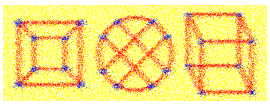
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 28, no. 3, pp. 87–101 (2024)
DOI: 10.7155/jgaa.v28i3.2973

Reconfiguration of vertex-disjoint shortest paths on graphs

Rin Saito[1] *Hiroshi Eto*[2] *Takehiro Ito*[1] *Ryuhei Uehara*[3]

[1]Graduate School of Information Sciences, Tohoku University, Sendai, Japan
[2]School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka, Japan
[3]School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan

Submitted: June 2023	Accepted: May 2024	Published: September 2024	
Article type: Regular paper		Communicated by: C.-C. Lin, B. M.-T. Lin, G. Liotta	

Abstract. We introduce and study reconfiguration problems for (internally) vertex-disjoint shortest paths: Given two tuples of internally vertex-disjoint shortest paths for fixed terminal pairs in an unweighted graph, we are asked to determine whether one tuple can be transformed into the other by exchanging a single vertex of one shortest path in the tuple at a time, so that all intermediate results remain tuples of internally vertex-disjoint shortest paths. We also study the shortest variant of the problem, that is, we wish to minimize the number of vertex-exchange steps required for such a transformation, if exists. These problems generalize the well-studied Shortest Path Reconfiguration problem. In this paper, we analyze the complexity of these problems from the viewpoint of graph classes, and give some interesting contrast.

1 Introduction

Combinatorial reconfiguration [7] has been extensively studied in the field of theoretical computer science. One of the most well-studied problems is the *reachability variant*: we are given two feasible solutions of a combinatorial search problem, and are asked to determine whether we can transform one into the other by repeatedly applying a prescribed reconfiguration step so that all intermediate results are also feasible. This kind of problems has been studied intensively for several combinatorial search problems. (See surveys [6, 9].)

For example, *the* Shortest Path Reconfiguration (SPR) *problem* is defined as follows [8]: we are given two shortest paths between two specified vertices s and t (called *terminals*) in an unweighted graph, and are asked to determine whether or not we can transform one into the

A preliminary version of this paper has appeared in the Proceedings of the 17th International Conference and Workshops on Algorithms and Computation (WALCOM 2023) [10].

E-mail addresses: rin.saito@dc.tohoku.ac.jp (Rin Saito) eto@ai.kyutech.ac.jp (Hiroshi Eto) takehiro@tohoku.ac.jp (Takehiro Ito) uehara@jaist.ac.jp (Ryuhei Uehara)

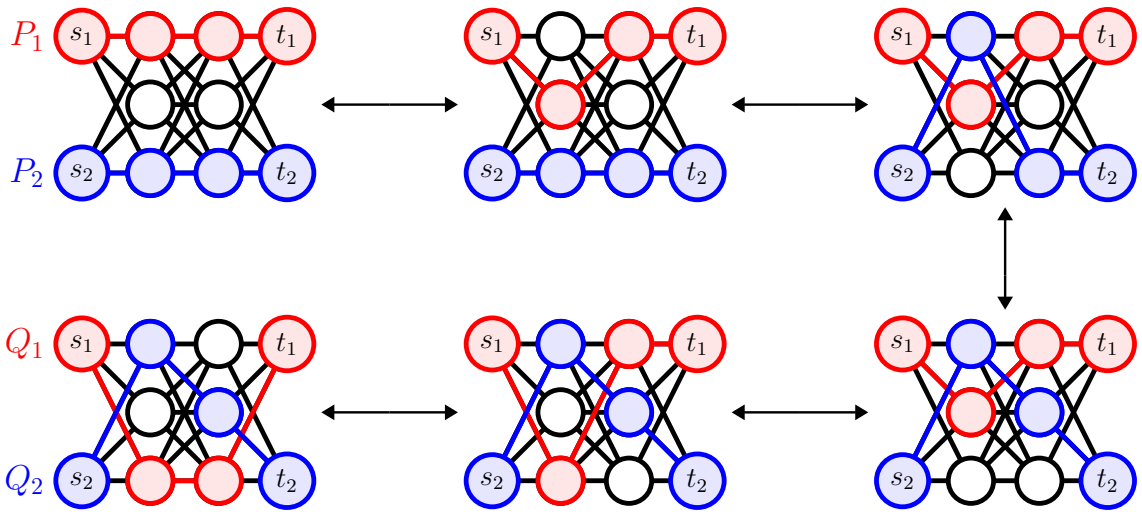

Figure 1: A sequence of tuples of (internally) vertex-disjoint shortest paths, for terminal pairs (s_1, t_1) and (s_2, t_2), between $\mathcal{P} = (P_1, P_2)$ and $\mathcal{Q} = (Q_1, Q_2)$.

other by exchanging a single vertex in a shortest path at a time, so that all intermediate results remain shortest paths between s and t. Surprisingly, the problem is PSPACE-complete [2, 11], and polynomial-time algorithms have been developed for restricted graph classes [1, 2, 3, 4]. (We will explain these known results in Section 1.1 later.)

1.1 Our problems and related results

In this paper, as generalizations of the SPR problem, we introduce and study reconfiguration problems for (internally) vertex-disjoint shortest paths in an unweighted graph G. For k terminal pairs (s_i, t_i), $i \in \{1, 2, \ldots, k\}$, consider a tuple of k paths in G such that the i-th path in the tuple joins s_i and t_i. Then, the k paths in the tuple are said to be *internally vertex-disjoint* if the internal vertices of k paths are all distinct and do not contain any terminal.

We now introduce two reconfiguration problems for vertex-disjoint shortest paths. Suppose that we are given two tuples $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ and $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ of internally vertex-disjoint paths such that each of P_i and Q_i is a shortest path in an unweighted graph G joining two terminals s_i and t_i for all $i \in \{1, 2, \ldots, k\}$. Then, *the* Reachability of Vertex-Disjoint Shortest Paths (RVDSP) *problem* asks to determine whether or not one can transform $\mathcal{P}$ into $\mathcal{Q}$ by exchanging a single vertex of one shortest path in the tuple at a time, so that all intermediate results remain tuples of internally vertex-disjoint shortest paths for k terminal pairs. (See Figure 1 as an example.) Thus, the RVDSP problem for $k = 1$ is equivalent to the SPR problem. In addition, we also study the *shortest variant* of RVDSP, *the* Shortest Reconfiguration of Vertex-Disjoint Shortest Paths (SRVDSP) *problem* which asks to determine whether or not there is a transformation between $\mathcal{P}$ and $\mathcal{Q}$ by at most ℓ vertex-exchange steps, for a given integer $\ell \geq 0$.

Kamiński et al. [8] introduced the SPR problem (i.e., the RVDSP problem for $k = 1$), and posed an open question of the complexity of the SPR problem. Bonsma [2] answered by proving that the SPR problem is PSPACE-complete for bipartite graphs. Since P $\subseteq$ NP $\subseteq$ PSPACE,

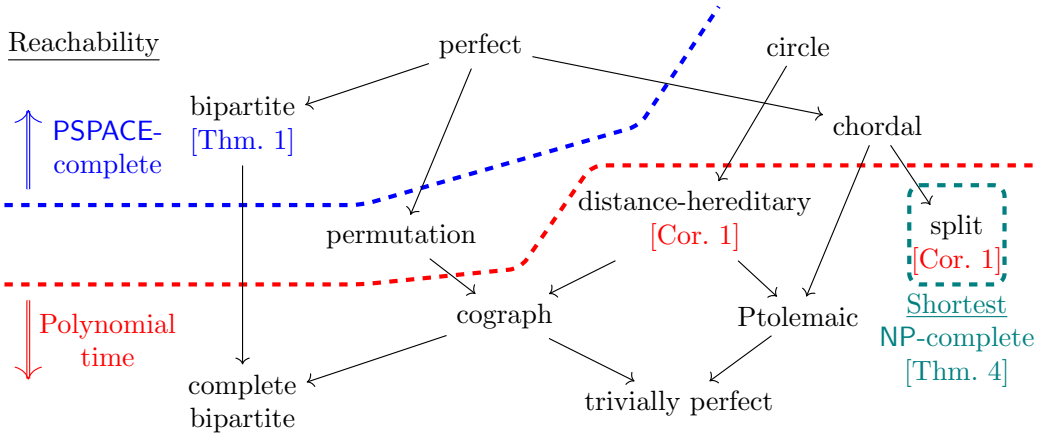

Figure 2: Our results with respect to graph classes. Each arrow represents the inclusion relationship between graph classes: $A \to B$ means that the graph class B is a proper subclass of the graph class A. In addition, we prove that the RVDSP problem is PSPACE-complete for bipartite graphs, and for bounded bandwidth graphs (Theorem 1), and the SRVDSP problem is solvable in polynomial time for distance-hereditary graphs and for split graphs if all k terminal pairs are the same (Theorem 3).

this means that the problem admits no polynomial-time algorithm under the assumption of P $\neq$ PSPACE, and furthermore implies that there is a yes-instance that requires super-polynomial steps for transforming one shortest path to the other under the assumption of NP $\neq$ PSPACE. These are somewhat surprising because the problem of finding shortest paths (especially, in an unweighted graph) is easy. Bonsma [2] posed another open question whether the SPR problem can be solved in polynomial time for graphs with bounded treewidth. This question was answered negatively by Wrochna [11]: the SPR problem remains PSPACE-complete even for graphs with bounded bandwidth. Note that the bandwidth of a graph gives an upper bound on the pathwidth (and hence the treewidth) of the graph; and hence the PSPACE-completeness holds also for graphs with bounded treewidth.

On the positive side, the SPR problem has been shown to be solvable in polynomial time when restricted to graph classes, such as chordal graphs and claw-free graphs [2]; planar graphs [3]; grid graphs [1]; circle graphs, permutation graphs, the Boolean hypercube, bridged graphs, and circular-arc graphs [4]. Furthermore, the shortest variant of the SPR problem (i.e., the SRVDSP problem for $k = 1$) is solvable in polynomial time for chordal graphs [2].

1.2 Our contribution

In this paper, we study the computational complexity of the RVDSP and SRVDSP problems from the viewpoint of graph classes. Figure 2 summarizes our results. (Throughout the paper, k denotes the number of terminal pairs.)

We first observe that the RVDSP problem for every fixed $k \geq 1$ is PSPACE-complete for bipartite graphs and bounded bandwidth graphs. On the positive side, we give a polynomial-time algorithm to solve the RVDSP problem for distance-hereditary graphs and for split graphs. Interestingly, our algorithm for these two graph classes can be obtained as a corollary of a single theorem (Theorem 2) by introducing the concept of "st-completeness" of graphs for terminal pairs

(s, t). Our algorithm is constructive and finds an actual transformation (if exists) that requires polynomial number of vertex-exchange steps.

We then prove that the SRVDSP problem is NP-complete for split graphs. On the positive side, we show that the problem is solvable in polynomial time for distance-hereditary graphs and for split graphs if all k terminal pairs are the same, that is, $(s_1, t_1) = (s_2, t_2) = \cdots = (s_k, t_k)$.

Our results give the following interesting contrast:

1. the RVDSP problem is PSPACE-complete for bipartite graphs (Theorem 1), while it is solvable in polynomial time for complete bipartite graphs (Corollary 1);
2. for split graphs, the RVDSP problem is solvable in polynomial time (Corollary 1), whereas the SRVDSP problem is NP-complete (Theorem 4); and
3. the SRVDSP problem for $k = 1$ is solvable in polynomial time for chordal graphs [2] (thus, for split graphs), while the SRVDSP problem for general k is NP-complete for split graphs (Theorem 4).

The rest of this paper is organized as follows. In Section 2, we provide some notation and terminology used in this paper. In Section 3, we present a polynomial-time algorithm for distance-hereditary graphs and for split graphs. In Section 4, we deal with the SRVDSP problem. Finally, in Section 5, we conclude the paper with some open questions.

2 Preliminaries

In this paper, we assume that graphs are simple and unweighted. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex and edge sets of G, respectively. Let $n = |V(G)|$ and $m = |E(G)|$ throughout the paper. For $u, v \in V(G)$, a path in G joining u and v is called a *uv-path*. We denote by $d_G(u, v)$ the minimum number of edges in any uv-path in G; we sometimes omit the subscript G if it is clear from the context. The *diameter of G* is the maximum $d_G(u, v)$ among any two vertices u, v in G. For two sets A and B, we denote by $A \triangle B$ the *symmetric difference* of A and B, that is, $(A \setminus B) \cup (B \setminus A)$.

Let k be a positive integer, and let (s_i, t_i) be a pair of vertices in G, called *terminals*, for $i \in \{1, 2, \ldots, k\}$. Then, k paths $P_1, P_2, \ldots, P_k$ in G are said to be *internally vertex-disjoint* if P_i is an $s_i t_i$-path in G for each $i \in \{1, 2, \ldots, k\}$, and the internal vertices of k paths are all distinct and do not contain any terminal. Note that internally vertex-disjoint paths may share terminals. In the following, we call internally vertex-disjoint paths simply *vertex-disjoint* paths. For a tuple $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ of vertex-disjoint paths, let $V(\mathcal{P}) = \bigcup_{i=1}^{k} V(P_i)$.

In this paper, we consider only *shortest* $s_i t_i$-paths in G, $i \in \{1, 2, \ldots, k\}$. For two tuples $\mathcal{P} = (P_1, P_2, \ldots, P_k)$ and $\mathcal{P}' = (P'_1, P'_2, \ldots, P'_k)$ of vertex-disjoint shortest paths, we write $\mathcal{P} \leftrightarrow \mathcal{P}'$ if $\sum_{i=1}^{k} |V(P_i) \triangle V(P'_i)| = 2$; in other words, $\mathcal{P}'$ can be obtained from $\mathcal{P}$ by exchanging a single (internal) vertex in some shortest path P_i with a vertex that is not contained in $V(\mathcal{P})$. A sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\ell \rangle$ of tuples of vertex-disjoint shortest paths is called a *reconfiguration sequence* between $\mathcal{P}_0$ and $\mathcal{P}_\ell$ if $\mathcal{P}_{r-1} \leftrightarrow \mathcal{P}_r$ for all $r \in \{1, 2, \ldots, \ell\}$. The *length* of a reconfiguration sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\ell \rangle$ is defined to be ℓ. We now define two following problems.

Reachability of Vertex-Disjoint Shortest Paths (RVDSP)
Input: An unweighted graph G, and two tuples $\mathcal{P}$ and $\mathcal{Q}$ of vertex-disjoint shortest paths for k terminal pairs (s_i, t_i).
Task: Determine if there is a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$.

Shortest Reconfiguration of Vertex-Disjoint Shortest Paths (SRVDSP)	
Input:	An unweighted graph G, two tuples $\mathcal{P}$ and $\mathcal{Q}$ of vertex-disjoint shortest paths for k terminal pairs (s_i, t_i), and an integer $\ell \geq 0$.
Task:	Determine if there is a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ of length at most ℓ.

Note that both RVDSP and SRVDSP problems are decision problems, and do not ask for an actual reconfiguration sequence as an output. We sometimes denote simply by $(G, \mathcal{P}, \mathcal{Q})$ an instance of the RVDSP problem, and by $(G, \mathcal{P}, \mathcal{Q}, \ell)$ an instance of the SRVDSP problem.

Definitions of layers and st-completeness

For two distinct vertices $s, t \in V(G)$ and $j \in \{0, 1, \ldots, d(s,t)\}$, we define

$$L_j = \{v \in V(G) \mid d(s,v) = j, d(s,v) + d(v,t) = d(s,t)\}. \tag{1}$$

We call L_j the *j-th st-layer*, that is, L_j is the set of vertices v such that $d(s,v) = j$ and v is contained in some shortest st-path. Note that $L_0 = \{s\}$ and $L_{d(s,t)} = \{t\}$. We denote by G_{st} the subgraph of G induced by all st-layers L_j, $j \in \{0, 1, \ldots, d(s,t)\}$. Then, any shortest st-path in G is contained in G_{st}. We say that G is *st-complete* if every vertex in L_j is adjacent in G to all vertices in L_{j+1} for all $j \in \{0, 1, \ldots, d(s,t) - 1\}$. We have the following lemma.

Lemma 1 *For two vertices $s, t \in V(G)$, one can construct G_{st} and check whether G is st-complete in $O(m + n)$ time.*

Proof: We first show that G_{st} can be constructed in $O(m + n)$ time. Since G is an unweighted graph, Eq. (1) says that the vertices in the layers L_j, $j \in \{0, 1, \ldots, d(s,t)\}$, can be determined by computing $d(s,v)$ and $d(t,v)$ for every vertex $v \in V(G)$. This can be done by the breadth-first search starting from s and t, and hence we can construct G_{st} in $O(m + n)$ time.

We then present a linear-time algorithm to check if G is st-complete. When we construct the graph G_{st}, we attach the label j to all vertices in L_j, $j \in \{0, 1, \ldots, d(s,t)\}$. Then, by the breadth-first search starting from s, we can count the number of edges joining the currently visiting vertex $u \in L_j$ and the vertices $v \in L_{j+1}$, $j \in \{1, 2, \ldots, d(s,t) - 2\}$. If the number of such edges is equal to $|L_{j+1}|$ for every vertex $u \in L_j$, then G is st-complete; otherwise not. □

The st-completeness of a graph G is a useful property, because we can forget the structure of G in the following sense: if we choose exactly one vertex from each st-layer L_j, $j \in \{0, 1, \ldots, d(s,t)\}$, the set of the chosen vertices *always* forms a shortest st-path in G.

3 Reachability variant

Recall that the RVDSP problem for $k = 1$ (i.e., the SPR problem) is PSPACE-complete for bipartite graphs [2], and for graphs with bounded bandwidth [11]. By introducing dummy vertex-disjoint shortest paths, one can observe the following hardness results.

Theorem 1 *For every fixed $k \geq 1$, the* RVDSP *problem is* PSPACE*-complete for bipartite graphs, and for graphs of bounded bandwidth.*

The main result of this section is the following theorem, whose proof will be given in Subsections 3.1 and 3.2.

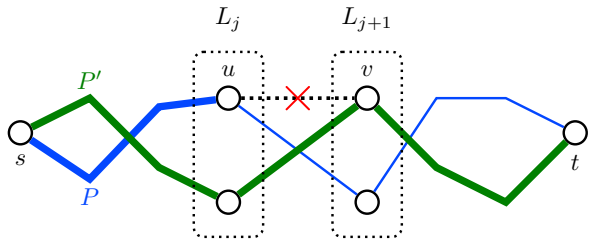

Figure 3: Illustration for the proof of Corollary 1 for distance-hereditary graphs.

Theorem 2 *Let G be a graph of diameter d as an input of the* RVDSP *problem such that G is st-complete for all k terminal pairs. Then, the* RVDSP *problem is solvable in $O(mk+ndk^2)$ time. Furthermore, in the same running time, one can find a reconfiguration sequence of length $O(d^2k^2)$ if exists.*

From Theorem 2, one can show that the RVDSP problem is solvable in polynomial time for some graph classes, as in the following corollary. A graph G is *split* if $V(G)$ can be partitioned into a clique and an independent set. A graph G is *distance hereditary* if $d_G(u,v) = d_{G'}(u,v)$ for every connected induced subgraph G' of G and all $u, v \in V(G')$.

Corollary 1 *The* RVDSP *problem is solvable in polynomial time for split graphs, and for distance-hereditary graphs.*

Proof: We first prove the claim for split graphs. Consider a split graph G such that $V(G)$ can be partitioned into a clique C and an independent set I. Then, according to the placement of a terminal pair (s,t), there are three cases to consider: $s,t \in C$; $s,t \in I$; and one is in C and the other is in I. Then, one can observe that G is st-complete for all the three cases.

We then prove the claim for distance-hereditary graphs. Suppose for a contradiction that a distance-hereditary graph G is not st-complete for some terminal pair (s,t). Then, there exist two vertices $u, v \in V(G)$ such that $uv \notin E(G)$, $u \in L_j$ and $v \in L_{j+1}$ for some $j \in \{1, 2, \ldots, d_G(s,t)-2\}$ (see Figure 3). By the definition of layers, there exists a shortest st-path P passing through u, and also exists a shortest st-path P' passing through v. Since $uv \notin E(G)$, we know that $P \neq P'$ although they may share vertices and edges. We consider the subgraph G' induced by $V(P')$ and $V(P) \cap \big(L_0 \cup L_1 \cup \cdots \cup L_j\big)$. (In Figure 3, G' is indicated by thick lines.) Then, G' is connected. Since $uv \notin E(G)$ and $d_{G'}(v,t) = d_G(v,t)$, we have

$$d_{G'}(u,t) = d_{G'}(u,v) + d_{G'}(v,t) > 1 + d_{G'}(v,t) = 1 + d_G(v,t) = d_G(u,t).$$

This contradicts the assumption that G is a distance-hereditary graph. □

Note that any split graph is of diameter at most 3, and hence we can drop the factor d in Theorem 2 for split graphs. On the other hand, the diameter of distance-hereditary graphs can be $\Omega(n)$. We also note that a complete bipartite graph is distance hereditary, and hence the RVDSP problem can be solved in polynomial time for complete bipartite graphs. Therefore, Theorem 1 and Corollary 1 give an interesting contrast of the complexity of the RVDSP problem. (See Figure 2 again.)

3.1 Characterization of reachability

Let $(G, \mathcal{P}, \mathcal{Q})$ be an instance of the RVDSP problem such that G is s_it_i-complete for each $i \in \{1, 2, \ldots, k\}$. We denote by L^i_j the j-th s_it_i-layer for each pair of integers $i \in \{1, 2, \ldots, k\}$ and $j \in \{0, 1, \ldots, d_G(s_i, t_i)\}$. For G and $\mathcal{P}$, we define a directed graph $G_{\mathcal{P}}$, called an *auxiliary graph for* $\mathcal{P}$, as follows: the vertex set of $G_{\mathcal{P}}$ is $V(G)$, and for each i, j, we add arcs (u, v) from $u \in V(P_i) \cap L^i_j$ to all vertices $v \in L^i_j \setminus \{u\}$; more specifically, the arc set of $G_{\mathcal{P}}$ is $\bigcup_{i,j}\{(u, v) \mid u \in V(P_i) \cap L^i_j,\ v \in L^i_j \setminus \{u\}\}$. Then, only the vertices in $\mathcal{P}$ have out-going arcs in $G_{\mathcal{P}}$.

For each pair of $i \in \{1, 2, \ldots, k\}$ and $j \in \{0, 1, \ldots, d_G(s_i, t_i)\}$, we place a (labeled) *token* t^i_j on the vertex $u \in V(P_i) \cap L^i_j$ in the auxiliary graph $G_{\mathcal{P}}$. Note that no two tokens are placed on the same vertex, because paths in $\mathcal{P}$ are vertex-disjoint. Conversely, the s_it_i-completeness of G ensures that any placement of the token t^i_j to a vertex in L^i_j yields a shortest s_it_i-path in G. The vertex on which t^i_j is placed is sometimes referred simply as t^i_j. We say that the token t^i_j is $\mathcal{P}$-*movable* if there exists a directed path in $G_{\mathcal{P}}$ from t^i_j to a vertex $v \notin V(\mathcal{P})$. We sometimes call such a $\mathsf{t}^i_j v$-path a t^i_j-*escape path under* $\mathcal{P}$. Recall that only the vertices in $\mathcal{P}$ have out-going arcs in $G_{\mathcal{P}}$, and hence only the last vertex v in the t^i_j-escape path under $\mathcal{P}$ is not contained in $V(\mathcal{P})$; in other words, tokens are placed on all the intermediate vertices in the t^i_j-escape path under $\mathcal{P}$. The $\mathcal{P}$-movable tokens have a good property, as follows.

Lemma 2 *Let $\mathcal{P}$ and $\mathcal{P}'$ be two tuples of vertex-disjoint shortest paths in G such that $\mathcal{P} \leftrightarrow \mathcal{P}'$. Then, a token t^i_j is $\mathcal{P}$-movable if and only if t^i_j is $\mathcal{P}'$-movable.*

Proof: We only prove the only-if direction; the other direction is symmetric. Suppose that $\mathcal{P}'$ is obtained from $\mathcal{P}$ by moving some token on a vertex y_1 to another vertex y_2. Consider any token t^i_j which is $\mathcal{P}$-movable. Then, there exists a t^i_j-escape path under $\mathcal{P}$; let $P = x_1x_2 \cdots x_l$. Since only the vertices in $\mathcal{P}$ have out-going arcs in $G_{\mathcal{P}}$, we know that $x_1, x_2, \ldots, x_{l-1} \in V(\mathcal{P})$. In addition, by the definition of t^i_j-escape paths, we know that $x_l \notin V(\mathcal{P})$. If neither y_1 nor y_2 appears in P, then P is a t^i_j-escape path also under $\mathcal{P}'$.

Consider the case where y_1 appears in P. If $x_1 = y_1$ and hence the token t^i_j was moved from $y_1 = x_1$ to y_2, then the auxiliary graph $G_{\mathcal{P}'}$ has an arc (y_2, y_1); this arc forms a t^i_j-escape path under $\mathcal{P}'$, and hence t^i_j is $\mathcal{P}'$-movable. We thus consider the case where y_1 appears in P and $x_1 \neq y_1$; let $x_r = y_1$, $r > 1$. Then, $x_r \notin V(\mathcal{P}')$, since we moved the token on $y_1 = x_r$ to y_2. Therefore, $x_1x_2 \cdots x_r$ forms a t^i_j-escape path under $\mathcal{P}'$, and hence t^i_j is $\mathcal{P}'$-movable.

We consider the remaining case: y_2 appears in P but y_1 does not appear in P. In this case, since $x_1, x_2, \ldots, x_{l-1} \in V(\mathcal{P})$ and $x_l \notin V(\mathcal{P})$, we know that $y_2 = x_l$. Then, a token is placed on $y_2 = x_l$ (which was moved from y_1) under $\mathcal{P}'$, and hence the auxiliary graph $G_{\mathcal{P}'}$ has an arc (x_l, y_1). Therefore, $x_1x_2 \cdots x_ly_1$ forms a t^i_j-escape path under $\mathcal{P}'$, and hence t^i_j is $\mathcal{P}'$-movable. □

For each pair of integers $i \in \{1, 2, \ldots, k\}$ and $j \in \{0, 1, \ldots, d_G(s_i, t_i)\}$, the vertex in $V(Q_i) \cap L^i_j$ is called the *target position* for the token t^i_j. For a tuple $\mathcal{P}'$ of vertex-disjoint shortest paths, we denote by $T_{\mathcal{P}'}$ the set of all tokens that are *not* placed on their target positions in $\mathcal{P}'$. The following lemma is the key for the proof of Theorem 2.

Lemma 3 *$(G, \mathcal{P}, \mathcal{Q})$ is a yes-instance if and only if every token in $T_{\mathcal{P}}$ is $\mathcal{P}$-movable.*

Proof: We first prove the only-if direction. Suppose that $(G, \mathcal{P}, \mathcal{Q})$ is a yes-instance, and hence there is a reconfiguration sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_\ell \rangle$ between $\mathcal{P} = \mathcal{P}_0$ and $\mathcal{Q} = \mathcal{P}_\ell$. Because every token t^i_j in $T_{\mathcal{P}}$ is not placed on its target position in $\mathcal{P}$, the token must be moved at least once

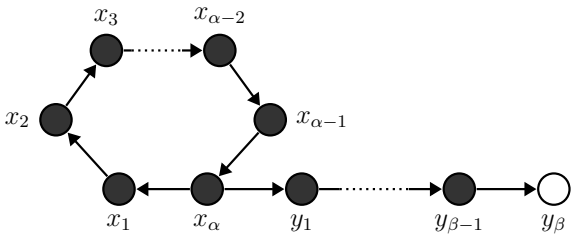

Figure 4: Illustration for Case (b).

in the reconfiguration sequence. Assume that $\mathtt{t}_j^i$ was moved between $\mathcal{P}_r$ and $\mathcal{P}_{r+1}$, from a vertex $y_1 \in L_j^i \cap V(\mathcal{P}_r)$ to another vertex $y_2 \in L_j^i \setminus V(\mathcal{P}_r) = L_j^i \setminus \{y_1\}$. Then, the auxiliary graph $G_{\mathcal{P}_r}$ has an arc (y_1, y_2); this arc forms a $\mathtt{t}_j^i$-escape path under $\mathcal{P}_r$, and hence $\mathtt{t}_j^i$ is $\mathcal{P}_r$-movable. Since $\mathcal{P}_r \leftrightarrow \mathcal{P}_{r-1} \leftrightarrow \cdots \leftrightarrow \mathcal{P}_0 = \mathcal{P}$, Lemma 2 implies that $\mathtt{t}_j^i$ is $\mathcal{P}$-movable.

We then prove the if direction, by induction on $|T_\mathcal{P}|$. If $|T_\mathcal{P}| = 0$, then $\mathcal{P} = \mathcal{Q}$ and hence $(G, \mathcal{P}, \mathcal{Q})$ is a yes-instance. Thus, suppose that $|T_\mathcal{P}| \geq 1$ and every token in $T_\mathcal{P}$ is $\mathcal{P}$-movable. We consider two following cases.

Case (a): We first consider the case where there exists a token $\mathtt{t}_j^i$ placed on the vertex $y_1 \in V(P_i) \cap L_j^i$ such that its target position y_2 is not occupied by any token in $\mathcal{P}$, that is, $V(Q_i) \cap L_j^i = \{y_2\}$ and $y_2 \notin V(\mathcal{P})$. In this case, we can move $\mathtt{t}_j^i$ to y_2 directly, and obtain the tuple $\mathcal{P}'$ of vertex-disjoint shortest paths; notice that there is an arc (y_1, y_2) in $G_\mathcal{P}$ by the definition of $G_\mathcal{P}$. Since $\mathcal{P} \leftrightarrow \mathcal{P}'$, Lemma 2 says that every $\mathcal{P}$-movable token is $\mathcal{P}'$-movable. Since $\mathtt{t}_j^i$ reaches its target position y_2, we have $|T_{\mathcal{P}'}| = |T_\mathcal{P}| - 1$. Therefore, we can apply the induction hypothesis to $(G, \mathcal{P}', \mathcal{Q})$.

Case (b): We then consider the other case, that is, the target positions of all tokens are occupied by tokens in $\mathcal{P}$. In this case, we can find a directed cycle $C = x_1 x_2 \dots x_\alpha$ in $G_\mathcal{P}$ such that x_{r+1} is the target position of the token placed on x_r for all $r \in \{1, 2, \dots, \alpha\}$; for convenience, we regard $x_{\alpha+1} = x_1$. Since $|T_\mathcal{P}| \geq 1$, we can assume that $\alpha \geq 2$ (i.e., C is not a self-loop). All tokens placed on $x_1, x_2, \dots, x_\alpha$ belong to $T_\mathcal{P}$, and hence all of them are $\mathcal{P}$-movable. Then, there exists at least one token $\mathtt{t}$ such that $\mathtt{t}$ is placed on a vertex in C, say x_α, and $G_\mathcal{P}$ has a $\mathtt{t}$-escape path $x_\alpha y_1 y_2 \dots y_\beta$ with $y_1, y_2, \dots, y_\beta \notin V(C)$. (See Figure 4.) Then, we move tokens as follows:

Step 1. move the token on y_r to y_{r+1} for each r, $\beta - 1 \geq r \geq 1$;

Step 2. move the token on x_α to y_1 (now no token is placed on x_α);

Step 3. move the token on x_r to x_{r+1} for each r, $\alpha - 1 \geq r \geq 1$;

Step 4. move the token on y_1 (which was placed on x_α in $\mathcal{P}$) to x_1; and

Step 5. move the token on y_r to y_{r-1} for each r, $2 \leq r \leq \beta$.

Note that, in Step 4, we can move the token on y_1 directly to x_1, because x_1 is the target position of the token which was placed on x_α in $\mathcal{P}$. After Step 5, each token on $y_1, y_2, \dots, y_\beta$ are placed on the same vertex as in $\mathcal{P}$, and each token on $x_1, x_2, \dots, x_\alpha$ reaches its target position. Let $\mathcal{P}'$ be the resulting tuple of vertex-disjoint shortest paths. Lemma 2 says that every $\mathcal{P}$-movable token is $\mathcal{P}'$-movable. Since $|T_{\mathcal{P}'}| = |T_\mathcal{P}| - \alpha$ and $\alpha \geq 2$, we can apply the induction hypothesis to $(G, \mathcal{P}', \mathcal{Q})$. □

3.2 Proof of Theorem 2

Our proof of Lemma 3 naturally yields an algorithm which finds an actual reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ if exists. Starting from any token t in $T_{\mathcal{P}}$, we traverse $G_{\mathcal{P}}$ by repeatedly visiting the target position of the currently visited token; if we reach a vertex which is not occupied by any token (that is, the vertex is not contained in any path in $\mathcal{P}$), then we apply Case (a) of the proof; otherwise we find a directed cycle $x_1, x_2, \ldots, x_\alpha$ to which we apply Case (b). In Case (b) we can find the path $x_\alpha y_1 y_2 \ldots y_\beta$ by a breadth-first search on $G_{\mathcal{P}}$ starting from any vertex in the directed cycle.

We now estimate the length of our reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$. Recall that there are $O(dk)$ tokens, where d is the diameter of G and k is the number of terminal pairs. In Case (a), one token in $T_{\mathcal{P}}$ reaches its target position by one step. In Case (b), α tokens in $T_{\mathcal{P}}$ reach their target positions by $(\beta-1)+1+(\alpha-1)+1+(\beta-1) = \alpha+2\beta-1 = O(dk)$ steps. Therefore, in both cases, at least one token in $T_{\mathcal{P}}$ reaches its target position by $O(dk)$ steps. Since there are $O(dk)$ tokens, the length of our reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ can be bounded by $O(dk) \times O(dk) = O(d^2k^2)$ in total.

Finally, we estimate the running time of the algorithm. By Lemma 1 we can check if a given graph G is st-complete for all k terminal pairs, and construct layers in $O(k(n+m))$ time. Since $\sum_{j=1}^{d(s_i,t_i)} |L_j^i| \leq n$ for each $i \in \{1, 2, \ldots, k\}$, the auxiliary graph $G_{\mathcal{P}}$ has at most nk arcs. For each token in $T_{\mathcal{P}}$, we traverse $G_{\mathcal{P}}$ at most twice to apply Case (a) or (b). Thus, we can move at least one token in $T_{\mathcal{P}}$ to its target position in $O(nk) + O(dk) = O(nk)$ time. Since there are $O(dk)$ tokens, all tokens can be moved to their target positions in $O(nk) \times O(dk) = O(ndk^2)$ time. In total, our algorithm runs in $O(mk + ndk^2)$ time. This completes the proof of Theorem 2. □

4 Shortest variant

Our polynomial-time algorithm in Section 3 for the RVDSP problem does not always return a reconfiguration sequence of the shortest length. Indeed, we will show in this section that the SRVDSP problem is NP-complete even for split graphs, while the RVDSP problem is solvable in polynomial time for split graphs.

4.1 Polynomial-time solvable cases

We first give tractable cases of the SRVDSP problem, based on the algorithm in Section 3. We say that k terminal pairs are *identical* if $(s_1, t_1) = (s_2, t_2) = \cdots = (s_k, t_k)$. Then, we have the following theorem.

Theorem 3 *Let $(G, \mathcal{P}, \mathcal{Q}, \ell)$ be an instance of the* SRVDSP *problem such that k terminal pairs are identical and G is st-complete for the terminal pairs. Then, the* SRVDSP *problem is solvable in polynomial time.*

Proof: By Theorem 2 we first check if there is a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$; if not, $(G, \mathcal{P}, \mathcal{Q}, \ell)$ is a no-instance. Since k terminal pairs are identical, let $(s, t) = (s_1, t_1) = (s_2, t_2) = \cdots = (s_k, t_k)$, and let L_j denote the j-th st-layer for each $j \in \{0, 1, \ldots, d_G(s, t)\}$. Then, we have $L_j = L_j^1 = L_j^2 = \cdots = L_j^k$ for all j.

Recall our algorithm in Section 3. First consider Case (a) in the proof of Lemma 3. Then, the algorithm moves a token in $T_{\mathcal{P}}$ to its target position directly by one step. Since every token in $T_{\mathcal{P}}$

must be moved at least once in any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$, this step preserves the shortest length.

We thus consider Case (b) in the proof of Lemma 3. In this case, the algorithm finds a directed cycle $C = x_1 x_2 \dots x_\alpha$ in $G_\mathcal{P}$ such that x_{r+1} is the target position of the token placed on x_r for all $r \in \{1, 2, \dots, \alpha\}$, where $\alpha \geq 2$. (See Figure 4 again.) Then, the definition of $G_\mathcal{P}$ implies that all vertices $x_1, x_2, \dots, x_\alpha$ are contained in the same st-layer L_j for some j, because $L_j = L_j^1 = L_j^2 = \cdots = L_j^k$. Furthermore, there exists at least one vertex $y_1 \in L_j \setminus V(\mathcal{P})$, because tokens placed on the vertices in C are $\mathcal{P}$-movable. Then, $G_\mathcal{P}$ has an arc (x_r, y_1) for each $r \in \{1, 2, \dots, \alpha\}$. We take the arc (x_α, y_1) as a $\mathtt{t}$-escape path for the token $\mathtt{t}$ placed on x_α, that is, $\beta = 1$ in Figure 4. Then, α tokens in $T_\mathcal{P}$ can reach their target positions by $\alpha + 2\beta - 1 = \alpha + 1$ steps. Since any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ needs to move at least one token placed on $V(C)$ to some vertices in $L_j \setminus V(\mathcal{P})$, this step also preserves the shortest length. □

Similarly as in Corollary 1, we have the following corollary from Theorem 3. Note that, if $k = 1$, then it is an identical terminal pair.

Corollary 2 *The* SRVDSP *problem is solvable in polynomial time for split graphs, and for distance-hereditary graphs, if k terminal pairs are identical. In particular, the shortest variant of the* SPR *problem is solvable in polynomial time for split graphs, and for distance-hereditary graphs.*

4.2 NP-completeness

We finally prove the following theorem.

Theorem 4 *The* SRVDSP *problem is* NP*-complete for split graphs.*

By Theorem 2, recall that the RVDSP problem for split graphs can be solved in polynomial time, and admits a reconfiguration sequence of length $O(k^2)$ if exists. Therefore, the SRVDSP problem for split graphs belongs to the class NP. As a proof of Theorem 4, we will thus prove that the SRVDSP problem is NP-hard for split graphs, by giving a polynomial-time reduction from 3SAT [5].

Suppose that we are given a 3CNF formula ϕ, where each clause consists of exactly three literals. Let α and β be the numbers of variables and clauses in ϕ, respectively. We write $x_1, x_2, \dots, x_\alpha$ for variables in ϕ, and $C_1, C_2, \dots, C_\beta$ for clauses in ϕ. We will construct the corresponding graph G_ϕ which forms a split graph. Recall that a split graph G_ϕ can be partitioned into a clique and an independent set. In the following, we call a vertex in the clique a *clique vertex*, and call a vertex in the independent set an *independent vertex*. In our reduction, independent vertices will be terminals. As shown in Corollary 1, a split graph G_ϕ is st-complete for any terminal pair (s, t). Therefore, roughly speaking, we will focus on how to move tokens placed on clique vertices in G_ϕ.

4.2.1 Reduction

We first create a *variable gadget* G_{x_i} for each variable x_i in ϕ. The variable gadget G_{x_i} has five clique vertices $a_i, b_i, c_i, x_i^\top, x_i^\bot$, and eight independent vertices $s_{i1}, s_{i2}, s_{i3}, s_{i4}, t_{i1}, t_{i2}, t_{i3}, t_{i4}$. Then, we define four vertex sets $L^{i1}, L^{i2}, L^{i3}, L^{i4}$, as follows:

$$L^{i1} = \{a_i, b_i, x_i^\top\},\ L^{i2} = \{a_i, b_i, x_i^\bot\},\ L^{i3} = \{c_i, x_i^\top\},\ L^{i4} = \{c_i, x_i^\bot\}.$$

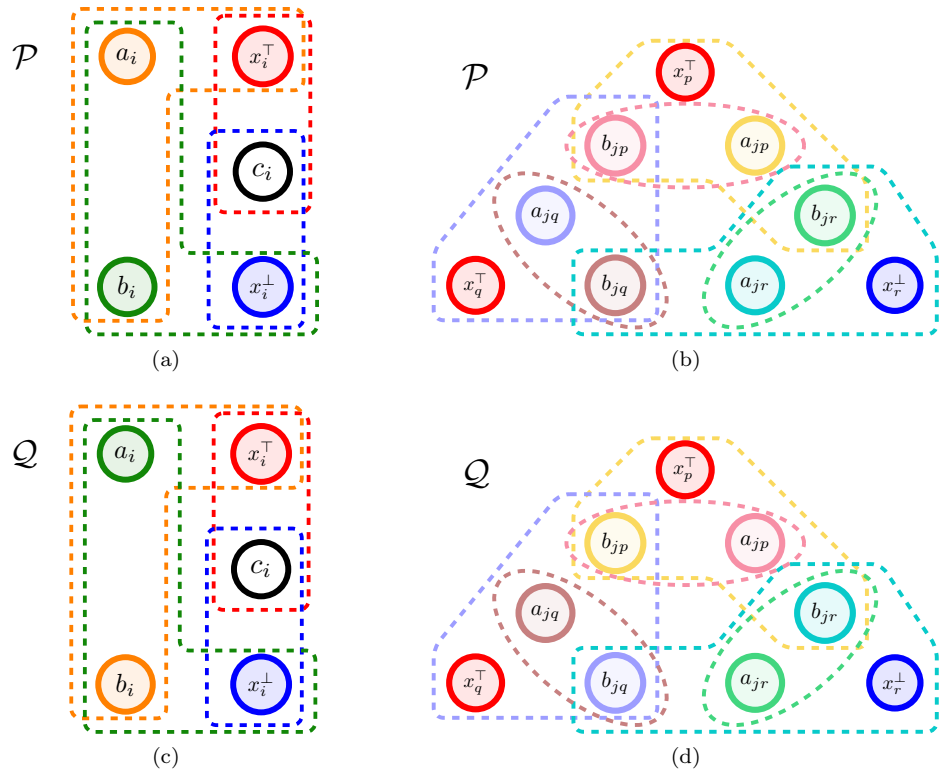

Figure 5: (a)(c) Clique vertices in the variable gadget G_{x_i}, and (b)(d) clique vertices in the clause gadget G_{C_i} for $C_i = (x_p \vee x_q \vee \neg x_r)$, where we omit the edges in the graph.

In Figure 5(a) and (c), we illustrate only clique vertices in G_{x_i}, and each colored box in the figure surrounding vertices corresponds to the sets above. We join the clique and independent vertices in G_{x_i} so that each L^{ir} forms the first $s_{ir}t_{ir}$-layer for the terminal pair (s_{ir}, t_{ir}), $r \in \{1,2,3,4\}$; more specifically, for each $r \in \{1,2,3,4\}$, we join each of s_{ir} and t_{ir} with all clique vertices in L^{ir}. For each $r \in \{1,2,3,4\}$, we define shortest $s_{ir}t_{ir}$-paths P_{ir} and Q_{ir}, as follows:

$$P_{i1} = s_{i1}a_it_{i1}, \qquad Q_{i1} = s_{i1}b_it_{i1},$$
$$P_{i2} = s_{i2}b_it_{i2}, \qquad Q_{i2} = s_{i2}a_it_{i2},$$
$$P_{i3} = Q_{i3} = s_{i3}x_i^\top t_{i3},$$
$$P_{i4} = Q_{i4} = s_{i4}x_i^\perp t_{i4}.$$

Notice that only the internal vertices of P_{i1} and P_{i2} are swapped in Q_{i1} and Q_{i2}. By the construction, it suffices to focus on which clique vertex is chosen as an internal vertex of a shortest $s_{ir}t_{ir}$-path. In Figure 5(a) and (c), each colored box represents the vertices that can be an internal vertex of shortest $s_{ir}t_{ir}$-paths, and assume that a token with the same color is placed on a vertex when the vertex is chosen as the internal vertex of a shortest $s_{ir}t_{ir}$-path; the vertex colored with white (i.e., the vertex c_i in the figure) is not chosen by any shortest path, and no token is placed on it. For example, the orange box in the figure represents the the first $s_{i1}t_{i1}$-layer $L^{i1} = \{a_i, b_i, x_i^\top\}$, and the orange token represents the internal vertex a_i (resp. b_i) of the shortest $s_{i1}t_{i1}$-path P_{i1}

(resp. Q_{i1}). By the construction of the gadgets, any shortest $s_{ir}t_{ir}$-path passes through one of the vertices in L^{ir} for all $r \in \{1, 2, 3, 4\}$. Thus, any token cannot be moved to a vertex outside the box of the same color.

We then create a *clause gadget* G_{C_j} for each clause C_j in ϕ. Let $C_j = (l_{jp} \vee l_{jq} \vee l_{jr})$, where l_{jh} is either x_h or $\neg x_h$ for each $h \in \{p, q, r\}$. Let $\sigma(j, h)$ denote $\top$ if $l_{jh} = x_h$, otherwise $\bot$. The clause gadget G_{C_j} has six new clique vertices a_{jh}, b_{jh} where $h \in \{p, q, r\}$, and three clique vertices $x_p^{\sigma(j,p)}, x_q^{\sigma(j,q)}, x_r^{\sigma(j,r)}$ which are already introduced in variable gadgets. In addition, G_{C_j} has 12 new independent vertices $s_{jh1}, s_{jh2}, t_{jh1}, t_{jh2}$ where $h \in \{p, q, r\}$. We define six vertex sets, as follows (see also Figure 5(b) and (d)):

$$
\begin{aligned}
L^{jp1} &= \{x_p^{\sigma(j,p)},\ b_{jp},\ a_{jp},\ b_{jr}\},\\
L^{jq1} &= \{x_q^{\sigma(j,q)},\ b_{jq},\ a_{jq},\ b_{jp}\},\\
L^{jr1} &= \{x_r^{\sigma(j,r)},\ b_{jr},\ a_{jr},\ b_{jq}\},\\
L^{jh2} &= \{a_{jh},\ b_{jh}\} \quad \text{for } h \in \{p, q, r\}.
\end{aligned}
$$

Similarly as in the variable gadgets, we join the clique and independent vertices in G_{C_j}, as follows: for each $h \in \{p, q, r\}$, we join each of s_{jh1} and t_{jh1} with all clique vertices in L^{jh1}, and also join each of s_{jh2} and t_{jh2} with all clique vertices in L^{jh2}. For each $h \in \{p, q, r\}$, we define shortest $s_{jh1}t_{jh1}$-paths P_{jh1} and Q_{jh1}, and shortest $s_{jh2}t_{jh2}$-paths P_{jh2} and Q_{jh2}, as follows:

$$
\begin{aligned}
P_{jh1} &= s_{jh1}a_{jh}t_{jh1}, & Q_{jh1} &= s_{jh1}b_{jh}t_{jh1},\\
P_{jh2} &= s_{jh2}b_{jh}t_{jh2}, & Q_{jh2} &= s_{jh2}a_{jh}t_{jh2}.
\end{aligned}
$$

We next join all clique vertices in variable and clause gadgets so that they form a clique in G_ϕ. This completes the constructions of G_ϕ, $\mathcal{P}$, and $\mathcal{Q}$. Then, there are $k = 4\alpha + 6\beta$ terminal pairs, and we set $\ell = 5\alpha + 9\beta$. In this way, the corresponding instance $(G_\phi, \mathcal{P}, \mathcal{Q}, \ell)$ of the SRVDSP problem can be constructed in time polynomial in α and β.

4.2.2 Correctness of the reduction

We first observe that there is a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$. To see this, by Lemma 3 it suffices to show that all tokens in $T_\mathcal{P}$ (placed on the swapped vertices between $\mathcal{P}$ and $\mathcal{Q}$) are $\mathcal{P}$-movable. Notice that $V(G_\phi) \setminus V(\mathcal{P})$ consists of only α vertices c_i in variable gadgets G_{x_i}, $i \in \{1, 2, \ldots, \alpha\}$. In the variable gadget G_{x_i}, each token placed on a_i (resp. b_i) has an escape path under $\mathcal{P}$ to c_i via $x_i^\top$ (resp. $x_i^\bot$). In the clause gadget G_{C_j}, tokens placed on a_{jh} and b_{jh} have escape paths under $\mathcal{P}$ to c_h via $x_h^{\sigma(j,h)}$, where $h \in \{p, q, r\}$. Therefore, all tokens in $T_\mathcal{P}$ are $\mathcal{P}$-movable, and hence there is a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$.

Before proving the correctness of our reduction, we roughly explain how our gadgets work. In order to swap the tokens on a_i and b_i in a variable gadget G_{x_i}, we need to move a token placed on either $x_i^\top$ or $x_i^\bot$ to c_i. Note that this token movement is required in any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$. If we move the token placed on $x_i^\top$ (resp. $x_i^\bot$) to c_i, we can also exchange *all* tokens in the clause gadget G_{C_j} to their target positions if the corresponding clause C_j has the literal x_i (resp. $\neg x_i$). Therefore, if ϕ is satisfiable, we can save token movements required for moving the tokens in G_{C_j} to their target positions. Formally, we prove the following lemma, which completes the proof of Theorem 4.

Lemma 4 *ϕ is satisfiable if and only if $(G_\phi, \mathcal{P}, \mathcal{Q}, \ell)$ is a yes-instance.*

Proof: We first prove the only-if direction. Let $\psi\colon \{x_1, x_2, \ldots, x_\alpha\} \to \{\top, \bot\}$ be a truth assignment which satisfies all clauses of ϕ, where $\top$ corresponds to true and $\bot$ to false. We construct a reconfiguration sequence of length at most ℓ, as follows (see also Figure 5):

Step 1. Move the token on $x_i^{\psi(x_i)}$ to c_i for each $i \in \{1, 2, \ldots, \alpha\}$.

Step 2. Move the tokens on a_i and b_i to their target positions using the vertex $x_i^{\psi(x_i)}$.

Step 3. Since each clause $C_j = (l_{jp} \vee l_{jq} \vee l_{jr})$ has at least one literal, say l_{jp}, satisfied by ψ, there is at least one vertex $x_p^{\psi(x_p)}$ on which no token is placed. Then, the path $b_{jr}\ a_{jr}\ b_{jq}\ a_{jq}\ b_{jp}\ a_{jp}\ x_p^{\psi(x_p)}$ forms an escape path, and we can move the tokens through this escape path.

Step 4. Move the tokens on b_{jq}, b_{jp} and $x_p^{\psi(x_p)}$ to their target positions.

Step 5. Move back the token on c_i to $x_i^{\psi(x_i)}$ for each $i \in \{1, 2, \ldots, \alpha\}$.

In this way, we obtain a reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$, whose length is

$$\alpha + 3\alpha + 6\beta + 3\beta + \alpha = 5\alpha + 9\beta = \ell.$$

Therefore, $(G_\phi, \mathcal{P}, \mathcal{Q}, \ell)$ is a yes-instance.

We then prove the if direction. Suppose that there is a reconfiguration sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{\ell'} \rangle$, where $\mathcal{P}_0 = \mathcal{P}$, $\mathcal{P}_{\ell'} = \mathcal{Q}$ and $\ell' \leq \ell = 5\alpha + 9\beta$. Then, we define a truth assignment $\psi\colon \{x_1, x_2, \ldots, x_\alpha\} \to \{\top, \bot\}$, as follows:

$$\psi(x_i) = \begin{cases} \top & \text{if there is a tuple } \mathcal{P}_\lambda \text{ in } \langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{\ell'} \rangle \text{ such that } x_i^\top \notin V(\mathcal{P}_\lambda); \\ \bot & \text{otherwise.} \end{cases} \tag{2}$$

As a proof of the if direction, we will show that ψ satisfies all clauses in ϕ.

We first observe that there are $(\alpha + 3\beta)$ pairs of tokens that need to be swapped between $\mathcal{P}$ and $\mathcal{Q}$: one pair of tokens placed on a_i and b_i in each variable gadget G_{x_i}, $i \in \{1, 2, \ldots, \alpha\}$; and three pairs of tokens placed on a_{jh} and b_{jh}, $h \in \{p, q, r\}$, in each clause gadget G_{C_j}, $j \in \{1, 2, \ldots, \beta\}$. Since swapping a pair of tokens requires at least three token movements, any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ requires at least $3(\alpha + 3\beta)$ token movements. Furthermore, recall that only the vertex c_i in each variable gadget G_{x_i}, $i \in \{1, 2, \ldots, \alpha\}$, is unoccupied by a token, and only two tokens placed on $x_i^\top$ and $x_i^\bot$ can move to c_i. Therefore, any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ needs to move at least one of the two tokens to c_i, and move back the token to the original position; that is, 2α token movements are required. In total, any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ requires at least

$$3(\alpha + 3\beta) + 2\alpha = 5\alpha + 9\beta = \ell$$

token movements. (Thus, we indeed know that $\ell' = \ell$.)

By the arguments above, we can conclude that exactly one of the two tokens placed on $x_i^\top$ and $x_i^\bot$ moves to c_i in every variable gadget G_{x_i}, $i \in \{1, 2, \ldots, \alpha\}$; otherwise the reconfiguration sequence is of length more than ℓ. Therefore, the truth assignment ψ defined by (2) is consistent with the reconfiguration sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{\ell'} \rangle$. Furthermore, recall that all vertices in the clause gadgets G_{C_j} for each clause $C_j = (l_{jp} \vee l_{jq} \vee l_{jr})$ are occupied by tokens. Thus, any reconfiguration sequence between $\mathcal{P}$ and $\mathcal{Q}$ needs to move at least one of tokens placed on $x_p^{\sigma(j,p)}$, $x_q^{\sigma(j,q)}$ and $x_r^{\sigma(j,r)}$. This ensures that every clause in ϕ has at least one true literal, and hence ψ satisfies all clauses in ϕ. □

5 Conclusions

In this paper, we introduced two reconfiguration problems, the RVDSP and SRVDSP problems, as generalizations of the well-studied SPR problem. We studied the computational complexity of the RVDSP and SRVDSP problems from the viewpoint of graph classes, and gave some interesting contrast. (Recall Figure 2 and the numbered list at the end of the introduction.)

It remains open to clarify the complexity status of the RVDSP problem for chordal graphs, and for planar graphs. Indeed, our algorithm in Section 3 does not work even for interval graphs, which form a subclass of chordal graphs. Note that the SPR problem (i.e., the RVDSP problem for $k = 1$) is solvable in polynomial time for chordal graphs [2], and for planar graphs [3]. Another interesting viewpoint is the parameterized complexity of the RVDSP and SRVDSP problems. For example, it is unclear yet whether the problem admits an XP-algorithm for planar graphs when parameterized by k.

Acknowledgements

We are grateful to Kai Matsudate, Tohoku University, Japan, for valuable discussions with him. We also thank anonymous referees of the conference version [10] and this journal version for their helpful comments.

This work is partially supported by JSPS KAKENHI Grant Numbers JP18H04091, JP19K11814, JP20H05793, JP20H05961, JP20H05964, JP20K11673, JP24H00686, and JP24H00690.

References

[1] J. Asplund, K. D. Edoh, R. Haas, Y. Hristova, B. Novick, and B. Werner. Reconfiguration graphs of shortest paths. *Discrete Mathematics*, 341(10):2938–2948, 2018. doi:10.1016/j.disc.2018.07.007.

[2] P. S. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1–12, 2013. doi:10.1016/j.tcs.2013.09.012.

[3] P. S. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017. doi:10.1016/j.dam.2016.05.024.

[4] K. Gajjar, A. V. Jha, M. Kumar, and A. Lahiri. Reconfiguring shortest paths in graphs. In *Proceedings of AAAI 2022*, pages 9758–9766. AAAI Press, 2022. doi:10.1609/aaai.v36i9.21211.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] J. van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

[7] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

[8] M. Kamiński, P. Medvedev, and M. Milanič. Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210, 2011. doi:10.1016/j.tcs.2011.05.021.

[9] N. Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):Paper id 52, 2018. doi:10.3390/a11040052.

[10] R. Saito, H. Eto, T. Ito, and R. Uehara. Reconfiguration of vertex-disjoint shortest paths on graphs. In *Proceedings of WALCOM 2023*, pages 191–201, 2023. doi:10.1007/978-3-031-27051-2_17.

[11] M. Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.