessors) is a TJ-sequence in G that reconfigures $I = I_0$ to $J = I_q$. □

Lemma 4. *k-PVCR is* PSPACE*-complete under each of* TJ *and* TAR *on planar graphs of maximum degree four and bounded bandwidth.*

Proof. As we mention in Section 1.2, it is known that VCR is PSPACE-complete under each of TJ, and TAR for planar graphs of maximum degree three [19], and bounded bandwidth graphs [28]. In fact, these results are also hold in the case MIN-VCR [19, 28]. It is not hard to see that in the reduction presented in the proof of Lemma 3, if the input graph G is one of the mentioned graphs, then so is the constructed graph G'. (In fact the bandwidth of G' is $O(k)$. However, since we defined that k is a fixed integer, G' is of bounded bandwidth.) The hardness results under TAR are followed by combining the known results for Vertex Cover Reconfiguration, the above results, and Lemma 1. □

Lemma 5. *k-PVCR is* PSPACE*-complete under* TS *on planar graphs of maximum degree four and bounded bandwidth, and chordal graphs.*

Proof. It is not hard to see that in the reduction presented in the proof of Lemma 3, if the input graph G is one of the mentioned graphs, then so is the constructed graph G'.

It is sufficient to show that any TJ-sequence $S = \langle I_0, I_1, \ldots, I_q \rangle$ between two minimum k-path vertex covers $I = I_0$ and $J = I_q$ of the constructed graph G' can be converted into a TS-sequence between them in G'.

First of all, if $I_i \subseteq V(G)$ for all $i \in \{0, 1, \ldots, q\}$ then we claim that S itself is indeed a TS-sequence. More precisely, we show that for each $i \in \{0, 1, \ldots, q-1\}$, if x_i and y_i are two distinct vertices of G such that $I_i \setminus I_{i+1} = \{x_i\}$ and $I_{i+1} \setminus I_i = \{y_i\}$ then $x_i y_i \in E(G) \subseteq E(G')$. Suppose to the contrary that y_i is not adjacent to x_i. We note that each I_i ($i \in \{0, 1, \ldots, q\}$) is also a minimum vertex cover of G. Now, in order to move the token on x_i to y_i for obtaining a new vertex cover I_{i+1} of G, each edge of G incident with x_i must already be covered by its other endpoint; otherwise, moving x_i to y_i left some non-covered edge. However, this means that one can obtain a vertex cover of smaller size by simply removing x_i from I_i, which contradicts the fact that I_i is minimum. Therefore, y_i must be a neighbor of x_i.

Now, from the above reduction, we know that there is always a TJ-sequence S' between two k-path vertex covers $I' = I \setminus \bigcup_{x \in V(G)} V(P^x) \cap \bigcup_{x \in V(G)} \{x : I \cap V(P^x) \neq \emptyset\}$ and $J' = J \setminus \bigcup_{x \in V(G)} V(P^x) \cap \bigcup_{x \in V(G)} \{x : J \cap V(P^x) \neq \emptyset\}$, where all members of S' are subsets of $V(G)$. Here P^x denotes the new path joined to the vertex $x \in V(G)$. As a result, S' is also a TS-sequence in G'. To construct a TS-sequence between I and J, it suffices to show that one can construct a TS-sequence S'' between I and I' in G'. In a similar manner, we will be able to construct a TS-sequence between J and J', and a TS-sequence between I and J can be formed by simply reconfiguring I to I', then I' to J', and finally J' to J. Let $x \in V(G)$ be such that $I \cap V(P^x) = \{x'\}$. Since I is a minimum k-path vertex cover of G', we have $x \notin I$. We claim that I can be reconfigured to $I \setminus \{x'\} \cup \{x\}$ using TS-steps. Let $P = v_0 v_1 \ldots v_\ell$ ($0 \leq \ell \leq \lfloor (k-1)/2 \rfloor$) be the unique path in G' joining $v_0 = x$ and $v_\ell = x'$. Note that for each $j \in \{1, \ldots, \ell\}$, any k-path covered by v_j is also covered by each vertex in $\{v_0, \ldots, v_{j-1}\}$. Moreover, as we consider minimum k-path vertex covers, exactly one of v_j ($j \in \{0, 1, \ldots, \ell\}$) contains a token. Hence, one can obtain $I \setminus \{x'\} \cup \{x\}$ from I by simply sliding the token on $x' \in I$ to x along the path P. Applying this process repeatedly for each $x \in V(G)$ where $I \cap V(P^x) \neq \emptyset$, we obtain a TS-sequence in G' between I and I'. □

Our proof of Theorem 2 is complete.

3.2 Reduction from Nondeterministic Constraint Logic

In Theorem 2, we show the PSPACE-completeness for planar graphs of maximum degree four. Furthermore, using a reduction from the Nondeterministic Constraint Logic [16, 29] (NCL, for short), we can improve this result as follows.

Theorem 6. *k-PVCR remains* PSPACE*-complete under each of* TS*,* TJ*, and* TAR *even on planar graphs of bounded bandwidth and maximum degree three.*

In this section, we briefly define NCL and show the cases for TS and TJ, because the case for TAR can be shown similar to the proof of Lemma 1. This result can be obtained by constructing polynomial-time reductions from NCL—a well-known

PSPACE-complete problem first introduced by Hearn and Demaine [16]. This problem is often used to prove the computational hardness of puzzles and games, because a reduction from this problem requires to construct only two types of gadgets, called AND and OR gadgets.

3.2.1 NONDETERMINISTIC CONSTRAINT LOGIC

Now we define NCL problem [16]. An NCL "machine" is an undirected graph together with an assignment of weights from $\{1, 2\}$ to each edge of the graph. An (*NCL*) *configuration* of this machine is an orientation (direction) of the edges such that the sum of weights of in-coming arcs at each vertex is at least two. Figure 2(a) illustrates a configuration of an NCL machine, where each weight-2 edge is depicted by a thick (blue) line and each weight-1 edge by a thin (red) line. Then, two NCL configurations are *adjacent* if they differ in a single edge direction. Given an NCL machine and its two configurations, it is known to be PSPACE-complete to determine whether there exists a sequence of adjacent NCL configurations which transforms one into the other [16].

An NCL machine is called an *AND/OR constraint graph* if it consists of only two types of vertices, called "NCL AND vertices" and "NCL OR vertices" defined as follows:

- A vertex of degree three is called an *NCL AND vertex* if its three incident edges have weights 1, 1 and 2. (See Figure 2(b).) An NCL AND vertex u behaves as a logical AND, in the following sense: the weight-2 edge can be directed outward for u if and only if both two weight-1 edges are directed inward for u. Note that, however, the weight-2 edge is not necessarily directed outward even when both weight-1 edges are directed inward.

- A vertex of degree three is called an *NCL OR vertex* if its three incident edges have weights 2, 2 and 2. (See Figure 2(c).) An NCL OR vertex v behaves as a logical OR: one of the three edges can be directed outward for v if and only if at least one of the other two edges is directed inward for v.

It should be noted that, although it is natural to think of NCL AND/OR vertices as having inputs and outputs, there is nothing enforcing this interpretation; especially for NCL OR vertices, the choice of input and output is entirely arbitrary because an NCL OR vertex is symmetric.

For example, the NCL machine in Figure 2(a) is an AND/OR constraint graph. From now on, we call an AND/OR constraint graph simply an *NCL machine*, and call an edge

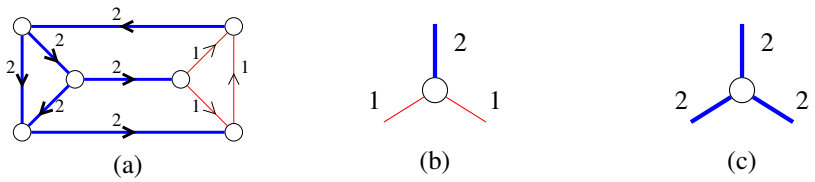


Figure 2: (a) A configuration of an NCL machine, (b) NCL AND vertex, and (c) NCL OR vertex.

in an NCL machine an *NCL edge*. NCL remains PSPACE-complete even if an input NCL machine is planar and bounded bandwidth [29].

3.2.2 Constructing gadgets

In our reduction, we construct two types of gadgets named *AND/OR gadgets*, which correspond to NCL AND/OR vertices, respectively. Both AND/OR gadgets consist of one *main part* and three *connecting parts*. Each connecting part corresponds to each incident NCL edge of the corresponding vertex. Then we replace each of vertices in the NCL machine with its corresponding gadget so that each pair of adjacent vertices sharing their connecting parts.

Each connecting part is formed P_{2k-2}. Note that if we want to cover this path with only one vertex, we must choose one of the two center vertices. In our reduction, choosing one of the two vertices corresponds to inward direction, and the other one corresponds to outward direction.

Now we explain the construction of the AND gadget. Consider an NCL AND vertex. Figure 4(a) illustrates all valid orientations of the edges incident to an NCL AND vertex. Two boxes are joined by an edge if their orientations are adjacent. We construct our AND gadget so that it correctly simulates this reconfiguration graph in Fig. 4(a).

Figure 3(a) illustrates our AND gadget for the case where $k = 3$. The main part of AND gadget forms P_k. Note that we must choose at least one of the vertices on this part to obtain k-PVC. Then we connect one endpoint to two connecting parts which corresponds to weight-1 edges, and connect the other endpoint to a connecting part which corresponds to weight-2 edge. If at least one of the weight-1 edges is directed outward, we must choose the endpoint of main part next to the connecting part corresponding to the weight-1 edge to obtain k-PVC. On the other hand, if the weight-2 edge is directed outward, we must choose the endpoint of main part next to the connecting part corresponding to the weight-2 edge to obtain k-PVC. Fig. 4(b) illustrates the reconfiguration graph for all 3-PVCs of the AND gadget where we allow to choose at most four vertices as 3-PVC. Each large dashed box surrounds all 3-PVCs choosing the same vertices from their connecting part. Then we can see that these 3-PVCs are "internally connected," that is, any two 3-PVCs in the same dashed box are reconfigurable with each other without changing the vertices in connecting parts. Furthermore, this gadget preserves the "external adjacency" in the following sense: if we contract the 3-PVCs in the same dashed box in Fig. 4(b) into a single vertex, then the resulting graph is exactly the graph depicted in Fig. 4(a). Therefore, we can conclude that our AND gadget correctly works as an NCL AND vertex.

Next we explain the construction of OR gadget. Figure 3(b) illustrates our OR gadget for the case where $k = 3$. The main part of OR gadget forms C_{k+1} (cycle consists of $k+1$ vertices). Note that we must choose at least two of the vertices on this part to obtain k-PVC. Then we arbitrary choose three distinct vertices from this cycle and connect them to three connecting parts one by one. If a weight-2 edge is directed outward, we must choose the vertex in main part next to the connecting part corresponding to the edge to obtain k-PVC. To verify that this OR gadget correctly simulates an NCL OR vertex, it suffices to show that this gadget satisfies both the internal connectedness and the external adjacency. Since this gadget has only 18 3-PVCs where we allow to

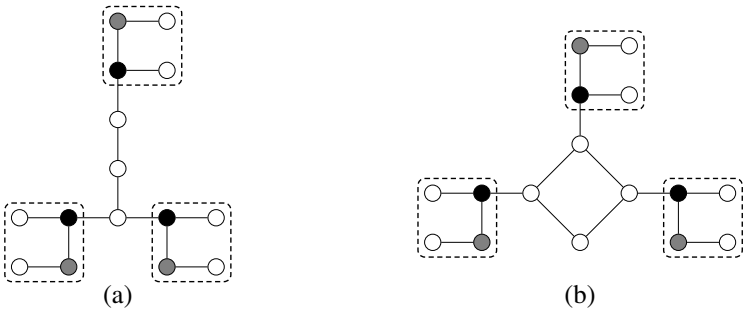


Figure 3: Gadgets for 3-PVCR. (a) The AND gadget. (b) The OR gadget. Each dashed rectangle represents a connecting part. For each connecting part, choosing the black vertex corresponds to inward direction, and the gray vertex corresponds to outward direction.

choose at most five vertices as 3-PVC. Therefore, by same way to AND gadget, we can easily check these sufficient conditions. (See Fig. 5.)

3.2.3 Reduction

As we have explained before, we replace each of NCL AND/OR vertices with its corresponding gadget; let G be the resulting graph. Recall that NCL remains PSPACE-complete even if an input NCL machine is planar and bounded bandwidth [29]. Since both our gadgets are planar, consist of only a constant number of edges, and of maximum degree three, the resulting graph G is also planar, bounded bandwidth and of maximum degree three. (In fact the number of edges in our gadget is $O(k)$. However, since we defined that k is a fixed integer, it becomes constant.)

In addition, we construct two k-PVCs of G which correspond to two given NCL configurations of the NCL machine. Note that there are (in general, exponentially) many k-PVCs which correspond to the same NCL configuration. However, by the construction of the gadgets, no two distinct NCL configurations correspond to the same k-PVC of G. Therefore, we arbitrarily choose two k-PVCs of G which correspond to two given NCL configurations.

This completes the construction of our corresponding instance of k-PVCR. Clearly the construction can be done in polynomial time.

3.2.4 Correctness

Let C_I and C_J be two given NCL configurations of the NCL machine. Let I and J be two k-PVCs of G which correspond to C_I and C_J, respectively. We now prove that there exists a desired sequence of NCL configurations between C_I and C_J if and only if there exists a reconfiguration sequence between I and J.

We first prove the only-if direction. Suppose that there exists a desired sequence $S = \langle C_0, C_1, \ldots, C_\ell \rangle$ of NCL configurations between $C_0 = C_I$ and $C_\ell = C_J$. Consider any two adjacent NCL configurations C_{i-1} and C_i in the sequence. Then only one

NCL edge vw changes its orientation between C_{i-1} and C_i. Notice that, since both C_{i-1} and C_i are valid NCL configurations, the NCL AND/OR vertices v and w have enough in-coming NCL edges even without vw. Recall that both AND/OR gadgets are internally connected and preserve the external adjacency. Therefore, any reversal of an NCL edge can be simulated by a reconfiguration sequence of k-PVCs of G, and hence there exists a reconfiguration sequence between I and J.

We now prove the if direction. Suppose that there exists a reconfiguration sequence $S = \langle I_0, I_1, \ldots, I_\ell \rangle$ ($I_0 = I$ and $I_\ell = J$). Notice that, by the construction of gadgets, any k-PVC of G corresponds to a valid NCL configuration. Let C_i be an NCL configuration corresponds to I_i, for $i \in \{0, \ldots, \ell\}$. By deleting redundant orientations from $C_0, C_1, \ldots, C_\ell$ if needed, we can obtain a sequence of valid adjacent orientations between C_I and C_J.

This completes the proof of Theorem 6.

4 Polynomial-Time Algorithms

4.1 Trees

In this section, we show polynomial-time algorithms for k-PVCR on trees under each of TJ and TAR. We first show a polynomial-time algorithm for the problem under TJ. Then, using Lemma 1 and the above result, we show a polynomial-time algorithm for the problem under TAR.

First, in order to solve the problem under TJ, we claim that for an instance (T, I, J, TJ) of k-PVCR on a tree T, if $|I| = |J|$, one can construct in polynomial time a TJ-sequence between I and J. The idea is to construct a canonical k-path vertex cover $I^\star$ such that both I and J can be reconfigured to $I^\star$ under TJ.

Before constructing $I^\star$, we prove the following lemma, which describes an useful algorithm for partitioning a tree into subtrees satisfying certain conditions.

Lemma 7. *Let T be a tree on n vertices rooted at a vertex r. Assume that $\psi_k(T) \geq 1$. Then, in $O(n)$ time, one can partition T into $\psi_k(T)$ subtrees $T_1(r), \ldots, T_{\psi_k(T)}(r)$ such that for each $i \in \{1, \ldots, \psi_k(T)\}$,*

(i) Each k-path vertex cover I satisfies $I \cap V(T_i(r)) \neq \emptyset$.

(ii) There is a vertex that covers all k-paths in $T_i(r)$.

Proof. To construct a partition $P(T) = \{T_1(r), \ldots, T_{\psi_k(T)}(r)\}$ of T satisfying the described conditions, we slightly modify the algorithm `PVCPTree`(T, k) in [8] as follows. A *properly rooted subtree* T_v of T is a subtree of T induced by the vertex v and all its descendants (with respect to the root r) satisfying the following conditions

1. T_v contains a k-path;
2. $T_v - v$ does not contain a k-path.

The modified algorithm `Partition`(T, k, r) systematically searches for a properly rooted tree T_v, decides whether T_v belongs to a solution $P(T)$, and if so, adds T_v to

Algorithm 1: `Partition`(T, k, r).

Input: A tree T on n vertices rooted at r and a positive integer k;
Output: A partition $P(T)$ of T into ψ_k(T) subtrees;

```
1  i := 1;
2  while T contains a properly rooted subtree T_v do
3  |  if T − T_v contains a properly rooted subtree then
4  |  |  T_i(r) := T_v;
5  |  |  i := i + 1;
6  |  else
7  |  |  T_i(r) := T;
8  |  T := T − T_v;
9  P(T) = {T_1(r), ..., T_i(r)};
10 return P(T);
```

$P(T)$, and removes T_v from the input tree T. To check if T contains a properly rooted subtree T_v, one can start by assigning v to a vertex of largest *depth* (i.e., distance from r) and verify if T_v is properly rooted. If so, we return "yes". Otherwise, we assign v to its parent and repeat, until a T_v is found (returning "yes") or there is nothing to check (returning "no").

From [8], it follows that `Partition`(T, k, r) runs in $O(n)$ time. From the construction of $P(T)$, it is clear that (i) always holds. We show (ii) by induction on $\psi_k(T)$.

For a tree T with $\psi_k(T) = 1$, let T_v be a properly rooted subtree of T. Since any k-path vertex cover of T contains a vertex from T_v, it follows that $\psi_k(T - T_v) = \psi_k(T) - 1 = 0$, which implies that $T - T_v$ does not contain any properly rooted subtree, and therefore $P(T) = \{T\}$. To see that (ii) holds, note that v must cover all k-paths in T_v, and therefore it also covers all k-paths in T; otherwise, $T - T_v$ contains a k-path that is not covered by v, and then must contain a properly rooted subtree, which is a contradiction.

Assume that (ii) holds for any tree T with $\psi_k(T) < c$, for some constant $c > 1$. For a tree T rooted at some vertex r with $\psi_k(T) = c$, let T_v be a properly rooted subtree of T, where v is some vertex of T. From the algorithm `Partition`, it follows that v must cover all k-paths in $T_v = T_1(r)$. Since $c > 1$, the tree $T - T_v$ contains a properly rooted subtree. By inductive hypothesis, for each $i \in \{2, 3, \ldots, \psi_k(T)\}$, there is a vertex that covers all k-paths in $T_i(r)$. Therefore, (ii) holds for any tree T with $\psi_k(T) \geq 1$. □

We are now ready to show the following theorem.

Theorem 8. *For any instance (T, I, J, TJ) of k-PVCR on a tree T, I and J are reconfigurable if and only if $|I| = |J|$. Moreover, a reconfiguration sequence between them, if exists, can be constructed in $O(n)$ time. Consequently, k-PVCR under* TJ *can be solved in linear time on trees.*

Proof. Clearly, if I and J are reconfigurable under TJ, they must be of the same size. To prove this theorem, it suffices to show that for an instance (T, I, J, TJ) of k-PVCR on a tree T, one can construct in polynomial time a TJ-sequence between I and J.

A minimum k-path vertex cover I^r can be easily constructed in linear time by modifying `Partition` as follows: Initially, $I^r = \emptyset$. In each iteration of the **while** loop, add to I^r the vertex v of the properly rooted subtree T_v that is currently considering. Such a vertex v can be obtained from the process of checking if T contains a properly rooted subtree described in the proof of Lemma 7. Let $I^\star$ be any k-path vertex cover of size $|I| = |J|$ such that $I^r \subseteq I^\star$. We claim that both I and J can be reconfigured to $I^\star$ under TJ. As a result, a TJ-sequence between I and J can be constructed by reconfiguring I to $I^\star$, and then $I^\star$ to J.

We now show how to construct a TJ-sequence between I and $I^\star$. Let $P(T) = \{T_1(r), \ldots, T_{\psi_k(T)}(r)\}$ be a partition of T resulting from the algorithm `Partition` and let $I_0 = I$. Intuitively, we will first "settle" the tokens in $I^r \subseteq I^\star$ (**Step 1**), and then, as the tokens in I^r already cover all k-paths in T, the remaining tokens in $I^\star \setminus I^r$ can be easily "settled" by jumping tokens one-by-one in arbitrary order (**Step 2**).

- **Step 1:** For each i from 1 to $\psi_k(T)$, let $v_i \in I^r \cap V(T_i(r))$. If v_i does not contain a token in I_{i-1}, we jump a token from some vertex $x_i \in I_{i-1} \cap V(T_i(r))$ to v_i. Otherwise, we do nothing. Let I_i be the resulting set. Note that any k-path in T covered by x_i must also be covered by some v_j with $j \leq i$. A simple induction shows that $I_i = I_{i-1} \setminus \{x_i\} \cup \{v_i\}$ forms a k-path vertex cover of T.

- **Step 2:** For $x \in I_{\psi_k(T)} \setminus I^\star$ and $y \in I^\star \setminus I_{\psi_k(T)}$, we simply jump the token on x to y, and repeat the process with $I_{\psi_k(T)} \setminus \{x\}$ and $I^\star \setminus \{y\}$ instead of $I_{\psi_k(T)}$ and $I^\star$, respectively. Since $I^r \subseteq I_{\psi_k(T)} \cap I^\star$ is already a minimum k-path vertex cover, any TJ-step described above results a k-path vertex cover of T.

Since each token in I is jumped at most once, the above construction can be done in linear time. We have described how to construct a TJ-sequence from J to $I^\star$. In a similar manner, a TJ-sequence between J and $I^\star$ can be constructed. Our proof of Theorem 8 is complete. □

Consequently, combining Theorem 8 and Lemma 1, we have the following theorem.

Theorem 9. *For any instance $(T, I, J, \mathsf{TAR}(u))$ of k-PVCR on a tree T, one can decide if I and J are reconfigurable in polynomial time.*

Proof. Clearly, if $u < \max\{|I|, |J|\}$ or $u = \psi_k(T)$ then $(T, I, J, \mathsf{TAR}(u))$ is a no-instance, because either I or J cannot be modified by adding/removing tokens. We now consider the case $u \geq \max\{|I|, |J|\}$ and $u > \psi_k(T)$. Note that if $|I| < |J|$ then we can add tokens to I until the resulting k-path vertex cover is of size $|J|$, simply because $u \geq \max\{|I|, |J|\}$. As a result, we can assume without loss of generality that $|I| = |J| = s$ for some constant s. By Theorem 8 and Lemma 1, it follows that there always exists a $\mathsf{TAR}(s+1)$-sequence between I and J. If $s + 1 \leq u$ then clearly a $\mathsf{TAR}(s+1)$-sequence is also a $\mathsf{TAR}(u)$-sequence, and we are done. Assume that $s + 1 > u$. Since $u \geq s$ and $u > \psi_k(T)$, it follows that $u = s$ and both I and J are not minimum. Now, we need to check if we can remove at least one token from I (resp. J), which can be done in polynomial time by checking each token one by one and verifying whether its removal results a k-path vertex cover. If this is possible for both I and J, we remove exactly one token from I (resp. J) to obtain a new k-path vertex cover I' (resp.

J') of size $s-1$. By Lemma 1, there exists a $\mathsf{TAR}(u)$-sequence between I' and J', and combining this sequence with the previous removal steps gives us a $\mathsf{TAR}(u)$-sequence between I and J. Otherwise, we can conclude that the given instance is a no-instance, because the first step of reconfiguring (either from I to J or vice versa) is to remove some token (since $u = s$, adding a token is not possible). □

4.2 Paths and Cycles

Here, we describe polynomial-time algorithms for k-PVCR on paths and cycles. As paths and cycles are the only (planar) graphs of maximum degree two, by combining Theorem 6 and our results, we have a complexity dichotomy of k-PVCR on (planar) graphs. Additionally, on paths, we claim that one can construct a *shortest* reconfiguration sequence between any two given k-path vertex covers (if exists) under each reconfiguration rule TS, TJ, and TAR.

4.2.1 k-PVCR on Paths

By Theorems 8 and 9, clearly k-PVCR on paths can be solved in polynomial time under each of TJ and TAR. In this section, we slightly improve this result by showing that one can construct a *shortest* reconfiguration sequence between two k-path vertex covers on a path not only under each of TJ and TAR but also under TS.

Given an instance (P, I, J, TJ) of k-PVCR where $|I| = |J| = s$, one can construct a shortest TJ-sequence between I and J. Suppose that vertices in $I = \{v_{i_1}, \ldots, v_{i_s}\}$ and $J = \{v_{j_1}, \ldots, v_{j_s}\}$ are ordered such that $1 \leq i_1 < \cdots < i_s \leq n$ and $1 \leq j_1 < \cdots < j_s \leq n$. In each step of the algorithm, we move a token on the "rightmost" vertex $v_{i_p} \in I \setminus J$ to the "rightmost" vertex $v_{j_p} \in J \setminus I$ if $i_p > j_p$ or vice-versa otherwise, for $p \in \{1, \ldots, s\}$. As a reconfiguration sequence is reversible, one can easily form a TJ-sequence between I and J. Note that each step of the algorithm reduces $|I \Delta J|/2$ by exactly one. Finally, we obtain a shortest TJ-sequence between I and J of length exactly $|I \Delta J|/2$.

Theorem 10. *Given an instance (P, I, J, TJ) of k-PVCR on a path P, the k-path vertex covers I and J are reconfigurable if and only if $|I| = |J|$. Moreover, we can compute a shortest reconfiguration sequence in $O(n)$ time.*

Proof. Let $P = v_1 v_2 \ldots v_n$ be a given path. In the following, we use the expression *rightmost* instead of using "with the largest index". Algorithm 2 describes an algorithm `PVCRPathTJ`(P, I, J) for k-PVCR on paths under TJ.

Clearly, if I and J are reconfigurable under TJ then they are of the same size. It remains to show the if direction. To this end, we show that `PVCRPathTJ`(P, I, J) correctly constructs a TJ-sequence between two k-path vertex covers I, J of the same size. In each iteration of the **while** loop, when $v_i \in I$, we confirm that if we move a token from v_i to v_j, the resulting token-set still keeps k-path vertex cover property. In other words, the constructed sequence S_I is indeed a TJ-sequence. Suppose to the contrary that moving the token on v_i to the left (i.e., to the direction in which the indices get smaller) results in some non-covered k-path, say $Q = v_\ell v_{\ell+1} \ldots v_{\ell+k-1}$, where $\ell \leq i \leq \ell + k - 1$ and $j + 1 \leq \ell \leq n - k + 1$. Since J is a k-path vertex cover, there must be some vertex $v_{\ell'} \in J$ for $\ell \leq \ell' \leq \ell + k - 1$. Also, since $v_i \in I \setminus J$, $\ell' \neq i$. If $\ell' < i$,

Algorithm 2: PVCRPathTJ(P, I, J)

Input: A path P of n vertices, initial token-set I, and target token-set J;
Output: A reconfiguration sequence S;

Let S, S_I, S_J be reconfiguration sequences, and initialize them by $\emptyset$;
while $I \Delta J \neq \emptyset$ **do**
  $v_i \leftarrow$ the rightmost vertex in $P[I \Delta J]$;
  if $v_i \in I$ **then**
    Find the rightmost token v_j in $J \setminus I$ (here $j < i$);
    $S_I := S_I \oplus \langle I, I \setminus \{v_i\} \cup \{v_j\} \rangle$;
    $I := I \setminus \{v_i\} \cup \{v_j\}$;
  if $v_i \in J$ **then**
    Find the rightmost token v_j in $I \setminus J$ (here $j < i$);
    $S_J := S_J \oplus \langle J, J \setminus \{v_i\} \cup \{v_j\} \rangle$;
    $J := J \setminus \{v_i\} \cup \{v_j\}$;
$S := S_I \oplus \mathrm{rev}(S_J)$;
return S;

then $v_{\ell'} \in I$; otherwise, $v_j \in J \setminus I$ is not rightmost. If $\ell' > i$, then $v_{\ell'} \in I$; otherwise, v_i is not rightmost in $P[I \Delta J]$. Therefore $v_{\ell'} \in J \cap I$ always covers P, a contradiction. In a similar manner, one can also verify that S_J is indeed a TJ-sequence. Let I' be the k-path vertex cover obtained when the condition of the **while** loop is violated. Clearly, S_I (resp. S_J) reconfigures I (resp. J) to I'. Therefore, $S = S_I \oplus \mathrm{rev}(S_J)$ reconfigures I to J.

Next, we claim that S is shortest. Note that any TJ-sequence between I and J uses at least $|I \Delta J|/2$ TJ-steps. Moreover, in PVCRPathTJ(P, I, J), we move tokens exactly $|I \Delta J|/2$ times: in each iteration, exactly one token (either from $I \setminus J$ or $J \setminus I$) is moved, and then the size of $I \Delta J$ decreases by 2. Therefore, S is shortest. Consequently, the running time is $O(n)$. □

By Theorem 10 and Lemma 1, we obtain the following result on k-PVCR on a path P under TAR.

Theorem 11. *For any instance $(P, I, J, \mathsf{TAR}(u))$ of k-PVCR on a path P on n vertices, one can decide if I and J are reconfigurable in linear time.*

Proof. A similar approach as in the proof of Theorem 9 can be applied. Note that in the case $s = u$, where $s = |I| = |J|$, we have to check if we can remove at least one token from I (resp. J) is as follows. Given a path $P = v_1 v_2 \dots v_n$, let us assume that $I = \{v_{i_1}, v_{i_2}, \dots, v_{i_s}\}$ where $1 \leq i_1 < i_2 < \cdots < i_s \leq n$. In order to check if a token on u can be removed, assuming $u = v_{i_j}$ for some j such that $1 \leq j \leq s$, we do as follows. (1) If $j \in \{2, \dots, s-1\}$, then check if $\mathrm{dist}_G(v_{i_{j-1}}, v_{i_{j+1}}) \leq k$, and (2) if $j = 1$, then check if $\mathrm{dist}_G(v_1, v_{i_j}) \leq k - 1$, and (3) if $j = s$, then check if $\mathrm{dist}_G(v_{i_j}, v_n) \leq k - 1$. Indeed, this can be done in $O(n)$ time: for each token, one needs $O(1)$ time for checking if the resulting set obtained by removing u is still a k-path vertex cover. The correctness of

Function 3: $\texttt{Push}(P, I, i, j)$

Input: A path $P = v_1 \dots v_n$, a k-path vertex cover I, and two indices i and j with $1 \le i < j \le i + k \le n$;

Output: A sequence S of TS-steps that moves the token on v_i to v_j;

```
S = ∅;
while i ≠ j do
    if v_{i+1} ∈ I then
        S := S ⊕ Push(P, I, i + 1, i + 2);    // Both S and I are updated
    S := S ⊕ ⟨I, I \ {v_i} ∪ {v_{i+1}}⟩;
    I := I \ {v_i} ∪ {v_{i+1}};
    i := i + 1;
return S;
```

this checking easily follows from the definition of k-path vertex cover. One can see that similar things can be done for J. □

Now we sketch the idea for solving the problem under TS in polynomial time. Given an instance (P, I, J, TS) of k-PVCR where $|I| = |J| = s$, one can construct a shortest TS-sequence between I and J. Suppose that vertices in $I = \{v_{i_1}, \dots, v_{i_s}\}$ and $J = \{v_{j_1}, \dots, v_{j_s}\}$ are ordered such that $1 \le i_1 < \cdots < i_s \le n$ and $1 \le j_1 < \cdots < j_s \le n$. Our goal is to construct a shortest TS-sequence (of length $\sum_{p=1}^{s} \mathrm{dist}_P(v_{i_p}, v_{j_p})$) that repeatedly slides the token on the "leftmost" vertex $v_{i_p} \in I$ to the "leftmost" vertex $v_{j_p} \in J$ if $i_p < j_p$ or vice-versa otherwise, for $p \in \{1, \dots, s\}$.

The key point is, in certain conditions, one can construct in polynomial time a function $\texttt{Push}(P, I, i, j)$ (Function 3) whose task is to output a TS-sequence that moves the token placed at some vertex v_i of the k-path vertex cover I to vertex v_j in a given path $P = v_1 v_2 \dots v_n$, where $1 \le i < j \le i + k \le n$. Roughly speaking, $\texttt{Push}(P, I, i, j)$ slides the token t on v_i toward v_j along the path $P_{ij} = v_i v_{i+1} \dots v_j$ until either t ends up at v_j or there is some index $p \in \{i, \dots, j-1\}$ where t is already placed at v_p but cannot immediately move to v_{p+1} because there is already some token t' placed there. In the latter case, one can recursively call Push to slide t' from v_{p+1} to v_{p+2} and therefore enabling t (which is currently placed at v_p) to slide to v_{p+1}. Now, the same situation happens again with t and t', and the resolving procedure can be done in the same manner as before. This process stops when t is finally placed at v_j.

The following lemma says that if certain conditions are satisfied, the output of $\texttt{Push}(P, I, i, j)$ is indeed a TS-sequence that reconfigures the k-path vertex cover I to some other k-path vertex cover of P.

Lemma 12. *Let $P = v_1 v_2 \dots v_n$ be a path on n vertices, and let I be a k-path vertex cover of P. Let $i \in \{1, \dots, n\}$ be such that either $i \le k + 1$ or $\{v_{i-1}, \dots, v_{i-k}\} \cap I \neq \emptyset$. If $\{v_i, v_{i+1}, \dots, v_{i+p}\} \subseteq I$ and $v_{i+p+1} \notin I$ for some integer p satisfying $0 \le p \le n - i - 1$, then there exists a TS-sequence in P that reconfigures I to $I \setminus \{v_i, v_{i+1}, \dots, v_{i+p}\} \cup \{v_{i+1}, \dots, v_{i+p+1}\}$. Consequently, if the assumption is satisfied, the output of $\texttt{Push}(P, I, i, j)$ is indeed a TS-sequence in P that reconfigures I to some k-path vertex cover of P.*

Proof. We prove the lemma by induction on p. If $p = 0$, then by the assumption, the lemma clearly holds because the token on v_i can indeed be moved to v_{i+1} without leaving any non-covered k-path. Assume that if $\{v_i, v_{i+1}, \ldots, v_{i+p-1}\} \subseteq I$ and $v_{i+p} \notin I$ for some integer p satisfying $0 \leq p \leq n - i - 1$, then there exists a TS-sequence S' in P that reconfigures I to $I \setminus \{v_i, v_{i+1}, \ldots, v_{i+p-1}\} \cup \{v_{i+1}, \ldots, v_{i+p}\}$. We claim that if $\{v_i, v_{i+1}, \ldots, v_{i+p}\} \subseteq I$ and $v_{i+p+1} \notin I$ for some integer p satisfying $0 \leq p \leq n - i - 1$, then there exists a TS-sequence S in P that reconfigures I to $I \setminus \{v_i, v_{i+1}, \ldots, v_{i+p}\} \cup \{v_{i+1}, \ldots, v_{i+p+1}\}$. Note that the k-path $v_{i+p-k+1} \ldots v_{i+p}$ is (at least) covered by both v_{i+p-1} and v_{i+p}. Therefore, the token on v_{i+p} can be slid to v_{i+p+1} without leaving any non-covered k-path. More formally, $I' = I \setminus \{v_{i+p}\} \cup \{v_{i+p+1}\}$ is a k-path vertex cover in P. By the inductive hypothesis, there exists a TS-sequence S' that reconfigures I' to $I' \setminus \{v_i, v_{i+1}, \ldots, v_{i+p-1}\} \cup \{v_{i+1}, \ldots, v_{i+p}\} = I \setminus \{v_i, v_{i+1}, \ldots, v_{i+p}\} \cup \{v_{i+1}, \ldots, v_{i+p+1}\}$. Thus, $S = \langle I, I' \rangle \oplus S'$ is our desired TS-sequence. It is not hard to see that each iteration of the **while** loop in $\texttt{Push}(P, I, i, j)$ performs exactly the procedure we have just described (the case $p = 0$ corresponds to the steps outside the **if** condition, the case $p \geq 0$ corresponds to the recursive call inside the **if** condition). As a result, if the assumption of this lemma is satisfied, $\texttt{Push}(P, I, i, j)$ is indeed a TS-sequence. □

Clearly, the function $\texttt{Push}(P, I, i_p, j_p)$ can be used to slide a token on v_{i_p} to v_{j_p} for $p \in \{1, \ldots, s\}$ and $i_p < j_p$. Thus, we have the following theorem.

Theorem 13. *Given an instance (P, I, J, TS) of k-PVCR on a path P, the k-path vertex covers I and J are reconfigurable if and only if $|I| = |J|$. Moreover, we can compute a shortest reconfiguration sequence in $O(n^2)$ time.*

Proof. Before proving Theorem 13, we describe the algorithm $\texttt{PVCRPathTS}(P, I, J)$ (Algorithm 4) that takes two k-path vertex covers I and J of P with $|I| = |J|$ as the input, and returns a TS-sequence between them. In the following, we use the expression *leftmost* instead of using "with the smallest index".

Suppose that vertices in $I = \{v_{i_1}, \ldots, v_{i_s}\}$ and $J = \{v_{j_1}, \ldots, v_{j_s}\}$ are ordered such that $1 \leq i_1 < \cdots < i_s \leq n$ and $1 \leq j_1 < \cdots < j_s \leq n$, where $s = |I| = |J|$. Intuitively, $\texttt{PVCRPathTS}(P, I, J)$ outputs a TS-sequence that slides the token on v_{i_p} to v_{j_p} for $p \in \{1, \ldots, s\}$. Since P is a path, this is the only way of sliding tokens, and thus any TS-sequence between I and J uses at least $\sum_{p=1}^{s} \mathrm{dist}_P(v_{i_p}, v_{j_p})$ TS-steps.

Now we prove Theorem 13. As before, the only-if direction is trivial. We show that $\texttt{PVCRPathTS}(P, I, J)$ constructs a shortest TS-sequence between two k-path vertex covers I, J of P with $|I| = |J|$ in $O(n^2)$ time.

We first verify that the output of $\texttt{PVCRPathTS}(P, I, J)$ is a TS-sequence between I and J in P. Note that if in the current iteration of the **while** loop in PVCRPathTS, the token on v_i is moved to v_j (i.e., $i < j$), then the distance between v_j and the two untouched vertices considered in the next iteration must be at most k; otherwise, some non-covered k-path appears. Then, the assumption of Lemma 12 is satisfied in the next iteration. A similar argument holds for $i > j$. As a result, the function Push always returns a TS-sequence. Let I' be the k-path vertex cover of P obtained when the condition of the **while** loop of $\texttt{PVCRPathTS}(P, I, J)$ is violated. Then, it is not hard to see that S_I (resp. S_J) is a TS-sequence that reconfigures I (resp. J) to I', and therefore $S = S_I \oplus \mathrm{rev}(S_J)$ reconfigures I to J.

Note that in the function $\texttt{Push}(P, I, i, j)$ (and also $\texttt{Push}(P, J, j, i)$), Push is called at most once for each vertex of P, which implies $\texttt{Push}(P, I, i, j)$ runs in $O(n)$ time. Moreover, PVCRPathTS marks each vertex in I and J exactly twice. Thus, in total, PVCRPathTS runs in $O(n^2)$ time.

To conclude the proof of Theorem 13, we show that the TS-sequence S between I and J in P obtained from $\texttt{PVCRPathTS}(P, I, J)$ is shortest. To see this, note that for each $p \in \{1, \ldots, s\}$, either the token t on $v_{i_p} \in I$ is slid to $v_{j_p} \in J$ or the token t' on $v_{j_p} \in J$ is slid to $v_{i_p} \in I$ in some iteration of the **while** loop in $\texttt{PVCRPathTS}(P, I, J)$, and either S_I or S_J is then updated accordingly. Suppose that the algorithm slides t to v_{j_p}. Note that if there is any token t'' placed at some vertex v_{i_q} ($i_q \in \{i_p + 1, \ldots, j_p\}$ in the path $v_{i_p} v_{i_p+1} \ldots v_{j_p}$, then even when t'' is moved by some Push calls, by the time t ends up at v_j, t'' cannot be placed at any vertex whose index is larger than j_q. (We always have $i_p < i_q \leq j_p < j_q$ for all such i_q.) Clearly, if no such v_{i_q} exists, sliding t has no effect on sliding any other token in the next iterations. A similar argument holds in case the algorithm slides t'. Thus, we can conclude that $\texttt{PVCRPathTS}(P, I, J)$ performs exactly $\sum_{p=1}^{s} \mathrm{dist}_P(v_{i_p}, v_{j_p})$ TS-steps, and therefore outputs a shortest TS-sequence. □

4.2.2 k-PVCR on Cycles.

Let $C = v_0 v_1 \ldots v_{n-1} v_0$ be a given n-vertex cycle, and let (C, I, J, R) be a k-PVCR instance on C under a reconfiguration rule $\mathsf{R} \in \{\mathsf{TJ}, \mathsf{TS}, \mathsf{TAR}(u)\}$. We remark that if $|I| = |J| = \lceil n/k \rceil$ and $n = c \cdot k$ for some c, then (C, I, J, R) where $\mathsf{R} \in \{\mathsf{TS}, \mathsf{TJ}\}$ is a no-instance. This is because no tokens can be moved in such instances.

Here we assume that the indices of vertices on the cycle increase in the clockwise manner. We claim that it is possible to apply the algorithms for paths to cycles, by cutting a cycle into a path with a vertex in $I \cap J$, if it exists. Moreover, if $I \cap J = \emptyset$, we claim that one can always move tokens to reach an instance where $I \cap J \neq \emptyset$. Our algorithms do not always achieve the shortest reconfiguration sequence. However, we later show that achieving the shortest sequence even on cycles under TJ might not be trivially easy, since we can systematically create the instances such that the length of the shortest reconfiguration sequence is not equal to $|I \Delta J|/2$.

Now, we describe the sketch how to cut C under TJ, TS, and TAR. In the TS case, without loss of generality, we can assume that either $|I| \neq \lceil n/k \rceil$ or $n \neq c \cdot k$ holds. If v is already in $I \cap J$, we cut C by removing v. The following lemma ensures that if I and J are reconfigurable in $C - v$, then $I \cup \{v\}$ and $J \cup \{v\}$ are reconfigurable in C.

Lemma 14. *Let C be an n-vertex cycle and v be a token in $I \cap J$ of C. Then, for any k-path vertex cover I' of $C - v$, $I' \cup \{v\}$ is a k-path vertex cover of C.*

Proof. Let us assume v to be v_0. Consider the path $P = C - v = v_1 v_2 \ldots v_{n-2} v_{n-1}$ and a k-path vertex cover I' on P. Since I' covers all the k-paths on P, I' has at least one token on the k-path $P' = v_1 v_2 \ldots v_k$ and also at least one token on the k-path $P'' = v_{n-k} v_{n-k+1} \ldots v_{n-1}$. Now v is a token in $I \cap J$, if we connect two endpoints v_1 and v_{n-1} with v and create a cycle, all new k-paths include v and those paths are covered by v. This completes the proof. □

If $I \cap J = \emptyset$, there exists at least one token movable in the clockwise or counter-clockwise direction. Here, we say a token u is *movable* if and only if (i) there exists

a neighbor v of u such that no token is placed on v, and (ii) moving a token on u to v results a k-path vertex cover.

Lemma 15. *If either $|I| \neq \lceil n/k \rceil$ or $n \neq c \cdot k$ holds, then there exists at least one token movable by at least one step in the clockwise or counterclockwise direction. Furthermore, we can find such a token in linear time.*

Proof. If $|I| \neq \lceil n/k \rceil$, since $\lceil n/k \rceil$ is a minimum size of k-path vertex cover on n-vertex cycle, we can assume $|I| \geq \lceil n/k \rceil + 1$. This implies that there exists some k-path that has at least two tokens on it. We can find such a path (and thus such tokens) in linear time, since there are at most n distinct k-paths on an n-vertex cycle. Once we find such tokens, e.g., u and v, at least one of them can move at least one step in clockwise or counterclockwise direction, since the k-path is now covered by u and v and if we move v, either u or v still covers the k-path. Hence, if $|I| \neq \lceil n/k \rceil$, this lemma holds.

Consider the case $|I| = \lceil n/k \rceil$ and n is not divisible by k. Since I is a k-path vertex cover, I covers all k-paths in C. Clearly, C is a cycle of size n if and only if the number of edges of C is n. Suppose to the contrary that each k-path in C has exactly one token of I. Then, the length of the cycle is $|I| \cdot (k-1) + |I| = |I| \cdot k$, which contradicts to the assumption that n is not divisible by k. By this argument, similarly to the above $|I| \neq \lceil n/k \rceil$ case, there exists at least one k-path which has two tokens of I, and we can find them in linear time. This completes the proof. □

After finding such a movable token, we can use *rotate* operation repeatedly until we obtain at least one vertex in $I \cap J$. Here, the rotate operation takes a token-set, a movable token u which can be slid at least one step towards direction $d \in \{\text{clockwise}, \text{counterclockwise}\}$ as input, and outputs a TS-sequence that slides all tokens one step towards d. Intuitively, moving u one step towards d enables its successor (with respect to direction d) to move one step towards d, and so on. After obtaining at least one vertex in $I \cap J$, we can perform the cutting operation as before.

Next, we consider the TJ case. Since any TS-sequence is also a TJ-sequence, we can perform the same cutting operation as in the TS case. Then, using this cutting operation, we can show the following theorem.

Theorem 16. *Given an instance (C, I, J, R) of k-PVCR on a cycle C where $R \in \{TS, TJ\}$, if $|I| = |J| = \lceil n/k \rceil$ and $n = c \cdot k$ for some c, then (C, I, J, R) is a no-instance. Otherwise, the k-path vertex covers I and J are reconfigurable if and only if $|I| = |J|$. Moreover, we can compute a reconfiguration sequence for* TJ *rule in $O(n)$ time, and for* TS *rule in $O(n^2)$ time.*

Proof. We describe an algorithm (Algorithm 6) that takes $C = v_0 v_1 \dots v_{n-1} v_0$, initial token-set I, and target token-set J and outputs a reconfiguration sequence S if exists, and otherwise says no-instance. Lemma 14 shows that it is possible to cut the cycle C with a vertex $v \in I \cap J$; in other words, it is equivalent to consider problems on a path $P = C - v$.

Lemma 15 allows us to find at least one movable token if either $|I| \neq \lceil n/k \rceil$ or $n \neq c \cdot k$ holds. After finding such a movable token, we can use the *rotate* operation described in Function 5 and obtain at least one vertex $v \in I \cap J$. Let us assume that $I = \{v_{i_0}, v_{i_1}, \dots, v_{i_{s-1}}\}$ where $0 \leq i_0 < i_1 < \cdots < i_{s-1} \leq n-1$. Here, let v_{i_j} be a token

that is movable to at least one step in the direction $d \in \{\text{clockwise}, \text{counterclockwise}\}$, where $j \in \{0, \ldots, s-1\}$.

One can observe that, by Lemma 15, the reconfiguration sequence obtained by $\texttt{rot}(I, i_j, d)$ is a TS-sequence. This is indeed true, since it moves each token in I by exactly one step keeping the k-path vertex cover property, by starting to move tokens from v_{i_j} along the cycle until we meet v_{i_j} again.

By Lemma 14 and Lemma 15, $\texttt{PVCRCycleTS}(C, I, J)$ is shown to be correct. Note here that, for k-PVCR on cycles under TJ, one can use $\texttt{PVCRPathTJ}(C - v, I, J)$ instead of applying $\texttt{PVCRPathTS}(C - v, I, J)$ in the algorithm. For the computation time, since (i) **while** loop takes $O(kn)$ time and (ii) $\texttt{PVCRPathTS}(C - v, I, J)$ runs in $O(n^2)$ time, $\texttt{PVCRCycleTS}(C, I, J)$ runs in $O(n^2)$ time. For TJ case, since $\texttt{PVCRPathTJ}(C - v, I, J)$ runs in $O(n)$ time, $\texttt{PVCRCycleTJ}(C, I, J)$ runs in $O(n)$ time. □

For the TAR case, we can use the result for the TJ case and Lemma 1 to show the following theorem.

Theorem 17. *For any instance $(C, I, J, \mathsf{TAR}(u))$ of k-PVCR on a cycle C, one can decide if I and J are reconfigurable in linear time.*

Proof. Clearly, if $u < \max\{|I|, |J|\}$ or $u = \psi_k(C)$ then (C, I, J, TAR) is a no-instance, because either I or J cannot be modified by adding/removing tokens. We now consider the case $u \geq \max\{|I|, |J|\}$ and $u > \psi_k(C)$. Note that if $|I| < |J|$ then we can add tokens to I until the resulting k-path vertex cover is of size $|J|$, simply because $u \geq \max\{|I|, |J|\}$. As a result, we can assume without loss of generality that $|I| = |J| = s$ for some constant s. Now we have $|I| = |J| = s$ and $u > \psi_k(C)$, we divide into two cases: $u \geq s + 2$ or $u = s + 1$.

Consider the case $u \geq s + 2$. If $I \cap J = \emptyset$, then we add one token $v \notin I \cup J$. Then we can cut C by v and consider the instance $(C - v, I \setminus \{v\}, J \setminus \{v\})$ on the path $C - v$ under $\mathsf{TAR}(u')$ where $u' \geq s + 1$. Then, by Theorem 10 and Lemma 1, I is always reconfigurable to J under $\mathsf{TAR}(u)$. Otherwise, i.e., $I \cap J \neq \emptyset$, we can use the similar argument as before with $|I| = |J| = s - 1$ and $u \geq s + 1$, therefore I is always reconfigurable to J under $\mathsf{TAR}(u)$.

Next, consider the case $u = s + 1$. If $I \cap J \neq \emptyset$, also similar argument can be applied as before with $|I| = |J| = s - 1$ and $u = s$. Hence, in this case, I is always reconfigurable to J under $\mathsf{TAR}(u)$. Otherwise, we first find a token of I which is movable in direction $d \in \{\text{clockwise}, \text{counterclockwise}\}$. Recall that a token u is movable to some vertex v if the resulting set still keeps a k-path vertex cover property. If we can find such a token, we can rotate I in d until $I \cap J \neq \emptyset$ as in the algorithm $\texttt{PVCRCycleTS}$. We note that though such rotation forms a TS-sequence (which is also a TJ-sequence), by Lemma 1, it can be converted to a $\mathsf{TAR}(u)$-sequence. If we finish the rotation, then we can also cut C by $v \in I \cap J$ and similar argument can be applied as before. Else, assume without loss of generality that no token in I can move. Then, by Lemma 15, it follows that I is minimum and $n = c \cdot k$. Now we have $u = s + 1$, and we can add exactly one token. However, even when adding a new token v, one cannot remove any other token u while keeping the k-path vertex cover property. Suppose to the contrary, let $I' = I \setminus \{u\} \cup \{v\}$. This implies that I' can be obtained from I by jumping the token

on u to v. However, since $n = c \cdot k$ and I and I' are token sets of minimum size, then I cannot be reconfigured to I' under TJ, a contradiction. □

So far, we have shown that k-PVCR on cycles under each of TJ and TAR(u) can be solved in $O(n)$ time, and under TS can be solved in $O(n^2)$ time (Theorems 16 and 17). To conclude this section, we give an example showing that even in a yes-instance (C, I, J, TJ) of k-PVCR ($k \geq 3$) under TJ on a cycle C, one may need to use more than $|I \Delta J|/2$ TJ-steps even in a shortest TJ-sequence. Intuitively, the lower bound $|I \Delta J|/2$ seems to be easy to achieve under TJ, simply by jumping tokens one by one from $I \setminus J$ to $J \setminus I$. However, as we show in the following lemma, to keep the k-path vertex cover property, sometimes a token in I may need to jump to some vertex not in $J \setminus I$ beforehand. This implies the non-triviality of finding a *shortest* reconfiguration sequence even under TJ.

Lemma 18. *For k-PVCR ($k \geq 3$) yes-instances (C, I, J, TJ) on cycles where $C = v_0 v_1 \dots v_{3k-2} v_0$, $I = \{v_0, v_k, v_{2k}\}$ and $J = \{v_{3k-2}, v_{2k-2}, v_{k-1}\}$, the length of a shortest reconfiguration sequence from I to J is greater than $|I \Delta J|/2$.*

Proof. We illustrate such instances in Figure 6. In Figure 6, black tokens are in I, and white tokens are in J. Note that $\mathrm{dist}_C(v_{2k-2}, v_{2k}) = 2$ and $\mathrm{dist}_C(v_{3k-2}, v_0) = \mathrm{dist}_C(v_{k-1}, v_k) = 1$.

First, v_0 is the only vertex that covers the path $P = v_0 v_1 \dots v_{k-1}$, which means v_0 cannot move to some vertex outside P, such as v_{3k-2}. Therefore, v_0 has no choice but to move to v_{k-1}. However then, the path $v_{2k} \dots v_{3k-2} v_0 \dots v_{k-1}$ is of length $2k - 2 \geq k$. By these arguments, v_0 cannot directly move to v_{k-1}. Similarly, since v_{2k} is the only vertex that covers the path $P' = v_{k+1} \dots v_{2k}$, the possible way is only to move v_{2k} to v_{2k-2}, which also results in an non-covered path $v_{2k-1} \dots v_{3k-2}$ of length $k - 1$. It is clear that v_k cannot move directly to either v_{2k-2} or v_{k-1}. Therefore, every token in I cannot move directly to one of the tokens in J, which means it requires at least one step to put some token on some vertex $v \notin I \Delta J$. This also holds for the case moving tokens in J to I. Hence, the length of the reconfiguration sequence is greater than $|I \setminus J| = |J \setminus I| = |I \Delta J|/2$.

Finally, we confirm that the created instance is a yes-instance. First, for example, one can move v_{2k} to v_{2k-1}, since after such a move the k-vertex path $v_{2k} \dots v_0$ is covered by the token v_0 and another k-vertex path $v_{2k-1} \dots v_{3k-2}$ is covered by the token v_{2k-1}. Then, now the length of path $v_{2k-1} \dots v_k$ is k, hence v_k can be moved to v_{k-1} by the similar argument. Therefore, by the reconfiguration sequence $S = \langle I = \{v_0, v_k, v_{2k}\}, \{v_0, v_k, v_{2k-1}\}, \{v_0, v_{k-1}, v_{2k-1}\}, \{v_{3k-2}, v_{k-1}, v_{2k-1}\}, \{v_{3k-2}, v_{k-1}, v_{2k-2}\} = J \rangle$, one can reconfigure I to J. □

5 Concluding Remarks

In this paper, we have investigated the complexity of k-PVCR under each of TS, TJ, and TAR for several graph classes. In particular, several known hardness results for VCR (i.e., $k = 2$) can be generalized for k-PVCR when $k \geq 3$. Additionally, we proved a

complexity dichotomy for k-PVCR by showing that it remains PSPACE-complete even if the input (planar) graph is of maximum degree three (using a reduction from NCL) and can be solved in polynomial time when the input (planar) graph is of maximum degree two (i.e., it is either a path or a cycle). On the positive side, we designed polynomial-time algorithms for k-PVCR on trees under each of TJ and TAR. We also showed how to construct a *shortest* reconfiguration sequence on paths, and presented an example showing the nontriviality of finding shortest reconfiguration sequences on cycles even under TJ. The question of whether one can solve k-PVCR on trees under TS in polynomial time remains open. Another target graphs may be chordal graphs (under each of TJ and TAR), cographs, and graphs of treewidth at most 2. Even on graphs of treewidth at most 2 (and moreover, on outerplanar graphs), the complexity of VCR remains open.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful and valuable comments that help improving the preliminary versions of this paper. This work is partially supported by JSPS KAKENHI Grant Numbers JP20H05964 (D.A. Hoang), JP18H04091, JP20K11666 and JP20H05794 (A. Suzuki), Japan.

References

[1] Hrishikesh B. Acharya, Taehwan Choi, Rida A. Bazzi, and Mohamed G. Gouda. The k-observer problem in computer networks. *Netw. Sci.*, 1(1-4):15–22, 2012.

[2] Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. Complexity and approximation in reoptimization. In *Computability in Context: Computation and Logic in the Real World*, pages 101–129. World Scientific, 2011.

[3] Moritz Beck, Kam-Yiu Lam, Joseph Kee Yin Ng, Sabine Storandt, and Chun Jiang Zhu. Concatenated k-path covers. In *Proc. of ALENEX 2019*, pages 81–91, 2019.

[4] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. In Rolf Niedermeier and Christophe Paul, editors, *Proc. of STACS 2019*, volume 126 of *LIPIcs*, pages 13:1–13:7. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[5] Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Proc. of WG 2017*, volume 10520 of *LNCS*, pages 127–139. Springer, 2017.

[6] Paul S. Bonsma. Independent set reconfiguration in cographs and their generalizations. *J. of Graph Theory*, 83(2):164–195, 2016.

[7] Paul S. Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proc. of SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014.

[8] Boštjan Brešar, František Kardoš, Ján Katrenič, and Gabriel Semanišin. Minimum k-path vertex cover. *Discrete Appl. Math.*, 159(12):1189–1195, 2011.

[9] Boštjan Brešar, Rastislav Krivoš-Belluš, Gabriel Semanišin, and Petra Šparl. On the weighted k-path vertex cover problem. *Discrete Appl. Math.*, 177:14–18, 2014.

[10] Marcin Briański, Stefan Felsner, Jędrzej Hodor, and Piotr Micek. Reconfiguring independent sets on interval graphs. *arXiv preprint*, 2021. To appear in Proceedings of MFCS 2021.

[11] Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theor. Comput. Sci.*, 600:132–142, 2015.

[12] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.

[13] Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *Proc. of ISAAC 2015*, volume 9472 of *LNCS*, pages 237–247. Springer, 2015.

[14] Stefan Funke, André Nusser, and Sabine Storandt. On k-path covers and their applications. *Proc. VLDB Endow.*, 7(10):893–902, 2014.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[16] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.

[17] Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics*, volume 409 of *London Math. Soc. Lecture Note Ser.*, pages 127–160. Cambridge University Press, 2013.

[18] 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. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.

[19] Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. Parameterized complexity of independent set reconfiguration problems. *Discrete Applied Mathematics*, 838:45–57, 2020.

[20] Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012.

[21] Mehul Kumar, Amit Kumar, and C. Pandu Rangan. Reoptimization of path vertex cover problem. In *Proc. of COCOON 2019*, volume 11653 of *LNCS*, pages 363–374. Springer, 2019.

[22] Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms.*, 15(1):7:1–7:19, 2019.

[23] Eiji Miyano, Toshiki Saitoh, Ryuhei Uehara, Tsuyoshi Yagita, and Tom C. van der Zanden. Complexity of the maximum k-path vertex cover problem. In *Proc. of WALCOM 2018*, pages 240–251. Springer, 2018.

[24] C.M. Mynhardt and S. Nasserasr. Reconfiguration of colourings and dominating sets in graphs. In Fan Chung, Ron Graham, Frederick Hoffman, Ronald C. Mullin, Leslie Hogben, and Douglas B. West, editors, *50 years of Combinatorics, Graph Theory, and Computing*, pages 171–191. CRC Press, 1st edition, 2019.

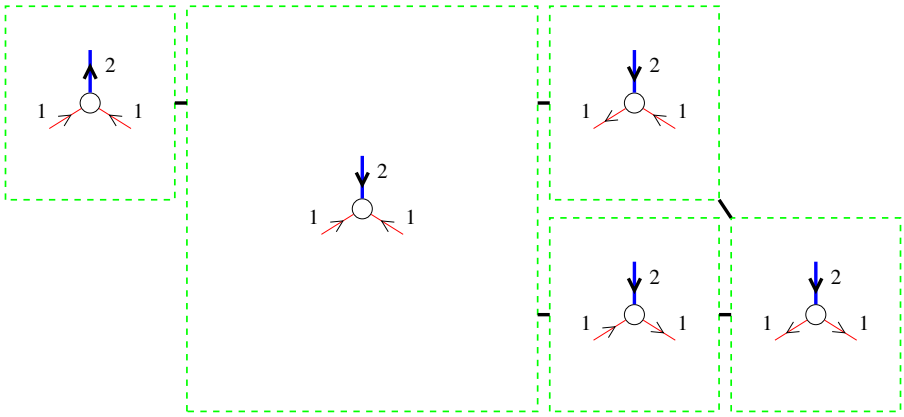
[25] Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4), 2018. (article 52).

[26] Yingli Ran, Zhao Zhang, Xiaohui Huang, Xiaosong Li, and Ding-Zhu Du. Approximation algorithms for minimum weight connected 3-path vertex cover. *Appl. Math. Comput.*, 347:723–733, 2019.

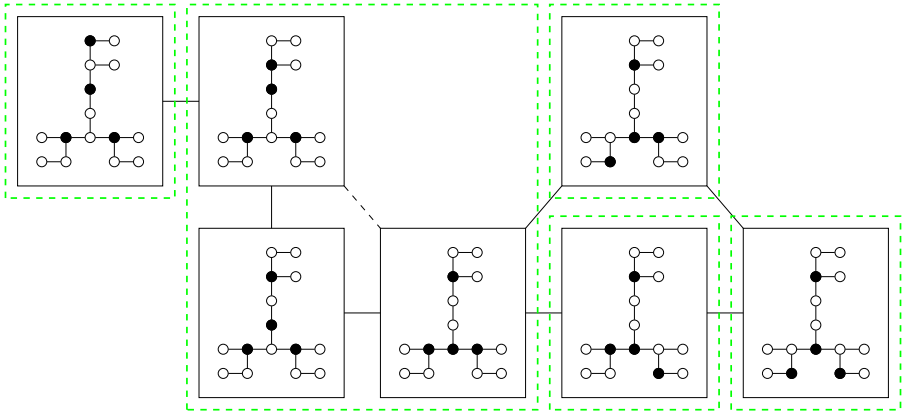
[27] Dekel Tsur. Parameterized algorithm for 3-path vertex cover. *Theor. Comput. Sci.*, 783:1–8, 2019.

[28] Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *J. Comput. Syst. Sci.*, 93:1–10, 2018.

[29] Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In *Proc. of IPEC 2015*, volume 43 of *LIPIcs*, pages 282–293, 2015.

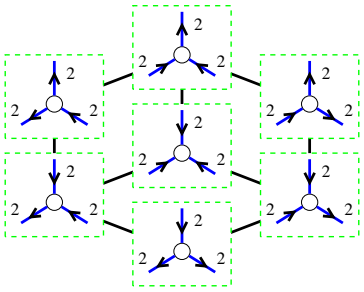


(a)

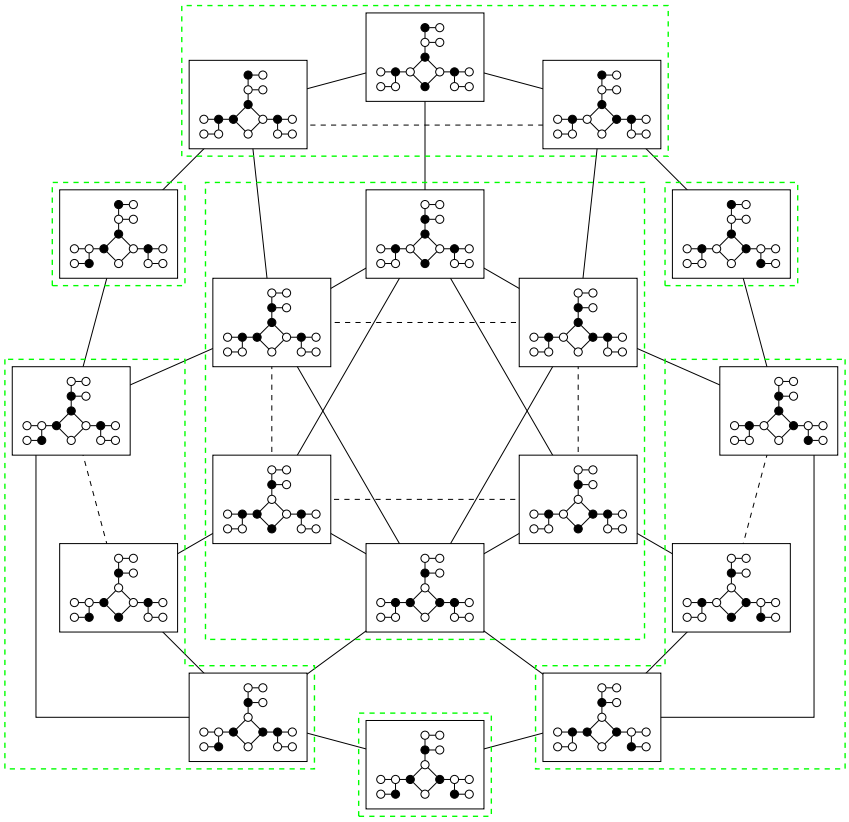


(b)

Figure 4: (a) All vaild orientations of the edges incident to an NCL AND vertex, and (b) all 3-PVCs of the AND gadget. The 3-PVCs connected by an edge are adjacent by TJ/TS rules, while the 3-PVCs connected by dashed edge are adjacent only by TJ rule.



(a)



(b)

Figure 5: (a) All vaild orientations of the edges incident to an NCL OR vertex, and (b) all 3-PVCs of the OR gadget. The 3-PVCs connected by an edge are adjacent by TJ/TS rules, while the 3-PVCs connected by dashed edge are adjacent only by TJ rule.

Algorithm 4: PVCRPathTS(P, I, J)

Input: A path $P = v_1 v_2 \dots v_n$, two k-path vertex covers I, J;
Output: A TS-sequence S between I and J in P;

Let S, S_I, S_J be reconfiguration sequences, and initialize them by $\emptyset$;
while $I \neq J$ **do**
  Mark all vertices in I and J as `untouched`;
  Find the leftmost `untouched` vertex $v_i \in I$ and the leftmost `untouched` vertex $v_j \in J$;
  if $i < j$ **then**
    $S_I := S_I \oplus \mathtt{Push}(P, I, i, j)$; `// I is updated in Push`
  else
    $S_J := S_J \oplus \mathtt{Push}(P, J, j, i)$; `// J is updated in Push`
  Mark v_i and v_j as `touched`;
$S := S_I \oplus \mathrm{rev}(S_J)$;
return S;

Function 5: rot(I, i_j, d)

Input: A token-set I, a token on $v_{i_j} \in I$, $d \in \{\text{clockwise}, \text{counterclockwise}\}$.
Output: A TS-sequence S that slides all tokens one step towards d, starting from v_{i_j}.

$S := \emptyset$;
$c := j$;
if d is clockwise **then**
  repeat
    $S := S \oplus \langle I, I \setminus \{v_{i_c}\} \cup \{v_{(i_c+1) \mod n}\}\rangle$;
    $I := I \setminus \{v_{i_c}\} \cup \{v_{(i_c+1) \mod n}\}$;
    $c := (c+1) \mod |I|$;
  until $c = j$;
else
  repeat
    $S := S \oplus \langle I, I \setminus \{v_{i_c}\} \cup \{v_{(i_c-1) \mod n}\}\rangle$;
    $I := I \setminus \{v_{i_c}\} \cup \{v_{(i_c-1) \mod n}\}$;
    $c := (c-1) \mod |I|$;
  until $c = j$;
return S;

Algorithm 6: PVCRCycleTS(C, I, J)

Input: A cycle $C = v_0 v_1 \ldots v_{n-1} v_0$, initial token-set I, and target token-set J;
Output: A reconfiguration sequence S if it exists; otherwise says no-instance;

$S := \emptyset$;
if $I \cap J = \emptyset$ and $|I| = \lceil n/k \rceil$ and n is divisible by k **then**
  return (C, I, J) is a no-instance;
Find a token $v_i \in I$ such that it can move at least one step in clockwise or counterclockwise direction, and let d be such a direction;
while $I \cap J = \emptyset$ **do**
  $S := S \oplus \texttt{rot}(I, i, d)$; `// I is updated in rot(I, i, d)`
  if d is clockwise **then**
    $i := (i + 1) \mod n$;
  else
    $i := (i - 1) \mod n$;
Pick one token $v \in I \cap J$;
$S' =$ PVCRPathTS$(C - v, I, J)$;
Update S' by adding v to each of its members;
$S := S \oplus S'$;
return S;

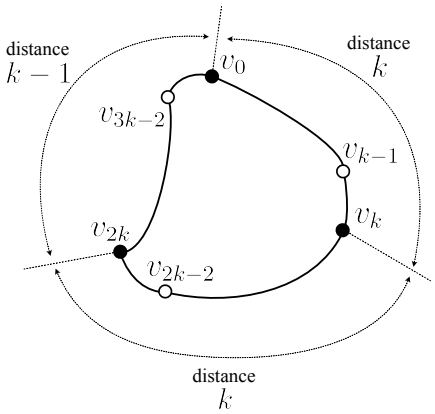


Figure 6: An instance (C, I, J, TJ) that requires more than $|I \Delta J|/2$ steps to reconfigure