Reconfiguration Problems for Graph Coloring, Homomorphism, and Constraint Satisfaction

by
Tatsuhiko Hatanaka

Submitted to
Department of System Information Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Information Science)

Graduate School of Information Sciences
Tohoku University, Japan

2019

Acknowledgements

First of all, I would like to express my deep gratitude for my supervisor Professor Xiao Zhou. He gave me a really large amount of worthful advices and suggestions, and offered a very comfortable place that made my life of research so pleasant. Without his continual help and encouragement, this thesis would not be completed.

I am really grateful to my thesis advisor, Associate Professor Takehiro Ito. He also helped me on the presentations and on the writing style of my papers, and gave me a lot of opportunities to build up my career as a researcher. Without his numerous number of helpful discussions and supports, this thesis would not be completed.

I would also like to show my special thanks to the other members of my graduate committee, Professor Takeshi Tokuyama and Professor Ayumi Shinohara, for their insightful suggestions and comments.

Furthermore, I would also like to thank Assistant Professor Akira Suzuki for his many suggestions and guidance, and for many things. All of his useful suggestions on my research and his kind help in everyday life have been invaluable to me.

Finally, I would like to acknowledge with sincere thanks the all-out cooperation and services rendered by the members of Algorithm Theory Laboratory for many things.

Abstract

Since the 2000s, the framework of (combinatorial) reconfiguration has been extensively studied in the field of theoretical computer science. This framework models several "dynamic" situations where we wish to find a step-by-step transformation between two feasible solutions of a combinatorial search problem such that all intermediate solutions are also feasible and each step respects a fixed reconfiguration rule. This reconfiguration framework has been applied to several well-studied combinatorial search problems. In this thesis, we mainly study the reconfiguration problem for the well-known constraint satisfaction problem (CSP), which is a generalization of several combinatorial search problems including graph coloring, Boolean satisfiability, graph homomorphism and so on. In CSP, we are asked to find an assignment of values to given variables so as to satisfy all of given constraints. In the reconfiguration problem for CSP, we are given an instance of CSP together with its two satisfying assignments, and asked to determine whether one assignment can be transformed into the other by changing a single variable assignment at a time, while always remaining satisfying assignment. We also study several special cases of the problem, especially the reconfiguration problems for graph coloring, graph homomorphism, and their list variants. In this thesis, we study these problems from the viewpoints of polynomial-time solvability and parameterized complexity, and give several interesting boundaries of tractable and intractable cases.

Contents

List of Figures

Chapter 1 Introduction

1.1 Combinatorial reconfiguration

Since the 2000s, the framework of (*combinatorial*) *reconfiguration* [34] has been extensively studied in the field of theoretical computer science. (See surveys [38, 50, 33].) This framework models several "dynamic" situations where we wish to find a step-by-step transformation between two feasible states such that all intermediate states are also feasible and each step respects a fixed reconfiguration rule. For example, *sliding block puzzles* [32] such as the *15-puzzle* (see Figure 1.1) can be captured in this framework; the feasible states are placements of rectangle blocks in a rectangle frame without an overlap, and a reconfiguration rule is sliding exactly one block to an empty cell at a time. As another example, consider the situation where frequency channels are assigned to base stations so that no interference occurs. Assume now that the current feasible assignment must be transformed (e.g., to use a newly found better assignment or to satisfy new side constraints), while maintaining the feasibility (so that the users receive service even during the transformation). This problem can also be modeled in the reconfiguration framework; the feasible states are feasible frequency assignments, and a reconfiguration rule may be reassigning a frequency channel of a single base station.

Generally, a (*combinatorial*) *reconfiguration problem* can be considered as a problem asking the reachability of two vertices in a *solution graph* (also called a *reconfiguration graph*) defined as follows. The vertex set of a solution graph corresponds to the set of feasible states and the edge set represents a reconfiguration rule. The set of

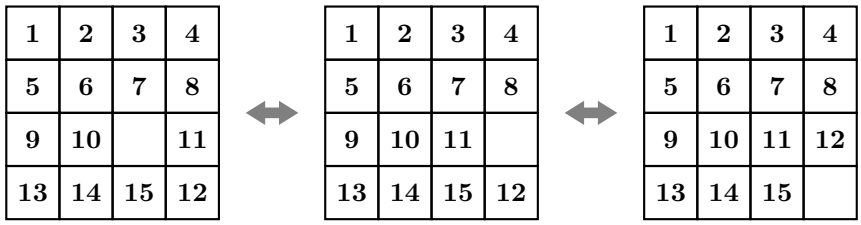


Figure 1.1: The 15-puzzle. Each placement of blocks is feasible and only one block is slid in each step.

feasible states may often be defined as a set of feasible solutions for an instance of a (combinatorial) search problem. Indeed, several reconfiguration problems based on search problems are studied well, such as Boolean satisfiability reconfiguration [26, 44, 47], shortest path reconfiguration [3, 40], independent set reconfiguration [5, 21, 32, 41, 42], vertex cover reconfiguration [37, 48], dominating set reconfiguration [28, 30].

As a concrete example of such reconfiguration problems, we here introduce the (vertex) coloring reconfiguration problem [4, 7, 17, 60], which is one of the most well-studied reconfiguration problems. (See Figure 1.2 for example.) Let $G = (V, E)$ be a graph and let C be a set of k *colors*. A *k-coloring* (or simply a vertex coloring) of G is a mapping $f\colon V \to C$ such that $f(v) \neq f(w)$ holds for every edge $vw \in E$. In coloring reconfiguration, the vertex set of a solution graph is the set of all k-colorings of G, and two vertices f and f' are adjacent in the solution graph if f' is obtained from f by changing the color assignment on a single vertex, and vice versa. Then, for a given graph G, a color set C of size k, and two k-colorings f_s and f_t of G, the problem asks whether there exists a walk between f_s and f_t in the solution graph for G and C; such a walk is called a *reconfiguration sequence*.

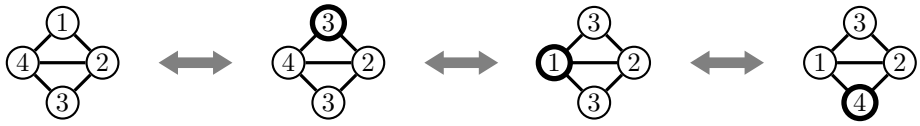


Figure 1.2: A transformation of 4-colorings. A vertex which is recolored from the previous 4-coloring is depicted by a thick circle.

1.2 Applications

The framework of reconfiguration has many applications as the "feasibility" is preserved during the transformation.

1.2.1 Dynamic transformation of system configurations

Recall the transformation of frequency assignments, which we presented in the beginning of the chapter. Actually, this problem can be modeled by COLORING RECONFIGURATION if we consider each base station as a vertex, and each frequency channel as a color, and joining each pair of two base stations which have the high potential of interference. Notice that a frequency assignment without interference corresponds to a k-coloring, and reassigning a frequency channel on a single base station corresponds to changing the color assignment on a single vertex. Similar situations may occur when we wish to dynamically transform a configuration of a system into another one while maintaining the feasibility. Another example is POWER SUPPLY RECONFIGURATION [34], which models the switching of the power supply routing between power stations and customers. Thus, the reconfiguration framework can be applied to several practical issues.

1.2.2 Feedback to search problems

We here note that a research on a reconfiguration problem may provide us a deeper understanding of the "solution space" of the corresponding search problem, since we deal not only with the existence of solutions but also with the relationship

between them. Thus, sometimes it brings a good feedback to the search problem. For example, Wrochna [12] gave another (much shorter) proof of the theorem that any graph without a cycle of length dividable by three has a 3-coloring, which was originally proved by Chen and Saito [18] in 1994, using ideas provided to solve 3-COLORING RECONFIGURATION [17]. Roughly speaking, the proof uses the fact that the reconfiguration preserves the feasibility.

1.3 Problems studied in this thesis

In this thesis, we mainly study (LIST) COLORING RECONFIGURATION, (LIST) HOMOMORPHISM RECONFIGURATION, and CONSTRAINT SATISFIABILITY RECONFIGURATION from the viewpoints of polynomial-time solvability and parameterized complexity. The formal definitions of the problems will be given in Chapter 2, but we here briefly introduce them.

LIST COLORING RECONFIGURATION is a generalization of COLORING RECONFIGURATION where we are given a *list* of allowed colors for each vertex as an additional input and feasible solutions are k-colorings which respect all lists. (LIST) HOMOMORPHISM RECONFIGURATION is a generalization of (LIST) COLORING RECONFIGURATION defined as follows. In addition to a color set, we are also given a set of pairs of colors which can be assigned to adjacent vertices in G. This is represented by a graph H which has C as the vertex set and the set of pairs as the edge set, called an underlying graph.[1] Then, feasible solutions are k-colorings which respect the edge set H (and all lists). We note that (LIST) HOMOMORPHISM RECONFIGURATION is equivalent to (LIST) COLORING RECONFIGURATION when an underlying graph H is a complete graph.

CONSTRAINT SATISFIABILITY RECONFIGURATION is the reconfiguration problem

[1] In this thesis, we only consider simple undirected underlying graphs unless otherwise stated, although loops and/or directed edges are often allowed in the literature of graph homomorphisms.

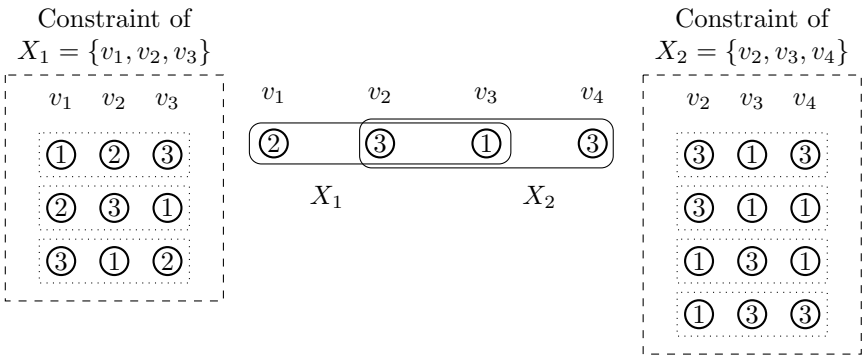


Figure 1.3: An example of constraints which represent allowed assignments to the vertices in CSP (left and right of the figure) and a mapping which satisfies all constraints (middle of the figure).

for well-known constraint satisfaction problem (CSP, for short), which is a generalization of several combinatorial problems including vertex coloring, Boolean satisfiability, graph homomorphism and so on. In this thesis, we formulate them by means of hypergraphs. Let $G = (V, E)$ be a hypergraph. Let D be a set, called a *domain*; each element of D is called a *value* and we always denote by k the size of a domain. In CSP, each hyperedge $X \in E$ has a *constraint* which represents the values allowed to be assigned to the vertices in X at the same time, and we wish to find a mapping $f\colon V \to D$ which satisfies the constraints of all hyperedges in G. (See Figure 1.3 for an example.) For example, in the case of vertex coloring, we can see that every hyperedge consists of two vertices, and has the common constraint that any two different colors can be assigned to the two vertices in the hyperedge at the same time. In CONSTRAINT SATISFIABILITY RECONFIGURATION, feasible solutions are mappings satisfying all constraints, and the reconfiguration rule is changing a value of a single vertex at a time. BOOLEAN CONSTRAINT SATISFIABILITY is the special case of CONSTRAINT SATISFIABILITY where $|D| = 2$. For an integer $r \geq 1$, r-ARY CONSTRAINT SATISFIABILITY is the special case of CONSTRAINT SATISFIABILITY where all hyperedges have size at most r.

As we will discuss in Chapter 2, CONSTRAINT SATISFIABILITY RECONFIGURA-

tion includes several reconfiguration problems as special cases such as Boolean constraint satisfiability reconfiguration, r-ary constraint satisfiability reconfiguration, (list) homomorphism reconfiguration, (list) coloring reconfiguration. In the remainder of the thesis, we use the following abbreviations:

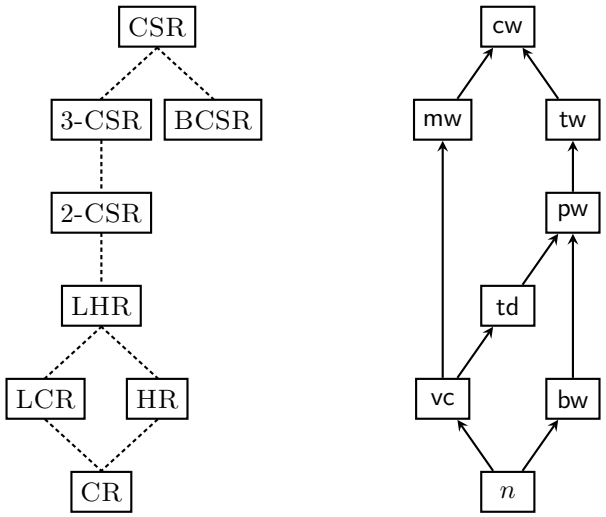
- CSR for constraint satisfiability reconfiguration;
- BCSR for Boolean constraint satisfiability reconfiguration;
- r-CSR for r-ary constraint satisfiability reconfiguration for each integer $r \geq 1$;
- (L)HR for (list) homomorphism reconfiguration; and
- (L)CR for (list) coloring reconfiguration.

Relationships between problems are illustrated in Figure 1.4(a).

1.4 Known and related results

There are many literatures which study special cases of CSR (including (L)CR and (L)HR) and their *shortest variants*. In the shortest variant, we are given an instance with an integer $\ell \geq 0$, and asked whether there exists a reconfiguration sequence of length at most ℓ. We here state only the results from the viewpoint of the computational complexity.

One of the most well-studied special cases of CSR is BCSR [6, 14, 26, 43, 44, 47, 57]. Gopalan et al. [26] gave a computational dichotomy for BCSR with respect to a set $\mathcal{S}$ of logical relations which can be used to define each constraint: the problem is PSPACE-complete or in P for each fixed $\mathcal{S}$. In addition, Cardinal et al. [14] showed that the problem remains PSPACE-complete even if $\mathcal{S}$ is equivalent to monotone Not-All-Equal 3-SAT (i.e., each constraint is a set of surjections)



(a) (b)

Figure 1.4: (a) Relationships between problems. Each dotted line between P (lower) and Q (upper) means that P is a special case of Q. (b) Relationships between graph parameters. cw, mw, tw, pw, td, vc, bw and n are the cliquewidth, the modular-width, the treewidth, the pathwidth, the tree-depth, the size of a minimum vertex cover, the bandwidth and the number of vertices of a graph, respectively. Each arrow $\alpha \to \beta$ means that α is stronger than β, that is, if α is bounded by a constant then β is also bounded by some constant.

and a "variable-clause incidence graph" is planar. For the shortest variant, a computational trichotomy is known: Mouawad et al. [47] proved that the problem is PSPACE-complete, NP-complete or in P for each fixed $\mathcal{S}$. Bonsma et al. [6] proved that the shortest variant is $W[2]$-hard when parameterized by ℓ even if $\mathcal{S}$ is equivalent to Horn SAT.

Another well-studied spacial case is CR [1, 2, 4, 6, 7, 8, 15, 16, 17, 23, 24, 27, 39, 51, 52, 60]. This problem is PSPACE-complete for $k \geq 4$ and bipartite planar graphs [4] but in P for $k \leq 3$ [17]. The second tractability result is extended to the shortest variant [39], and even can be applied to LCR. Moreover, it is known that the problem remains PSPACE-complete even if k is a fixed constant for line graphs (for any fixed $k \geq 5$) [52] and for bounded bandwidth graphs [60]. On the other hand, there exists a polynomial-time algorithm for $(k-2)$-connected chordal

graphs [7]. For the shortest variant parameterized by ℓ, some intractability results are known; it is $W[1]$-hard [6] and does not admit a polynomial kernelization when k is fixed unless the polynomial hierarchy collapses [39]. As a generalization of CR, LCR is also studied [36, 52, 60]. The problem is PSPACE-complete even for graphs with pathwidth two and constant bandwidth and some constant k [60] and for line graphs and any fixed $k \geq 4$ [52].

HR is also well-studied as a generalization of CR. Several literatures investigated HR from the viewpoint of graph classes in which a given graph G or an underlying graph H lies [11, 10, 12, 13, 59, 60]. Brewster et al. [12] gave a dichotomy for a special case of HR in which H is a (p, q)-circular clique; it is PSPACE-complete if $p/q \geq 4$ but is in P otherwise. We note that this result generalizes the complexity of CR with respect to k, since a complete graph K_k with k vertices is a $(k, 1)$-circular clique. It is also known that the problem is PSPACE-complete even if (a) H is an odd wheel [10] or (b) H is some fixed graph and G is a cycle [60]. On the other hand, it can be solved in polynomial time if G is a tree [60] or H contains no cycles of length four [59]. Furthermore, a fixed-parameter algorithm is known when parameterized by $k + \mathsf{td}$ [60]; note that it can be easily extended for LHR.

Finally, we refer to the shortest variant of CSR. Bonsma et al. [6] gave a fixed-parameter algorithm for the shortest variant parameterized by $k + r + \ell$, where r is the maximum size of a hyperedge. This implies that shortest variants of BCSR and 2-CSR are fixed-parameter tractable when parameterized by $r + \ell$ and $k + \ell$, respectively. They also showed that the problem is intractable in general if at least one of $\{k, r, \ell\}$ is excluded from the parameter.

1.5 Our contribution

In this thesis, we investigate the complexity of CSR and its spacial cases, especially 3-CSR, 2-CSR, (L)HR and (L)CR, from the viewpoints of polynomial-time

Table 1.1: Computational complexities with respect to the size k of a domain.

	$k \geq 4$	$k = 3$	$k = 2$
CSR	PSPACE-c.	PSPACE-c.	PSPACE-c.
3-CSR	PSPACE-c.	PSPACE-c.	PSPACE-c. [26]
2-CSR	PSPACE-c.	PSPACE-c. [Thm. 6.1]	P [Thm. 6.2]
LHR	PSPACE-c.	P [Thm. 5.2]	P
LCR	PSPACE-c.	P [17]	P
HR	PSPACE-c.	P [59]	P
CR	PSPACE-c. [4]	P	P

solvability and parameterized complexity, and give several interesting boundaries of tractable and intractable cases.

1.5.1 Polynomial-time solvability

We first classify the complexity of the problems for each fixed size k of a domain; we note that k corresponds to the number of colors in (L)CR and (L)HR. Together with known results, our results give interesting boundaries of (in)tractability as summarized in Table 1.1. In particular, our results unravel the boundaries with respect to k for 2-CSR and LHR. The other interesting contrast we show is the boundary between 2-CSR and LHR for $k = 3$.

In order to give more detailed analyses, we also focus on the structure of an input (hyper)graph and explore the structures which make the problems hard. We first analyze the complexities of CR and (L)CR from the viewpoint of graph classes. (See Figures 1.5 and 1.6.) In particular, the PSPACE-completeness of LCR for chordal graphs answers the open question posed by Bonsma and Paulusma [7]. Moreover, we show the boundary of the complexity of LCR with respect to pathwidth; we give a polynomial-time algorithm for graphs with pathwidth one, while it is PSPACE-complete for graphs with pathwidth two [60]. We next investigate the complexity for graphs with pathwidth one or two of more ganeral problems, that is, (L)HR, 2-CSR, 3-CSR and CSR. (See Table 1.2.)

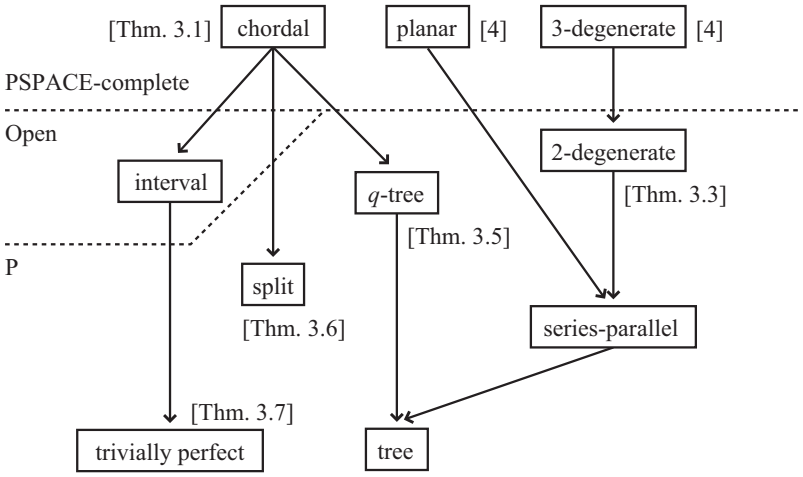


Figure 1.5: Known and our results for CR with respect to graph classes. Each arrow $A \to B$ represents that the graph class B is a subclass of the graph class A.

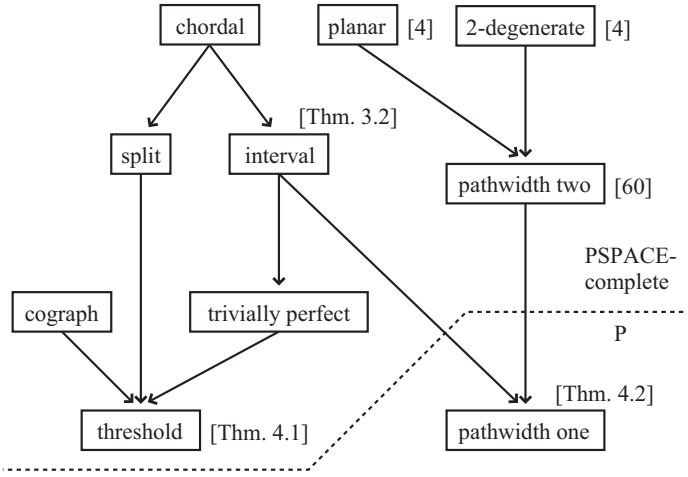


Figure 1.6: Known and our results for LCR with respect to graph classes. Each arrow $A \to B$ represents that the graph class B is a subclass of the graph class A.

1.5.2 Parameterized complexity

As we stated in Section 1.4, most fixed-parameter algorithms (other than the one in [60]) known for CSR and its special cases take the length ℓ of a reconfiguration sequence as a part of the parameter. If we consider somewhat practical situations (e.g., the transformation of frequency assignments), it may be reasonable enough to wish to find a short reconfiguration sequence and thus assume a small ℓ. On the other hand, when the reconfigurability matters, this is not the case, since the short-

Table 1.2: Computational complexity for graphs with pathwidth at most two.

	$\mathsf{pw} = 2$	$\mathsf{pw} = 1$
CSR	PSPACE-c.	PSPACE-c.
3-CSR	PSPACE-c.	PSPACE-c.
2-CSR	PSPACE-c.	PSPACE-c.
LHR	PSPACE-c.	PSPACE-c. [Thm. 5.1]
LCR	PSPACE-c. [60]	P [Thm. 4.2]
HR	PSPACE-c. [60]	P [60]
CR	P [Thm. 3.3]	P

est reconfiguration sequence can be arbitrarily large and even superpolynomial [4]. Therefore, we investigate the parameterized complexity for several parameters which do not contain ℓ.

One may come up with the size k of a domain as such natural parameter. However, the problems are PSPACE-complete even if k is fixed even for several restricted classes, and hence fixed-parameter algorithms parameterized k never exist under P $\neq$ PSPACE. Thus, as other natural parameters, we consider several graph parameters such as the cliquewidth cw, the modular-width mw, the treewidth tw, the pathwidth pw, the tree-depth td, the size vc of a minimum vertex cover, the bandwidth bw, and the number n of vertices; we again define each graph parameter of a hypergraph as that of its primal graph. The relationships between graph parameters are summarized in Figure 1.4(b); note that tractability (resp., intractability) result propagates downward (resp., upward).

We first show that HR parameterized by the number of vertices and LCR parameterized by the size of a minimum vertex cover are both $W[1]$-hard. These imply that fixed-parameter algorithms are unlikely to exist for almost all graph parameters in Figure 1.4. Therefore, we take as parameters k plus several graph parameters, and summarize the known and our results in Table 1.3. We note that a fixed-parameter tractability of CSR parameterized by $k + \mathsf{vc}$ can be obtained as a corollary of Theorem 9.1. However, Theorem 9.3 gives a faster algorithm and Theorem 9.2 gives an

Table 1.3: Parameterized complexity with respect to k plus graph parameters.

Parameter	$k + \mathsf{mw}$	$k + \mathsf{td}$	$k + \mathsf{vc}$	$k + \mathsf{bw}$
CSR	PSPACE-c.	FPT [Thm. 9.1]	FPT [Thms. 9.2, 9.3]	PSPACE-c.
3-CSR	PSPACE-c.	FPT	FPT	PSPACE-c.
2-CSR	PSPACE-c. [Cor. 6.1]	FPT	FPT	PSPACE-c.
LHR	FPT [Thm. 8.1]	FPT	FPT	PSPACE-c.
LCR	FPT	FPT	FPT	PSPACE-c.
HR	FPT	FPT [60]	FPT	PSPACE-c.
CR	FPT	FPT	FPT	PSPACE-c. [60]

Table 1.4: Parameterized complexity with respect to k and the number nb of non-Boolean vertices.

Parameter	$k + \mathsf{nb}$	nb
CSR	PSPACE-c.	PSPACE-c.
3-CSR	PSPACE-c. [26]	PSPACE-c.
2-CSR	FPT [Thm. 9.5]	$W[1]$-hard but XP [Thm. 9.5]
LHR	FPT	$W[1]$-hard but XP
LCR	FPT	$W[1]$-hard but XP
HR	FPT	$W[1]$-hard [Cor. 7.1] but XP
CR	FPT	FPT [Pro. 9.3]

algorithm for the shortest variant.

We next consider another parameter which is not graph-structural, that is, the number nb of "non-Boolean vertices", in order to extend the analysis for $k = 2$ presented in the previous section. Roughly speaking, nb reflects how an instance is close to that of BCSR. The parameterized complexity regarding nb is summarized in Table 1.4. We note that $k = 2$ implies that $\mathsf{nb} = 0$, and hence the algorithm given in Theorem 9.5 generalizes Theorem 6.2. Therefore, this result gives a more detailed boundary of the complexity of 2-CSR between $k = 3$ and $k = 2$.

Finally, we prove that 2-CSR cannot be solved in time $O^*((k+n)^{o(k+n)})$ under the exponential time hypothesis (ETH). This lower bound matches the running time shown in Theorems 2.1, 9.3 and 9.5, respectively.

1.6 Organization of this thesis

The main part of this thesis consists of two parts; each part is divided into chapters according to the target problem.

In Part I, we investigate the polynomial-time solvability. In Chapters 3 and 4, we study CR and LCR, respectively, from the viewpoint of graph classes. In Chapter 5, we show the PSPACE-completeness of LHR for graphs with pathwidth one and give a polynomial-time algorithm for $k = 3$. In Chapter 6, we show the PSPACE-completeness of 2-CSR for $k = 3$ and give a polynomial-time algorithm for $k = 2$.

In Part II, we investigate the parameterized complexity. In Chapter 7, we show two $W[1]$-hardness results for HR and LCR. In Chapter 8, we give a fixed-parameter algorithm for LHR parametrized by $k + \mathsf{mw}$. In Chapter 9, we give fixed-parameter algorithms for CSR parametrized by $k + \mathsf{td}$, $k + \mathsf{vc}$, and $k + \mathsf{nb}$, and give an ETH-based lower bound.

Finally, in Chapter 10, we conclude this thesis.

Chapter 2 Preliminaries

In this chapter, we formally present basic and standard terminologies and notations which will be used in this thesis. Definitions which are not presented in this chapter will be introduced as needed. We start, in Sections 2.1 through 2.2, by giving some definitions of standard graph-theoretical terminologies used throughout the remainder of this thesis. In Section 2.3, we next introduce some terminologies from algorithm theory. Finally, in Section 2.4, we then formally define the problems dealt with in this thesis and introduce some concepts.

2.1 Basic graph-theoretical terminologies

In this section, we introduce the most basic graph-theoretical terminologies.

2.1.1 Graphs and subgraphs

A *graph* G is an ordered pair (V, E) which consists of a finite set V of *vertices* (or *nodes*) and a finite set $E \subseteq V \times V$ of *edges*, where each edge in E is an pair of vertices in V. We sometimes denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. A graph G is *simple* if $E(G)$ contains no edge $e = vv$. A graph $G = (V, E)$ is *undirected* if $vw \in E$ if and only if $wv \in E$; and *directed* otherwise. For an undirected graph, we identify vw and wv. For a directed graph, we sometimes denote $vw = v \to w$. In this thesis, we consider a simple undirected graph unless otherwise stated. For example, Figure 2.1(a) illustrates a (simple undirected) graph G with the vertex set $V = \{v_1, v_2, \ldots, v_6\}$ and the edge set $E = \{e_1, e_2, \ldots, e_9\}$,

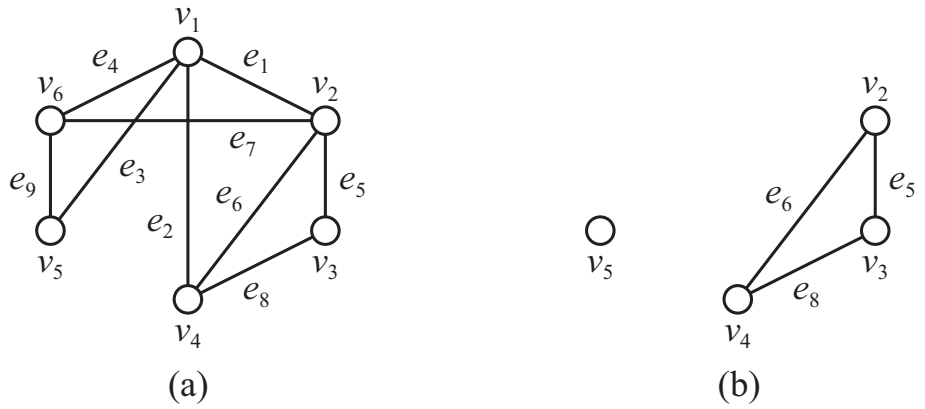


Figure 2.1: (a) A graph G with six vertices and nine edges, and (b) a subgraph G' of G induced by a vertex set $V' = \{v_2, v_3, v_4, v_5\}$.

where each vertex is drawn as a circle together with its index and each edge is drawn as a line joining two circles (vertices) together with its index. If $e = vw$ is an edge, then we say that e *joins* v and w; e is *incident* to v and w; v and w are *adjacent* (by the edge e;) and w is a *neighbor* of v. In the graph in Figure 2.1(a), for example, e_3 joins v_1 and v_5; e_7 is incident to v_2 and v_6; v_1 and v_2 are adjacent by e_1; and v_3 is a neighbor of v_4. The *neighborhood* $N(G, v)$ of a vertex v is a set of all neighbors of v, that is, $N(G, v) = \{w \in V(G) \mid vw \in E(G)\}$. For a vertex subset $V' \subseteq V(G)$, we denote $N(G, V') := \bigcup_{v \in V'} N(G, v) \setminus V'$. The *degree* $d(G, v)$ of a vertex v is the size (the cardinality) of the neighborhood $N(G, v)$ of v. In the graph in Figure 2.1(a), $N(G, v_6) = \{v_1, v_2, v_5\}$ and hence $d(G, v_6) = 3$.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$; we denote $G' \subseteq G$ and say that G *contains* G'. If E' is equal to the set of all the edges of G that join vertices in V', that is, $E' = \{vw \in E \mid v, w \in V' \wedge v \neq w\}$, then G' is called the *subgraph induced by* V' or simply called the *induced subgraph*; we denote $G' = G[V']$. Figure 2.1(b) depicts a subgraph G' of G in Figure 2.1(a) induced by a vertex set $V' = \{v_2, v_3, v_4, v_5\}$. For an induced subgraph $G' = (V', E')$ of G, we denote $G \setminus V' = G[V \setminus V']$.

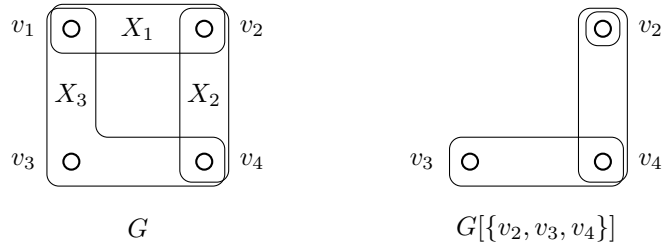


Figure 2.2: A hypergraph G and the subhypergraph $G[\{v_2, v_3, v_4\}]$ induced by $\{v_2, v_3, v_4\}$.

2.1.2 Hypergraphs, primal graphs and subhypergraphs

A *hypergraph* G is a pair (V, E), where V is a set of vertices and E is a family of non-empty vertex subsets, called *hyperedges*. We sometimes denote by $V(G)$ and $E(G)$ the vertex set and the hyperedge set of G, respectively. A hypergraph G is *r-uniform* if every hyperedge consists of exactly r (≥ 1) vertices. Any simple undirected graph can be considered as a 2-uniform hypergraph and a hyperedge of size exactly two is simply called an *edge*. An edge $\{v, w\}$ is sometimes denoted by vw or wv for notational convenience.

Let G be a hypergraph. The *primal graph* $\mathcal{P}(G)$ of G is a graph such that $V(\mathcal{P}(G)) = V(G)$ and two distinct vertices are connected by an edge if they are contained in the same hyperedge of G. We denote $N(G, v) := N(\mathcal{P}(G), v)$ for any vertex $v \in V$, and $N(G, V') := N(\mathcal{P}(G), V')$ for any vertex subset V'; we say that v is *adjacent to* w if $w \in N(G, v)$. For a vertex subset $V' \subseteq V(G)$, we define the *subhypergraph* of G *induced by* V' as the hypergraph G' such that $V(G') = V'$ and $E(G') = \{X \cap V' \colon X \in E(G), X \cap V' \neq \emptyset\}$. We denote by $G[V']$ the subhypergraph of G induced by V' for any vertex subset V'. We sometimes call a hypergraph G' an *induced subhypergraph* (*of* G) if $G[V'] = G'$ for some vertex subset $V' \subseteq V(G)$. (See Figure 2.2.) We use the notation $G \setminus V'$ to denote $G[V(G) \setminus V']$.

2.1.3 Paths, cycles and connectivities

Let $G = (V, E)$ be a graph. A *walk* from a vertex v to a vertex w is a sequence $\mathcal{W} = \langle v_0 = v, v_1, \ldots, v_\ell = w \rangle$ of vertices such that $v_{i-1}v_i \in E$ for every $i \in \{1, 2, \ldots, \ell\}$. The *length* of $\mathcal{W}$ is ℓ. If the vertices $v_1, v_2, \cdots, v_\ell$ are distinct, then the walk $\mathcal{W}$ is called a *path*. A path or walk is *closed* if $v_0 = v_\ell$. A closed path with length at least one is called a *cycle*. In the graph in Figure 2.1(a), $\langle v_1, v_2, v_6, v_1, v_4 \rangle$ is a walk but not a path, $\langle v_2, v_6, v_5, v_1 \rangle$ is a path but not a cycle, and $\langle v_2, v_3, v_4, v_2 \rangle$ is a cycle. The *distance* $\mathsf{dist}(v, w)$ between v and w is the length of a shortest path between v and w. In the graph in Figure 2.1(a), $\mathsf{dist}(v_3, v_5) = 3$ since $\langle v_3, v_4, v_1, v_5 \rangle$ is a shortest path between v_3 and v_5.

A graph G is *connected* if there is a path between v and w for any two distinct vertices v and w in G. A graph G is *disconnected* if it is not connected. The graph in Figure 2.1(a) is connected while the graph in Figure 2.1(b) is disconnected. Unless otherwise stated, we consider only simple connected undirected graphs. An induced subgraph G' of G is called a *connected component* if there is no connected subgraph G'' such that $G' \subseteq G''$. In Figure 2.1(b), the subgraphs induced by $\{v_2, v_3, v_4\}$ and $\{v_5\}$ are the connected components. Similarly, a hypergraph G is *connected* (resp., *disconnected*) if $\mathcal{P}(G)$ is connected (resp., disconnected).

2.1.4 Operations on graphs

In this section, we introduce some operations on graphs which construct a new graph. Let $G = (V, E)$ be a graph. For a vertex subset $X \subseteq V$ which induces a connected subgraph, the *contraction* of X is obtaining a graph $G' = (V', E')$ from G by deleting all edges of $G[X]$ and replacing all vertices in X with a new vertex x so that x is adjacent all neighbors of each vertex in X. Formally, $V' := (V \setminus X) \cup \{x\}$ and $E' := \{\phi(v)\phi(w) \mid vw \in E \setminus E(G[X])\}$, where ϕ is a mapping from V to V' such that $\phi(v) = x$ if $v \in X$, and $\phi(v) = v$ otherwise. The *complement* $\overline{G}$ of G is a graph

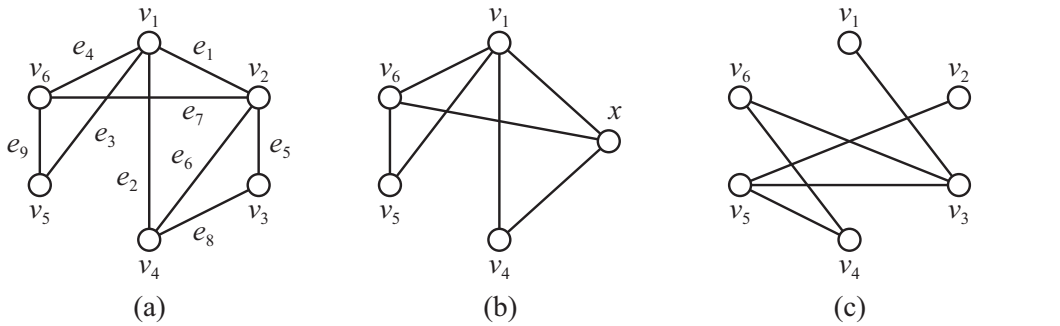


Figure 2.3: (a) A graph G, (b) the contraction of $\{v_2, v_3\} \subseteq V(G)$, and (c) the complement of G.

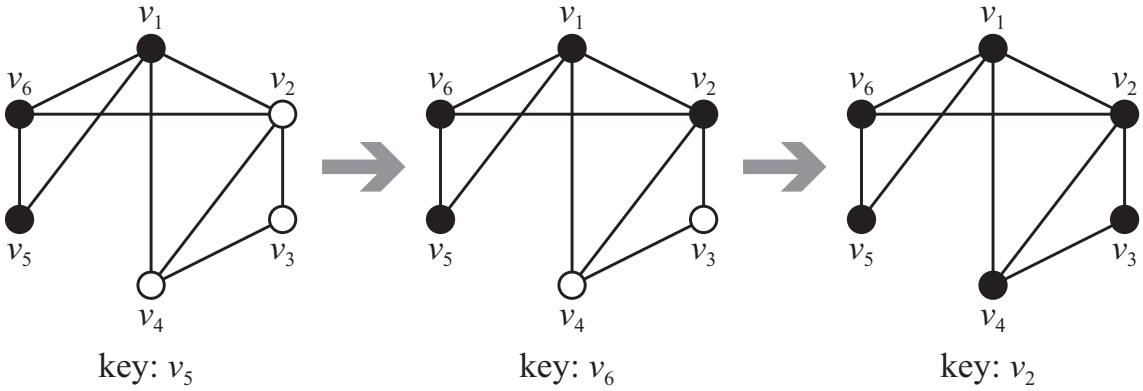


Figure 2.4: An example of the breadth-first search of th graph in Figure 2.1(a).

obtained from G by reversing the adjacencies between all pairs of distinct vertices. Formally, $\overline{G} = (V, \overline{E})$ such that $\overline{E} = \{vw \mid v, w \in V \wedge v \neq w\} \setminus E$.

2.1.5 Breadth-first search

The *breadth-first search* of a graph $G = (V, E)$ is an algorithm searching vertices of G as follows. It starts with some vertex $v \in V$ and lets v the "key" vertex. Then it recursively visits all unvisited neighbors of the key vertex, and selects an arbitrary vertex which has been visited and has a unvisited neighbor as a new key. This algorithm runs in time $O(|V| + |E|)$ [19]. Figure 2.1 shows an example of the breadth-first search starting with v_5, where the visited vertices are depicted as black circles in each step.

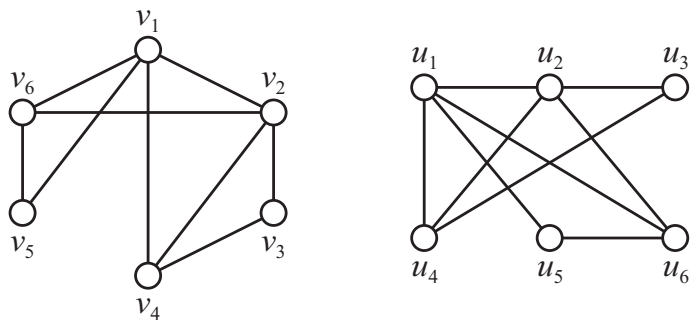


Figure 2.5: Two isomorphic graphs.

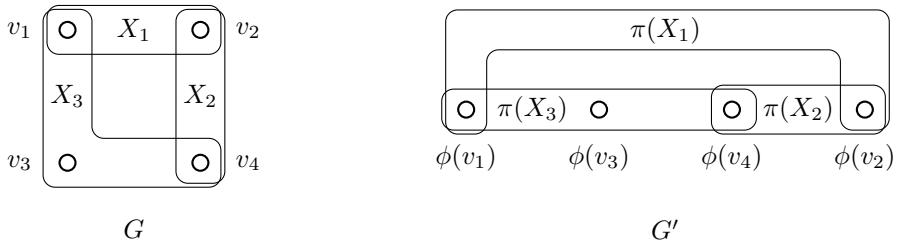


Figure 2.6: Two isomorphic hypergraphs G and G' under the bijections ϕ and π.

2.1.6 Graph isomorphism

Two graphs G_1 and G_2 are *isomorphic* if there exists a bijective function $\phi : V(G_1) \to V(G_2)$, that is, $vw \in E(G_1)$ if and only if $\phi(v)\phi(w) \in E(G_2)$. Figure 2.5 shows an example of two isomorphic graphs, where the bijective function maps v_i to u_i for each $i \in \{1, 2, 3, 4, 5, 6\}$.

Two hypergraphs G and G' are *isomorphic* if there exist two bijections $\phi\colon V(G) \to V(G')$ and $\pi\colon E(G) \to E(G')$ such that $\pi(X) = \{\phi(v_1), \phi(v_2), \ldots, \phi(v_r)\} \in E(G')$ holds for each hyperedge $X = \{v_1, v_2, \ldots, v_r\} \in E(G)$. (See Figure 2.6.) Notice that a bijection π between two hyperedge sets is uniquely determined by a bijection ϕ between two vertex sets.

2.1.7 Independent sets, vertex covers and cliques

Let $G = (V, E)$ be a graph. An *independent set* is a vertex subset V' of V such that every pair of two distinct vertices v and w in V' is not adjacent. A *vertex*

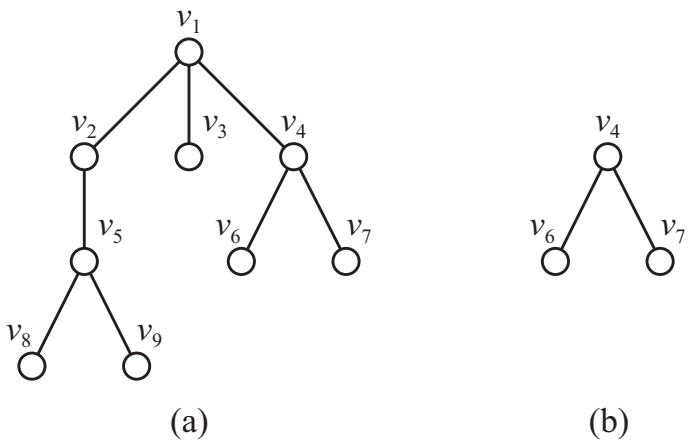


Figure 2.7: (a) A rooted tree with the root v_1, and (b) the rooted subtree with the root v_4.

cover is a vertex subset V' of V such that $V \setminus V'$ is an independent set. We denote by $\mathsf{vc}(G)$ the size of a minimum vertex cover of G. A *clique* is a vertex subset V' of V such that every pair of two distinct vertices v and w in V' is adjacent. We denote by $\omega(G)$ the maximum clique size of G. In the graph G in Figure 2.1(a), $\{v_3, v_5\}$ is a maximum independent set, $\{v_1, v_2, v_4, v_6\}$ is a minimum vertex cover, and $\{v_1, v_2, v_6\}$ is a maximum clique, and hence $\mathsf{vc}(G) = 4$ and $\omega(G) = 3$.

2.2 Graph paremeters and graph classes

2.2.1 Forests and trees

A *forest* is a graph which contains no cycle. A *tree* is a connected forest. A *path* is a tree which has at most two leaves.

A *rooted tree* T is a tree in which one of the vertices is distinguished from the others. The distinguished vertex is called the *root* of the tree. A rooted tree is usually drawn so that the root is located at the top, as illustrated in Figure 2.7(a) where v_1 is the root. For each edge $vw \in E(T)$ such that $\mathsf{dist}(r, v) + 1 = \mathsf{dist}(r, w)$, we say that v is the *parent* of w and w is a *child* of v. For example, in the rooted tree in Figure 2.7(a), v_4 is the parent of v_6 and v_7; and v_6 and v_7 are children of

v_4. A vertex v in a tree is called a *leaf* if it has no child. Otherwise, v is a called an *internal node*. For each vertex $v \in V(T)$, the *rooted subtree* T_v with the root v is a connected component of $T' = (V(T), E(T) \setminus \{e\})$ which contains v, where e joins v and v's parent. Figure 2.7(b) depicts the rooted subtree of the rooted tree in Figure 2.7(a) with the root v_4. A *full binary tree* T is a rooted tree such that for each vertex $v \in V(T)$, the number of children of v is either two or zero.

A *spanning tree* of a graph $G = (V, E)$ is a subgraph of G which is a tree and contains all vertices of G. Figure 2.8 shows a spanning tree of the graph in Figure 2.1(a).

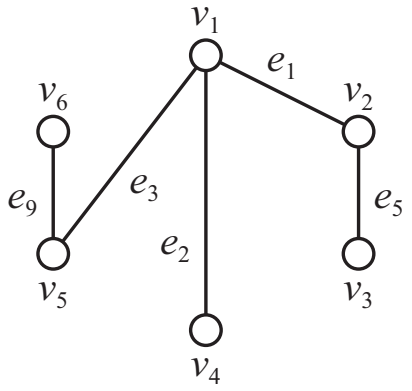


Figure 2.8: A spanning tree of the graph in Figure 2.1(a).

2.2.2 Pathwidth

We now define the notion of pathwidth [55]. A *path-decomposition* of a graph G is a sequence of subsets X_i, $1 \leq i \leq m$, of vertices in G such that

(1) $\bigcup_i X_i = V(G)$;

(2) for each $vw \in E(G)$, there is at least one subset X_i with $v, w \in X_i$; and

(3) for any three indices p, q, r such that $p \leq q \leq r$, $X_p \cap X_r \subseteq X_q$.

The *width* of a path-decomposition is defined as $\max_i |X_i| - 1$, and the *pathwidth* $\mathsf{pw}(G)$ of G is the minimum t such that G has a path-decomposition of width t. For example, Figure 2.9(b) illustrates a path-decomposition of a graph G in

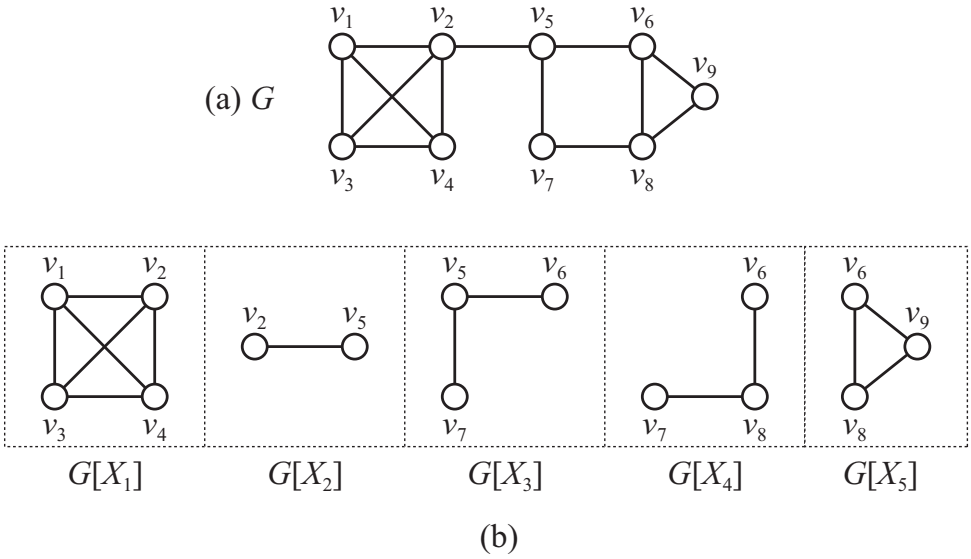


Figure 2.9: (a) A graph G, and (b) its path decomposition. In (b), each graph surrounded by dotted box is the graph induced each subset.

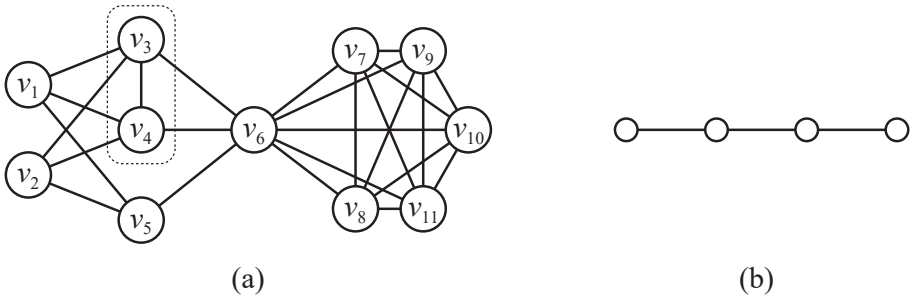


Figure 2.10: Examples of (a) a module and (b) a prime.

Figure 2.9(a); the width of this path-decomposition is three. Indeed, the path-decomposition in Figure 2.9(b) has minimum width, and hence the *pathwidth* of G is three.

2.2.3 Modules and modular-width

A *module* of a graph $G = (V, E)$ is a vertex subset $M \subseteq V$ such that $N(G, v) \setminus M = N(G, w) \setminus M$ for every two vertices v and w in M. In other words, the module M is a set of vertices whose neighborhoods in $G \setminus M$ are the same. For example, the graph in Figure 2.10(a) has a module $M = \{v_3, v_4\}$ for which $N(G, v_3) \setminus M =$

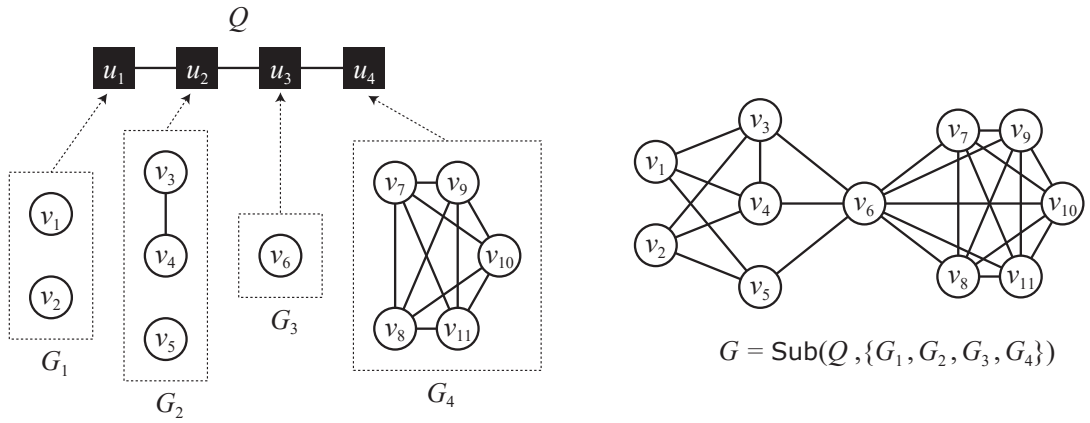


Figure 2.11: An example of substitution operation.

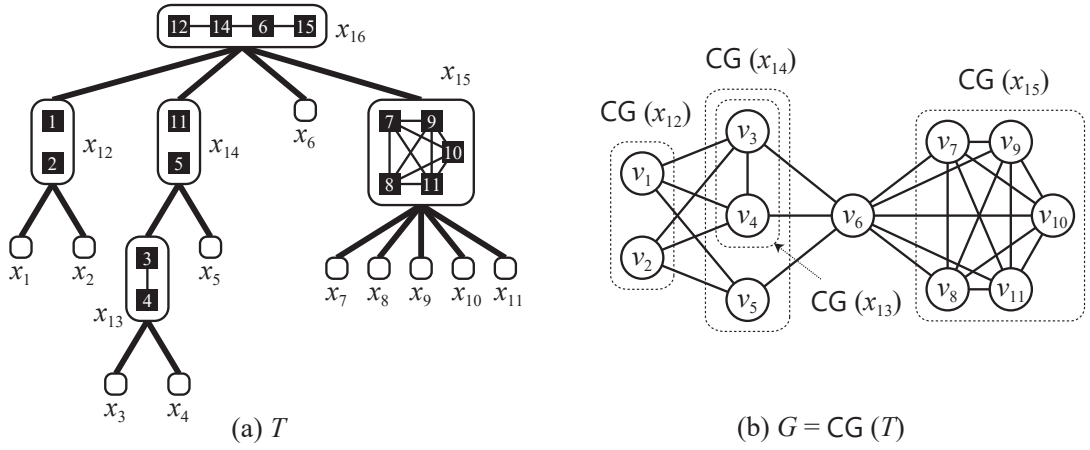


Figure 2.12: (a) A substitution tree T for (b) a graph G.

$N(G, v_4) \setminus M = \{v_1, v_2, v_6\}$ holds. Note that the vertex set V of G, the set consisting of only a single vertex, and the empty set $\emptyset$ are all modules of G; they are called *trivial*. A graph G is a *prime* if all of its modules are trivial; for an example, see Figure 2.10(b).

We now introduce the notion of modular decomposition, which was first presented by Gallai in 1967 as a graph decomposition technique [25]. For a survey, see, e.g., [29].

We first define the *substitution* operation, which constructs one graph from more than one graphs. Let Q be a graph, called a *quotient graph*, consisting of p (≥ 2) nodes $u_1, u_2, \ldots, u_p$, and let $\mathcal{F} = \{G_1, G_2, \ldots, G_p\}$ be a family of vertex-disjoint

graphs such that G_i corresponds to u_i for every $i \in \{1, 2, \ldots, p\}$. The *$Q$-substitution* of $\mathcal{F}$, denoted by $\mathsf{Sub}(Q, \mathcal{F})$, is the graph which is obtained by taking a union of all graphs in $\mathcal{F}$ and then connecting every pair of vertices $v \in V(G_i)$ and $w \in V(G_j)$ by an edge if and only if u_i and u_j are adjacent in Q. That is, the vertex set of $\mathsf{Sub}(Q, \mathcal{F})$ is $\bigcup\{V(G_i)\colon G_i \in \mathcal{F}\}$, and the edge set of $\mathsf{Sub}(Q, \mathcal{F})$ is the union of $\bigcup\{E(G_i)\colon G_i \in \mathcal{F}\}$ and $\{vw\colon v \in V(G_i), w \in V(G_j), u_iu_j \in E(Q)\}$. (See Figure 2.11 as an example.)

A *substitution tree* is a rooted tree T such that each non-leaf node $x \in V(T)$ is associated with a quotient graph $\mathsf{Q}(x)$ and has $|V(\mathsf{Q}(x))|$ child nodes; and each leaf node is associated with a single vertex. For each node $x \in V(T)$, we can recursively define the *corresponding graph* $\mathsf{CG}(x)$ as follows: If x is a leaf, $\mathsf{CG}(x)$ consists of the associated single vertex. Otherwise, let $y_1, y_2, \ldots, y_p$ be $p = |V(\mathsf{Q}(x))|$ children of x, then $\mathsf{CG}(x) = \mathsf{Sub}(\mathsf{Q}(x), \{\mathsf{CG}(y_1), \mathsf{CG}(y_2), \ldots, \mathsf{CG}(y_p)\})$. For the root r of T, $\mathsf{CG}(r)$ is called the *corresponding graph of* T, and we denote $\mathsf{CG}(T) := \mathsf{CG}(r)$. We say that T is a substitution tree *for* a graph G if $\mathsf{CG}(T) = G$, and refer to a *node* in T in order to distinguish it from a vertex in G. Figure 2.12(a) illustrates a substitution tree for the graph G in Figure 2.12(b); each leaf x_i, $i \in \{1, 2, \ldots, 11\}$, corresponds to the subgraph of G consisting of a single vertex v_i. We note that the vertex set $V(\mathsf{CG}(x))$ of each corresponding graph $\mathsf{CG}(x)$, $x \in V(T)$, forms a module of $\mathsf{CG}(T)$.

A *modular decomposition tree* T (an *MD-tree* for short) for a graph G is a substitution tree for G which satisfies the following three conditions:

- Each node $x \in V(T)$ applies to one of the following three types:
  - a *series* node, whose quotient graph $\mathsf{Q}(x)$ is a complete graph;
  - a *parallel* node, whose quotient graph $\mathsf{Q}(x)$ is an edge-less graph; and
  - a *prime* node, whose quotient graph $\mathsf{Q}(x)$ is a prime with at least four vertices.

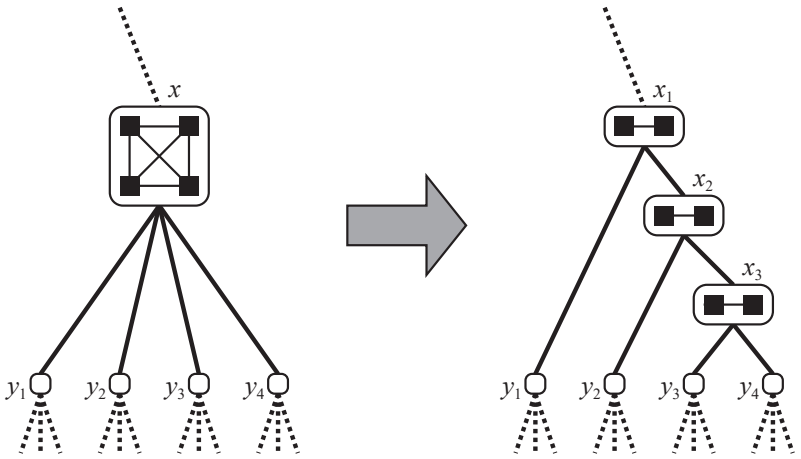


Figure 2.13: An example of the replacement described in the proof of Proposition 2.1.

- No edge in T connects two series nodes.
- No edge in T connects two parallel nodes.

It is known that any graph G has a unique MD-tree with $O(|V(G)|)$ nodes, and it can be computed in time $O(|V(G)|+|E(G)|)$ [58]. We denote by $\mathsf{MD}(G)$ the unique MD-tree for a graph G. The *modular-width* $\mathsf{mw}(G)$ of a graph G is the maximum number of children of a prime node in its MD-tree $\mathsf{MD}(G)$; we define $\mathsf{mw}(G) = 0$ if $\mathsf{MD}(G)$ has no prime node. The substitution tree T in Figure 2.12(a) is indeed the MD-tree for the graph G in Figure 2.12(b), and hence $\mathsf{mw}(G) = 4$; note that only x_{16} is a prime node in T.

We now define a variant of MD-trees, which will make our proofs and analyses simpler. A *pseudo modular decomposition tree* T (a *PMD-tree* for short) for a graph G is a substitution tree for G which satisfies the following two conditions:

- Each node $x \in V(T)$ applies to one of the following three types:
  - a 2-*join* node, whose quotient graph $\mathsf{Q}(x)$ is a complete graph with exactly two vertices;
  - a *parallel* node, whose quotient graph $\mathsf{Q}(x)$ is an edge-less graph; and

  - a *prime* node, whose quotient graph $\mathsf{Q}(x)$ is a prime with at least four vertices.

- No edge in T connects two parallel nodes.

Proposition 2.1 *For any graph G, there exists a PMD-tree T with $O(|V(G)|)$ nodes such that each prime node $x \in V(T)$ has at most $\mathsf{mw}(G)$ children, and it can be constructed in linear time.*

Proof. Recall that an MD-tree $\mathsf{MD}(G)$ for a graph G can be constructed in linear time. Given an MD-tree $\mathsf{MD}(G)$ for a graph G, we construct a PMD-tree T such that $\mathsf{CG}(T) = \mathsf{CG}(\mathsf{MD}(G))$ as follows. For each series node x of $\mathsf{MD}(G)$ having m (≥ 3) children $y_1, y_2, \ldots, y_m$, we replace it with a binary tree consisting of $2m-1$ nodes $x_1, x_2, \ldots, x_{m-1}$ and $y_1, y_2, \ldots, y_m$ such that x_i has two children y_i and x_{i+1} for each $i \in \{1, 2, \ldots, m-2\}$ and x_{m-1} has two children y_{m-1} and y_m. (See Figure 2.13 as an example.) A quotient graph $\mathsf{Q}(x_i)$ of each new node x_i is defined as a complete graph with exactly two vertices. Then, T is a PMD-tree for G, it has at most $O(|V(G)|)$ nodes, and each prime node $x \in V(T)$ has at most $\mathsf{mw}(G)$ children. Moreover, this process can be done in linear time since the MD-tree has $O(|V(G)|)$ nodes. □

We denote by $\mathsf{PMD}(G)$ a PMD-tree for G such that each prime node $x \in V(T)$ has at most $\mathsf{mw}(G)$ children. The *pseudo modular-width* $\mathsf{pmw}(G)$ of a graph G is the maximum number of children of a non-parallel node in its PMD-tree; we define $\mathsf{pmw}(G) = 0$ if $\mathsf{MD}(G)$ has only a parallel node, that is, G has no edge.[1]

A graph G is a *cograph* if $\mathsf{mw}(G) = 0$.

2.2.4 Other graph classes

Let $G = (V, E)$ be a graph. G is *bipartite* if the vertex set V can be partitioned into two independent sets. G is *split* if the vertex set V can be partitioned into a

[1]The pseudo modular-width itself is sometimes regarded as the modular-width of a graph.

clique and an independent set. G is *threshold* if there exist a real number s and a mapping $\omega : V \to \mathbb{R}$ such that $xy \in E$ if and only if $\omega(x) + \omega(y) \geq s$ [9], where $\mathbb{R}$ is the set of all real numbers. Equivalently, G is threshold if G is a cograph and a split graph. G is *planar* if it can be drawn in a plane without crossing edges. G is *complete* if $\omega(G) = |V|$. We denote by K_n the complete graph with n vertices.

2.3 Algorithm-theoretical terminologies

In this section, we introduce several algorithm-theoretical terminologies.

2.3.1 Problems and reductions

A *problem* is a language (i.e., a set of strings) $\mathcal{P} \subseteq \Sigma^*$, where $\Sigma = \{0, 1\}$. Each string x in Σ^* is called an *instance*. An instance x is a *yes-instance* of a problem $\mathcal{P}$ if $x \in \mathcal{P}$, and x is a *no-instance* of $\mathcal{P}$ otherwise.

Let $\mathcal{P}, \mathcal{P}' \subseteq \Sigma^*$ be two problems. A *reduction* from $\mathcal{P}'$ to $\mathcal{P}$ is an algorithm which compute an instance $x \in \Sigma^*$ for any given instance $y \in \Sigma^*$ such that $x \in \mathcal{P}$ if and only if $y \in \mathcal{P}'$. If a reduction runs in polynomial time, it is called a *polynoimal-time reduction*.

2.3.2 PSPACE

PSPACE is the class of all problems which can be solved in polynomial-space. A problem $\mathcal{P}$ is *PSPACE-hard* if for every problem $\mathcal{P}'$ in PSPACE, there exists a polynomial-time reduction from $\mathcal{P}'$ to $\mathcal{P}$. A problem $\mathcal{P}$ is *PSPACE-complete* if it is in PSPACE and is PSPACE-hard.

2.3.3 Parameterized complexity

In this section, we introduce some terminologies in parameterized complexity theory [22, 49].

A *parameterized problem* is a language $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma = \{0, 1\}$ and $\mathbb{N}$ is the set of all natural numbers. Each string (x, p) in $\Sigma^* \times \mathbb{N}$ is called an *instance*, and the second component p is called the *parameter* of the problem. An instance (x, p) is a *yes-instance* of a problem $\mathcal{P}$ if $(x, p) \in \mathcal{P}$, and (x, p) is a *no-instance* of $\mathcal{P}$ otherwise.

Let $\mathcal{P}$ be a parameterized problem. A *fixed-parameter algorithm* for $\mathcal{P}$ is an algorithm which determines whether $(x, p) \in \mathcal{P}$ or not in time $O(g(p) \cdot |(x, p)|^c)$ for any instance (x, p), where g is some computable function depending only on the parameter p, and c is some constant. A parameterized problem $\mathcal{P}$ is *fixed-parameter tractable* if there exists a fixed-parameter algorithm for $\mathcal{P}$. FPT is the class of all fixed-parameter tractable parametrized problems. Sometimes, an algorithm running in time $O(g(p)\cdot|(x, p)|^c)$ for any instance $(x, p) \in \mathcal{P}$ is called an *FPT-time* algorithm.

Let $\mathcal{P}, \mathcal{P}' \subseteq \Sigma^* \times \mathbb{N}$ be two parametrized problems. A *parametrized reduction* from $\mathcal{P}'$ to $\mathcal{P}$ is an FPT-time algorithm which computes an instance $(x, p) \in \Sigma^* \times \mathbb{N}$ for any given instance $(y, q) \in \Sigma^* \times \mathbb{N}$ such that $q \leq h(p)$ for some computable function h, and $x \in \mathcal{P}$ if and only if $y \in \mathcal{P}'$. $W[1]$ is the class of all parameterized problems from which there is a parametrized reduction to the weighted 3-SAT problem. A parameterized problem $\mathcal{P}$ is $W[1]$-*hard* if for every parameterized problem $\mathcal{P}'$ in $W[1]$, there exists a polynomial-time reduction from $\mathcal{P}'$ to $\mathcal{P}$.

Finally, we introduce the notion of *kernelization*, which is a technique for developing a fixed-parameter algorithm.

Let $\mathcal{P}$ be a parameterized problem. A *kernelization* for $\mathcal{P}$ is an algorithm which runs in time polynomial in the size of the input instance (x, p) and compute an instance (x', p') such that

- $p' \leq p$;
- $|x| \leq h(p)$ for some computable function h depending only on the parameter p; and

- $(x, p) \in \mathcal{P}$ if and only if $(x', p') \in \mathcal{P}$.

The instance (x', p') computed by this algorithm is called the *kernel* of (x, p). Then we have the following proposition.

Proposition 2.2 *Let $\mathcal{P}$ be a parameterized problem. Assume that $\mathcal{P}$ can be solved in $T(|x|, p)$ for any instance $(x, p) \in \mathcal{P}$ and there exists a kernelization for $\mathcal{P}$. Then, there exists a fixed-parameter algorithm for $\mathcal{P}$.*

Proof. Consider the following algorithm: we first run a kernelization for an instance (x, p) and then solve a kernel (x', p') in time $T(|x'|, p')$. This is indeed a fixed-parameter algorithm for $\mathcal{P}$ because $|x'|$ and p' depend only on p. □

In this thesis, we allow a kernelization to run in FPT time, and call such a relaxed kernelization a *kernelization*. Notice that Proposition 2.2 holds even for this relaxed kernelization.

2.4 Problems dealt with in this thesis

In this section, we give a formal definition of the problems dealt with in this thesis.

2.4.1 Mappings

Since all of our problems are based on mappings, we first introduce several concepts regarding mappings. Let A and B be any sets. We denote by B^A the set of all mappings from A to B, because we can identify a mapping $\phi\colon A \to B$ with a vector $(\phi(a_1), \phi(a_2), \ldots, \phi(a_{|A|})) \in B^{|A|}$, where $A = \{a_1, a_2, \ldots, a_{|A|}\}$. Let $\phi, \phi' \in B^A$ be two mappings. We define the *difference* $\mathsf{dif}(f, f')$ *between* f and f' as the set $\{a \in A \mid f(a) \neq f'(a)\}$. For any subset A' of A, we denote by $\phi|_{A'}$ the restriction of ϕ on A'; that is, $\phi|_{A'}$ is a mapping in $B^{A'}$ such that $\phi|_{A'}(a) = \phi(a)$ for any $a \in A'$. Let $\phi \in B^A$ and $\phi' \in B^{A'}$ be two mappings from the different sets A and A' Then, ϕ and ϕ' are *compatible* if $\phi|_{A \cap A''} = \phi'|_{A \cap A''}$ holds.

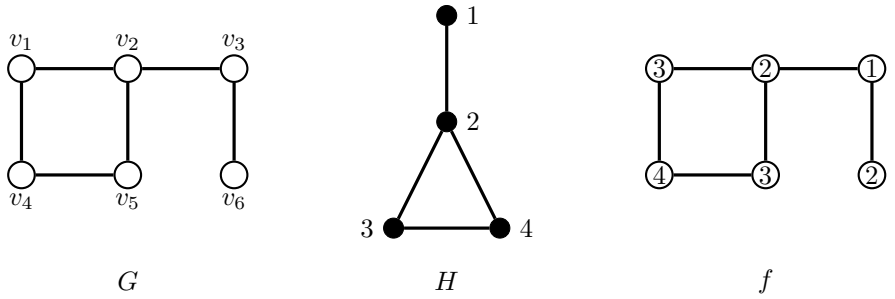


Figure 2.14: A graph G, a graph H whose vertex set is $\{1, 2, 3, 4\}$, and a homomorphism from f from G to H.

2.4.2 Graph colorings and homomorphisms

Let $G = (V, E)$ be a graph and let C be a set of k colors. Assume that each vertex $v \in V$ has a *list* $L(v) \subseteq C$ (*of* v); we sometimes call a mapping $L\colon V \to 2^C$ itself a *list*. A k-coloring $f\colon V \to C$ of G is called an *L-coloring* (or a *list coloring*) if $f(v) \in L(v)$ holds for every vertex $v \in V$. We note that an L-coloring is also a k-coloring if $L(v) = C$ holds for every vertex $v \in V$. Let H be a graph whose vertex set is C. A mapping $f\colon V \to C$ of G is called a *homomorphism* from G to H if $f(v)f(w) \in E(H)$ holds for every edge $vw \in E$. Moreover, f is also called an *L-homomorphism* (or a *list homomorphism*) from G to H if $f(v) \in L(v)$ holds for every vertex $v \in V$. A graph H is called an *underlying graph*. Notice that a (list) homomorphism is equivalent to a (list) coloring if an underlying graph is a complete graph. As we mentioned in the footnote 1 in the previous chapter, we only consider in this thesis simple undirected underlying graphs unless otherwise stated.

Let G, H and L be a graph, a simple underlying graph and a list $L\colon V(G) \to 2^{V(H)}$, respectively We then have the following observation.

Observation 2.1 *If there exists an L-homomorphism from G to H, then $\omega(G) \leq |V(H)|$ holds.*

2.4.3 Constraint satisfiability reconfiguration

In this subsection, we formally define CSP and its reconfiguration variant by means of hypergraphs.

Let $G = (V, E)$ be a hypergraph. Let D be a set, called a *domain*; each element of D is called a *value* and we always denote by k the size (cardinality) of a domain. In CSP, each hyperedge $X \in E$ has a set $\mathcal{C}(X) \subseteq D^X$; we call $\mathcal{C}(X)$ a *constraint of* X. If $\mathcal{C}(X) = D^X$, it is called a *trivial* constraint. An *arity* of a constraint $\mathcal{C}(X)$ of X is exactly $|X|$, and we call $\mathcal{C}(X)$ an *r-ary* constraint, where $r = |X|$. We define the *constraint $\mathcal{C}(G)$ of G* as the union of all constraints of hyperedges, that is, $\mathcal{C}(G) = \bigcup_{X \in E(G)} \mathcal{C}(X)$. For a vertex $v \in V(G)$, a *list* $L(v)$ of v is the set $\{i \in D \colon \exists g \in \mathcal{C}(G),\ g(v) = i\}$. For a hyperedge $X \in E$, we say that a mapping $f \in D^V$ *satisfies* a constraint of X if $f|_X \in \mathcal{C}(X)$ holds. f is a *solution* if it satisfies all constraints.

An instance of CONSTRAINT SATISFIABILITY is a triple $(G, D, \mathcal{C})$ consisting of a hypergraph G, a domain D, and a constraint assignment $\mathcal{C}$ to hyperedges over D. Then, the problem asks whether there exists a solution or not. CONSTRAINT SATISFIABILITY includes many combinatorial problems as its special cases as follows.

- BOOLEAN CONSTRAINT SATISFIABILITY is the special case of CONSTRAINT SATISFIABILITY where $|D| = 2$.
- For an integer $r \geq 1$, r-ARY CONSTRAINT SATISFIABILITY is the special case of CONSTRAINT SATISFIABILITY where all constraints are of arity at most r, that is, all hyperedges have size at most r.
- LIST HOMOMORPHISM is the special case of 2-ARY CONSTRAINT SATISFIABILITY where there exists a simple underlying graph H such that the solution set is exactly the set of all L-homomorphisms from G to H. That is, G is a 2-uniform hypergraph (i.e., a graph) and $\mathcal{C}(vw) = E(H) \cap (L(v) \times L(w))$

holds for every edge $vw \in E(G)$; in other words, $\mathcal{C}(vw)$ is the set of all list homomorphisms from the edge vw to H.

- HOMOMORPHISM is the special case of LIST HOMOMORPHISM where $L(v) = D$ holds for every vertex $v \in V(G)$.

- (LIST) COLORING is the special case of (LIST) HOMOMORPHISM where an underlying graph H is complete, that is, $\mathcal{C}(vw)$ is a set of all injective mappings from $\{v, w\}$ to D (which respect the lists of v and w).

We here note that there exist at least two ways of expressing a constraint $\mathcal{C}$ of each hyperedge $X \in E(G)$. The first is assuming an oracle which determines whether a given mapping $f\colon X \to D$ satisfies $\mathcal{C}(X)$ or not. The second is explicitly giving a set of all mappings from X to D which satisfies $\mathcal{C}(X)$. However, we can transform an instance of the first form into one of the second form in time $O^*(k^r)$, where r be the maximum arity of constraints. This transformation can be done in FPT time with respect to $k+r$ and even in polynomial time if r is a constant. Because we will only consider problems in which r is a constant or $k + r$ is bounded by the parameter, we assume that an instance is of the second form in this thesis.

We next define a reconfiguration variant of CONSTRAINT SATISFIABILITY, that is, CONSTRAINT SATISFIABILITY RECONFIGURATION. Let $\mathcal{I} = (G, D, \mathcal{C})$ is an instance of CSR. We now define the *solution graph* $\mathscr{S}(\mathcal{I})$ for $\mathcal{I}$ as follows. (See Figure 2.15.) $V(\mathscr{S}(\mathcal{I}))$ is the set of all solutions for $\mathcal{I}$, and two solutions f and f' are connected by an edge if and only if $|\mathsf{dif}(f, f')| = 1$. A walk in $\mathscr{S}(\mathcal{I})$ is called a *reconfiguration sequence*. Two solutions f and f' are *reconfigurable* if and only if there exists a reconfiguration sequence between them.

An instance of CONSTRAINT SATISFIABILITY RECONFIGURATION (CSR for short) is a 5-tuple $(G, D, \mathcal{C}, f_s, f_t)$, where $(G, D, \mathcal{C})$ is an instance of CONSTRAINT SATISFIABILITY, and f_s and f_t are two solutions to $(G, D, \mathcal{C})$, called *initial* and *target*

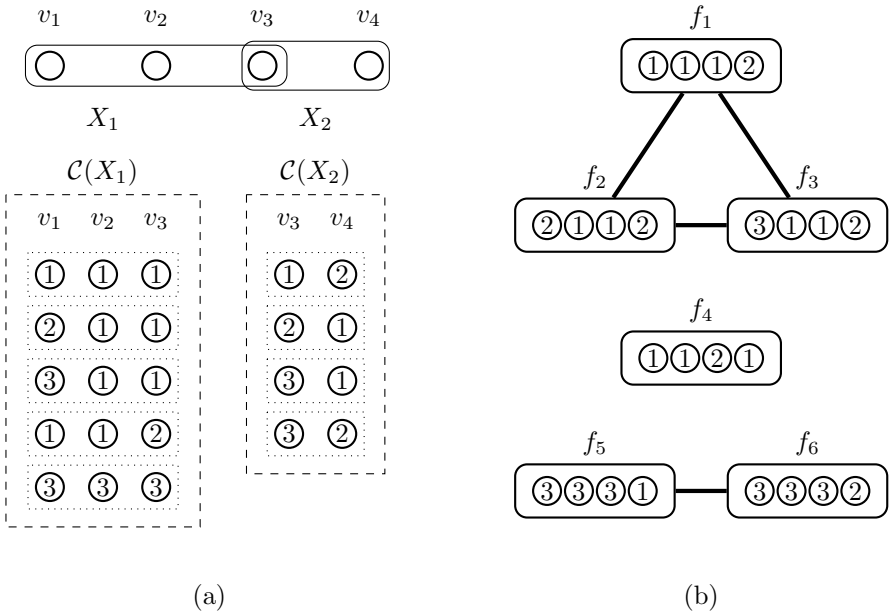


Figure 2.15: (a) An instance $\mathcal{I} = (G, \{1, 2, 3\}, \mathcal{C})$ of CONSTRAINT SATISFIABILITY, and (b) the solution graph $\mathscr{S}(\mathcal{I})$.

solutions, respectively. Then, the problem asks whether f_s and f_t are reconfigurable or not. Similarly, for each special case P of CONSTRAINT SATISFIABILITY, we define "P RECONFIGURATION" as a special case of CSR where $(G, D, \mathcal{C})$ forms an instance of P. (Recall the abbreviations for the problems introduced in the previous chapter and see Figure 1.4(a) for relationships between them.)

2.4.4 Other definitions and observations

Let $(G, D, \mathcal{C})$ be an instance of CONSTRAINT SATISFIABILITY. A *Boolean vertex* is a vertex $v \in V(G)$ with $|L(v)| \leq 2$, and a *non-Boolean vertex* is a vertex $v \in V(G)$ with $|L(v)| > 2$. Let X and X' be hyperedges in $E(G)$ such that $|X| = |X'|$. For a bijection $\phi\colon X \to X'$, we denote by $\mathcal{C}[\phi](X)$ the set $\{g \circ \phi^{-1}\colon g \in \mathcal{C}(X)\} \subseteq D^{X'}$ of mappings from X' to D, where $\circ$ means the composition of mappings. Intuitively, $\mathcal{C}[\phi](X)$ is a "translation" of $\mathcal{C}(X)$ into a constraint of X' via a bijection ϕ. For example, assume that $\mathcal{C}(\{v_1, v_2, v_3\})$ contains a mapping $g = (1, 2, 3)$. If a bijection $\phi\colon \{v_1, v_2, v_3\} \to \{u_1, u_2, u_3\}$ maps v_1, v_2, v_3 to u_2, u_1, u_3, respectively,

then $\mathcal{C}[\phi](\{v_1, v_2, v_3\})$ contains a mapping $g' \colon \{u_1, u_2, u_3\} \to \{1, 2, 3\}$ such that $(g'(u_1), g'(u_2), g'(u_3)) = (g \circ \phi^{-1}(u_1), g \circ \phi^{-1}(u_2), g \circ \phi^{-1}(u_3)) = (g(v_2), g(v_1), g(v_3)) = (2, 1, 3)$.

Let $(G, D, \mathcal{C})$ be an instance of 2-CONSTRAINT SATISFIABILITY. Without loss of generality, we assume that G is connected, $|V(G)| \geq 2$, and $D = \{0, 1\}$. Moreover, we can assume that G is 2-uniform as follows. If G contains a size-one hyperedge $\{v\}$, there must exist a size-two hyperedge (i.e., an edge) $vw \in E(G)$ from the assumption. Then, we remove $\{v\}$ from $E(G)$ and replace $\mathcal{C}(vw)$ with the set of all solutions satisfying $\mathcal{C}(\{v\})$ and $\mathcal{C}(vw)$. Note that this modification does not change the set of solutions and the primal graph.

We finally see that there exists the following straightforward exact algorithm.

Theorem 2.1 CSR *can be solved in time* $O^*(k^{O(n)})$*, and hence* CSR *is fixed-parameter tractable when parameterized by* $k + n$ *and in XP when parameterized by* n*, where* n *is the number of vertices of a given hypergraph.*

Proof. Our algorithm explicitly construct the solution graph and then check the connectivity between two given solutions. The solution graph has at most k^n vertices and can be constructed in time $O^*(k^{O(n)})$. The connectivity can be checked in time polynomial in the size of the solution graph by a simple breadth-first search. Therefore, our algorithm runs in time $O^*(k^{O(n)})$. □

Part I

Polynomial-Time Solvability

Chapter 3 Coloring Reconfiguration

In this chapter, we study the complexity of CR from the viewpoint of graph classes.

3.1 Defenitions and observations

We first introduce another expression of instances of (L)CR. Notice that an instance of COLORING (resp., LIST COLORING) can be uniquely determined by a graph, and the number of colors (resp., a list). Therefore, we sometimes express an instance of CR by a 4-tuple (G, k, f_s, f_t), consisting of a graph G, the number k of colors, and two k-colorings f_s and f_t. Similarly, we sometimes express an instance of LCR by a 4-tuple (G, L, f_s, f_t), consisting of a graph G, a list L, and two L-colorings f_s and f_t. Furthermore, for an integer $c \geq 1$, we define c-CR as a special case of CR where $k \leq c$. We express an instance of c-CR by a triple (G, f_s, f_t).

We next define the notion of "frozen vertices" [4, 60]. Let f be an L-coloring of a graph G with a list L. A vertex $v \in V(G)$ is said to be *frozen on* f if $f'(v) = f(v)$ holds for every L-coloring f' of G which is reconfigurable from f. Therefore, v cannot be recolored in any reconfiguration sequence. Thus, (G, L, f_s, f_t) is a no-instance of LCR if $f_s(v) \neq f_t(v)$ holds for at least one frozen vertex v on f_s or f_t. By the definition, a frozen vertex v on an L-coloring f stays frozen on any L-coloring which is reconfigurable from f. Generally speaking, it is not easy to characterize such frozen vertices for a given L-coloring. However, there is a simple sufficient condition for which a vertex is frozen, as follows. (See Figure 3.1 as an example of Observation 3.1.)

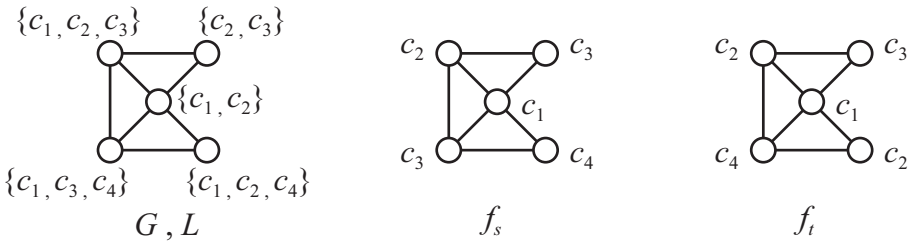


Figure 3.1: Example for frozen vertices: The upper three vertices are frozen on f_s and f_t because they form a clique of size three, and their lists contain only three colors in total.

Observation 3.1 *Let G be a graph with a list L, and assume that G contains a clique V_Q of size q. If $|\bigcup_{v \in V_Q} L(v)| = q$, then all vertices $v \in V_Q$ are frozen on any L-coloring of G.*

3.2 PSPACE-completeness on chordal graphs

A graph is *chordal* if it contains no induced cycle of length at least four. In this section, we prove the following theorem.

Theorem 3.1 *There exists a fixed constant k' such that k-*CR *is PSPACE-complete for chordal graphs and every $k \geq k'$.*

It is known that k-CR belongs to PSPACE [4]. Therefore, as a proof of Theorem 3.1, we show that there exists a fixed constant k' such that k-CR is PSPACE-hard for chordal graphs and any $k \geq k'$, by giving a polynomial-time reduction from LCR [60].

3.2.1 First step of the reduction

Wrochna [60, Theorem 4.3] proved that there exist two constants a and b such that LCR remains PSPACE-complete even when an input instance (H, L, g_s, g_t) satisfies the following four conditions (a)–(d) (see also Fig. 3.2):

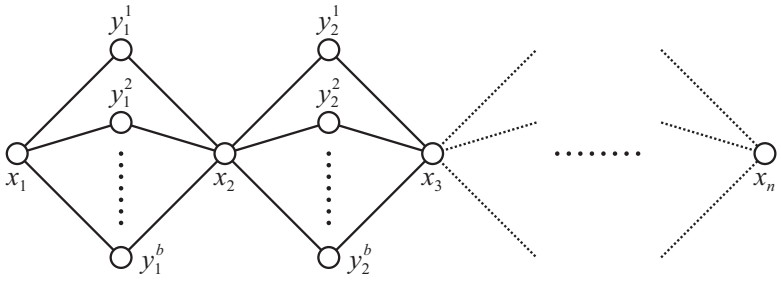


Figure 3.2: Graph H.

(a) $H = (X \cup Y, E)$ is a bipartite graph whose bipartition is X and Y such that

- $X = \{x_1, x_2, \ldots, x_{|X|}\}$;
- $Y = \{y_i^j : 1 \leq i \leq |X| - 1,\ 1 \leq j \leq a\}$; and
- $E = \{x_i y_i^j,\ y_i^j x_{i+1} : 1 \leq i \leq |X| - 1,\ 1 \leq j \leq a\}$;

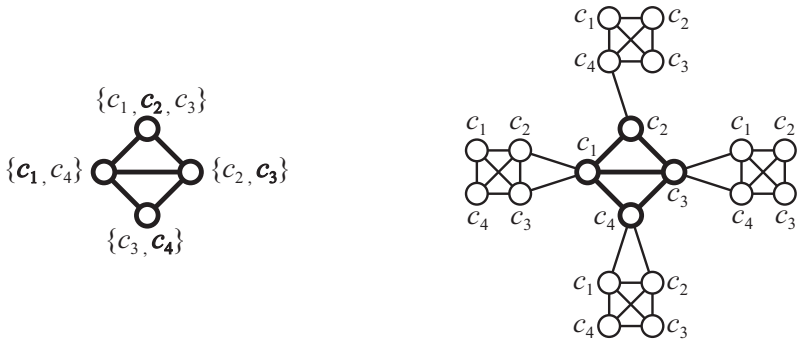
(b) the list $L(v)$ of each vertex $v \in V(H)$ is a subset of the color set $C_1 \cup C_2$ such that $C_1 \cap C_2 = \emptyset$ and $|C_1| = |C_2| = b$;

(c) $L(x_i) = C_1$ if i is odd, $L(x_i) = C_2$ otherwise; and

(d) $L(y) \subseteq C_1 \cup C_2$ for all $y \in Y$.

The graph H above can be modified to an interval graph (and hence a chordal graph) H' by adding an edge $x_i x_{i+1}$ for each $i \in \{1, 2, \ldots, |X| - 1\}$. This modification does not affect the existence and the reconfigurability of L-colorings, because any two vertices x_i and x_{i+1} joined by the new edge have disjoint lists C_1 and C_2. We note in passing that this modification gives the following theorem. For an integer $d \geq 0$, a graph G is *d-degenerate* if every subgraph G' of G has at least one vertex v such that $\mathsf{deg}(G', v) \leq d$. Note that the interval graph H' is 2-degenerate, because each vertex in Y is of degree two in H'.

Theorem 3.2 LCR *is PSPACE-complete for* 2*-degenerate interval graphs.*



(a) H', L, g_s (b) G, f_s

Figure 3.3: (a) A graph H', a list L and an L-coloring g_s, and (b) a constructed graph G and k-coloring f_s.

3.2.2 Reduction

We then construct an instance (G, f_s, f_t) of k-CR from the instance (H', L, g_s, g_t) above of LCR, as follows.

Let $k' = |C_1 \cup C_2| = |\bigcup_{u \in V(H')} L(u)| = 2b$ and let k be any integer at least k'. For each vertex $u \in V(H')$, we introduce a complete graph W_u of k vertices, which is called a *frozen clique gadget*. (See Figure 7.1 as an example, where $k = 4$.) The vertices in W_u are labeled as $w_1^u, w_2^u, \ldots, w_k^u$, and each vertex w_i^u corresponds to the color c_i for each $i \in \{1, 2, \ldots, k\}$. We denote by W the set of all vertices in frozen clique gadgets, that is, $W = \bigcup_{u \in V(H')} V(W_u)$.

We next add an edge between $u \in V(H')$ and $w_i^u \in V(W_u)$ if $L(u)$ does *not* contain color c_i. The constructed graph G is chordal, because the addition of frozen clique gadgets does not produce any induced cycle of length at least four.

Finally, we define f_s and f_t, as follows:

$$f_s(v) = \begin{cases} c_i & \text{if } v = w_i^u \in V(W_u) \text{ for some } u \in V(H'); \\ g_s(v) & \text{otherwise,} \end{cases}$$

and

$$f_t(v) = \begin{cases} c_i & \text{if } v = w_i^u \in V(W_u) \text{ for some } u \in V(H'); \\ g_t(v) & \text{otherwise.} \end{cases}$$

Therefore, we have $f_s(v) = f_t(v)$ for all vertices $v \in W$. From the construction, we note that both f_s and f_t are proper k-colorings of G.

This completes our construction of the corresponding instance (G, f_s, f_t) of k-CR. This construction can be done in polynomial time.

3.2.3 Correctness of the reduction

We note that all vertices in W are frozen on both f_s and f_t, because each frozen clique gadget W_u is a clique in G of size $|V(W_u)| = k$. Therefore, we can recolor vertices only in $V(H') = V(G) \setminus W$. In addition, we can use colors only in $L(u)$ for each vertex $u \in V(H')$; recall the construction with noting that $f_s(v) = f_t(v)$ for all vertices $v \in W$. Thus, (H', L, g_s, g_t) is a yes-instance of LCR if and only if the corresponding instance (G, f_s, f_t) of k-CR is a yes-instance.

This completes our proof of Theorem 3.1.

3.3 Polynomial-time solvable cases

In this section, we demonstrate that CR can be solved in polynomial time for some graph classes, even when the number k of colors is a part of input.

We start with noting the polynomial-time solvability for 2-degenerate graphs, which can be obtained straightforwardly by combining two known results. The class of 2-degenerate graphs properly contains graphs with treewidth at most two, and hence trees, cacti, outerplanar graphs, and series-parallel graphs. We note that one can check in linear time if a given graph is 2-degenerate [45].

Theorem 3.3 CR *can be solved in $O(n^2)$ time for 2-degenerate graphs, where n is the number of vertices in an input graph.*

Proof. Let (G, k, f_s, f_t) be an instance for CR. Bonsma and Cereceda [4, Theorem 11] proved that it is a yes-instance if G is d-degenerate and $k \geq d+2$. Therefore,

for 2-degenerate graphs, the answer is always yes if $k \geq 2 + 2 = 4$. On the other hand, Cereceda et al. [17, Theorem 1] gave an $O(nm)$-time algorithm to solve CR for any graph G if $k \leq 3$, where n and m are the numbers of vertices and edges in G, respectively. For a 2-degenerate graph G, we have $m < 2n$ and hence the theorem follows. □

In contrast to the polynomial-time solvability for 2-degenerate graphs even when k is a part of input, the reduction given by Bonsma and Cereceda [4, Theorem 3] indeed shows the following theorem.

Theorem 3.4 ([4]) 4-CR *is PSPACE-complete for* 3*-degenerate planar graphs.*

3.3.1 q-trees

In Subsections 3.3.1–3.3.3, we consider subclasses of chordal graphs such as q-trees with any integer $q \geq 1$[1], split graphs, and trivially perfect graphs. The following sufficient condition for yes-instances on chordal graphs will play an important role in those subsections.

Lemma 3.1 ([2, Theorems 2 and 6]) *Let* (G, k, f_s, f_t) *be an instance of* CR *such that* G *is a chordal graph. If* $\omega(G) \leq k - 1$*, then it is a yes-instance. Furthermore, there is a reconfiguration sequence between* f_s *and* f_t *whose length is at most* $2|V(G)|^2$*.*

In this subsection, we consider q-trees. For an integer $q \geq 1$, a *q-tree* is recursively defined, as follows:

(1) a complete graph consisting of q vertices is a q-tree; and

(2) if G' is a q-tree and $Q \subseteq V(G')$ is a clique of size q, then the graph G such that $V(G) = V(G') \cup \{v\}$ and $E(G) = E(G') \cup \{vw : w \in Q\}$ is a q-tree, where v is a new vertex.

[1]We note that q-trees are usually called k-trees, but we denote by k the number of colors in this thesis:.

Notice that a q-tree has at least q vertices. From the definition, we have the following proposition.

Proposition 3.1 *Let G be a q-tree. Then,*

(i) *$\omega(G) = q$ if $|V(G)| = q$, and $\omega(G) = q + 1$ otherwise; and*

(ii) *every vertex of G is contained in a maximum clique of G.*

We are now ready to give our result, as follows.

Theorem 3.5 *For any integer $q \geq 1$,* CR *can be solved in linear time for q-trees. Furthermore, if (G, k, f_s, f_t) is a yes-instance, there is a reconfiguration sequence whose length is at most $2|V(G)|^2$.*

Proof. We give such a linear-time algorithm for q-trees. Let $\mathcal{I} = (G, k, f_s, f_t)$ be a given instance of CR such that G is a q-tree. We first compute the value of $\omega(G)$ by Proposition 3.1, which can be done in linear time.

Consider the case where $\omega(G) \leq k - 1$. Since G is a q-tree and hence is a chordal graph, Lemma 3.1 implies that $\mathcal{I}$ is a yes-instance and there is a reconfiguration sequence whose length is at most $2|V(G)|^2$.

On the other hand, consider the remaining case, that is, $\omega(G) = k$ holds by Observation 2.1. In this case, Proposition 3.1 and Observation 3.1 imply that every vertex in G is frozen on both f_s and f_t. Therefore, $\mathcal{I}$ is a yes-instance if and only if $f_s = f_t$; this can be checked in linear time. □

We note that the quadratic upper bound on the length of a reconfiguration sequence given in Theorem 3.5 is asymptotically tight. More precisely, Bonamy et al. [2] gave an infinite family of yes-instances (G, k, f_s, f_t) such that G is a q-tree and the shortest length of a reconfiguration sequence is $\Omega(|V(G)|^2)$ for each integer $q \geq 1$.

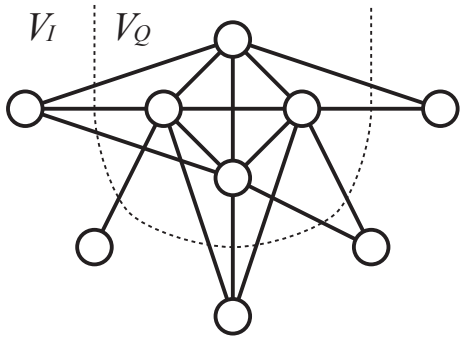


Figure 3.4: An example of a split graph, whose vertex set can be partitioned into a clique V_Q and an independent set V_I.

3.3.2 Split graphs

In this subsection, we consider split graphs. Recall that a graph is split if its vertex set can be partitioned into a clique and an independent set. (See Figure 3.4 as an example.)

Theorem 3.6 CR *can be solved in linear time for split graphs. Furthermore, if* (G, k, f_s, f_t) *is a yes-instance, there is a reconfiguration sequence whose length is at most* $2|V(G)|^2$.

Proof. We give such a linear-time algorithm for split graphs. Let $\mathcal{I} = (G, k, f_s, f_t)$ be a given instance of CR such that G is split. We first obtain a partition of $V(G)$ into a clique V_Q and an independent set V_I such that V_Q has the maximum size $\omega(G)$. Such a partition can be obtained in linear time [31]. By Observation 2.1, we have $|V_Q| = \omega(G) \leq k$. Therefore, there are two cases to consider.

Case 1: $|V_Q| < k$.

In this case, $|V_Q| = \omega(G) \leq k - 1$ holds. Since G is split and hence is a chordal graph, Lemma 3.1 implies that $\mathcal{I}$ is a yes-instance and there is a reconfiguration sequence whose length is at most $2|V(G)|^2$.

Case 2: $|V_Q| = k$.

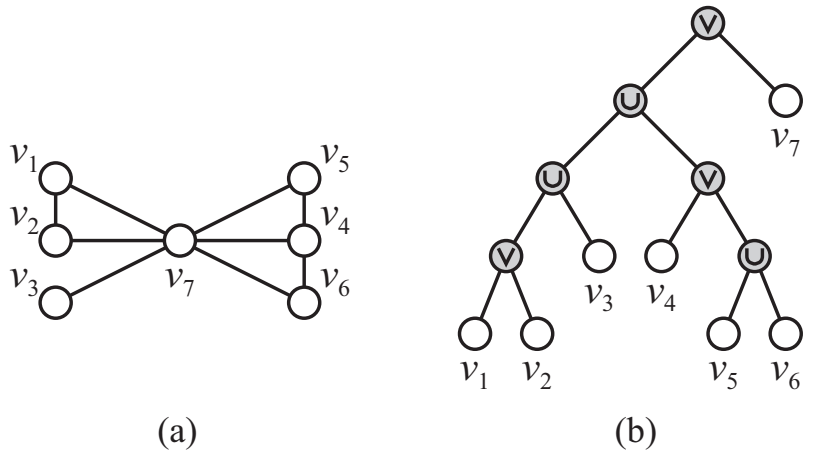


Figure 3.5: (a) A trivially perfect graph, and (b) its corresponding cotree.

In this case, every vertex in V_Q is frozen on both f_s and f_t. Thus, $\mathcal{I}$ is a no-instance if there exists a vertex $u \in V_Q$ such that $f_s(u) \neq f_t(u)$. Otherwise, because V_I is an independent set and both f_s and f_t are proper k-colorings of G, we can directly recolor each vertex $w \in V_I$ from $f_s(w)$ to $f_t(w)$; which implies that $\mathcal{I}$ is a yes-instance and the length of a reconfiguration sequence is at most $|V_I| \leq 2|V(G)|^2$.

We finally estimate the running time of our algorithm. We can obtain desired subsets V_Q and V_I in linear time [31]. Then, the algorithm simply checks if $|V_Q| < k$, and if $f_s(u) = f_t(u)$ holds for every vertex $u \in V_Q$. Therefore, our algorithm runs in linear time. □

3.3.3 Trivially perfect graphs

In this subsection, we consider trivially perfect graphs. The class of trivially perfect graphs has many characterizations. We here give its recursive definition. For two vertex-disjoint graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their *union* $G_1 \cup G_2$ is the graph such that $V(G_1 \cup G_2) = V_1 \cup V_2$ and $E(G_1 \cup G_2) = E_1 \cup E_2$, while their *join* $G_1 \vee G_2$ is the graph such that $V(G_1 \vee G_2) = V_1 \cup V_2$ and $E(G_1 \vee G_2) = E_1 \cup E_2 \cup \{vw \colon v \in V_1, w \in V_2\}$. Then, a *trivially perfect graph* can be recursively

defined, as follows:

(1) a graph consisting of a single vertex is a trivially perfect graph;

(2) if G_1 and G_2 are trivially perfect graphs, then their union $G_1 \cup G_2$ is a trivially perfect graph; and

(3) if G_1 and G_2 are trivially perfect graphs such that G_2 consists of a single vertex u, then their join $G_1 \vee G_2$ is a trivially perfect graph.

Notice that, by the join operation (3) above, the single vertex u in G_2 becomes a universal vertex in $G_1 \vee G_2$.

Theorem 3.7 CR *can be solved in linear time for trivially perfect graphs. Furthermore, if* (G, k, f_s, f_t) *is a yes-instance, there is a reconfiguration sequence whose length is at most* $2|V(G)|^2$.

Proof. We give such a linear-time algorithm for trivially perfect graphs. Since any trivially perfect graph G is a cograph, we can represent G by a binary tree, called a cotree, which can be naturally obtained from the recursive definition of trivially perfect graphs (see Figure 3.5 as an example): a *cotree* $T = (V_T, E_T)$ of a trivially perfect graph G is a binary tree such that each leaf of T corresponds to a single vertex in G, and each internal node of T has exactly two children and is labeled with either union $\cup$ or join $\vee$; notice that, for each join node in T, one of the two children must be a leaf of T. Such a cotree of G can be constructed in linear time [46]. Each node $i \in V_T$ corresponds to a subgraph G_i of G which is induced by all vertices corresponding to the leaves of T that are the descendants of i in T. Clearly, $G_0 = G$ for the root 0 of T.

We note that the maximum clique sizes $\omega(G_i)$ for all $i \in V_T$ can be computed in linear time, by a bottom-up computation according to the cotree T, as follows:

$$\omega(G_i) = 1$$

if i is a leaf of T;

$$\omega(G_i) = \max\{\omega(G_x), \omega(G_y)\}$$

if i is a union node with children x and y; and

$$\omega(G_i) = \omega(G_x) + 1$$

if i is a join node with children x and y such that y is a leaf of T. Therefore, we assume without loss of generality that we are given a trivially perfect graph G together with its cotree $T = (V_T, E_T)$ such that the maximum clique size $\omega(G_i)$ is associated to each node $i \in V_T$.

Let $\mathcal{I} = (G, k, f_s, f_t)$ be a given instance of CR such that G is a trivially perfect graph. For each node $i \in V_T$ and a k-coloring f of G, we denote by f^i the k-coloring of the subgraph G_i such that $f^i(v) = f(v)$ holds for every $v \in V(G_i)$. We propose the following algorithm to solve the problem, and will prove its correctness.

Input: An instance $\mathcal{I} = (G, k, f_s, f_t)$ of CR such that G is a trivially perfect graph

Output: yes/no as the answer to $\mathcal{I}$

Step 1. If $|V(G)| = 1$ or $\omega(G) < k$, then return yes.

Step 2. In this step, G has more than one vertex, and hence the root of the cotree T is either a union node or a join node. Let x and y be two children of the root of T. Then, we execute either (a) or (b):

Case (a): The root is a union node.

Return yes if both (G_x, k, f_s^x, f_t^x) and (G_y, k, f_s^y, f_t^y) are yes-instances; otherwise return no.

Case (b): The root is a join node.

Assume that G_y consists of a single vertex u. Return no if $f_s(u) \neq f_t(u)$; otherwise return the answer to $(G_x, k - 1, f_s^x, f_t^x)$.

We first verify the correctness of Step 1. If $|V(G)| = 1$, then we can directly recolor the vertex w in G from $f_s(w)$ to $f_t(w)$; thus, $\mathcal{I}$ is a yes-instance. If $\omega(G) < k$, then Lemma 3.1 yields that $\mathcal{I}$ is a yes-instance because G is a trivially perfect graph and hence is a chordal graph. Thus, Step 1 correctly returns yes.

We then verify the correctness of Step 2(a). This step is executed when the root of T is a union node. Then, there is no edge between G_x and G_y. Therefore, it suffices to solve each of (G_x, k, f_s^x, f_t^x) and (G_y, k, f_s^y, f_t^y), and combine their answers. Thus, Step 2(a) works correctly.

We finally verify the correctness of Step 2(b). This step is executed when the root of T is a join node. In addition, $\omega(G) = k$ holds because we execute this step after Step 1. Since $u \in V(G_y)$ becomes a universal vertex in $G = G_x \vee G_y$, it is contained in any maximum clique in G. Since $\omega(G) = k$ holds, u is frozen on both f_s and f_t. Thus, $\mathcal{I}$ is a no-instance if $f_s(u) \neq f_t(u)$ holds. On the other hand, if $f_s(u) = f_t(u)$ holds, then no vertex in $V(G) \setminus \{u\}$ can use the color $f_s(u) = f_t(u)$ in any reconfiguration sequence, because u is a universal vertex in G and is frozen on both f_s and f_t. Therefore, (G, k, f_s, f_t) is a yes-instance if and only if $(G_x, k-1, f_s^x, f_t^x)$ is a yes-instance. Thus, Step 2(b) works correctly.

Although the algorithm above is written as a recursive function, it can be implemented so as to run in linear time, as follows: we first traverse the cotree T of a given (whole) trivially perfect graph G from the root to leaves, and assign the sub-instance to each node $i \in V_T$; we then solve the sub-instances from leaves to the root of T by combining their children's answers.

Finally, we show that there is a reconfiguration sequence whose length is at most $2|V(G)|^2$ for a yes-instance (G, k, f_s, f_t), by the induction on $|V(G)|$. If $|V(G)| = 1$, then the claim clearly holds. Otherwise we consider the following three cases according to the steps of our algorithm. If the algorithm returns yes in Step 1, Lemma 3.1 guarantees the existence of such a reconfiguration sequence. If the algorithm returns

yes in Step 2(a), $\mathcal{I}_x = (G_x, k, f_s^x, f_t^x)$ and $\mathcal{I}_y = (G_y, k, f_s^y, f_t^y)$ are both yes-instances. Then, by the induction hypothesis, both $\mathcal{I}_x$ and $\mathcal{I}_y$ admit reconfiguration sequences whose lengths are at most $2|V(G_x)|^2$ and $2|V(G_y)|^2$, respectively. By combining these two reconfiguration sequences serially, we obtain a reconfiguration sequence for $\mathcal{I}$ whose length is at most $2|V(G_x)|^2 + 2|V(G_y)|^2 \leq 2|V(G)|^2$. Finally, if the algorithm returns yes in Step 2(b), $\mathcal{I}_x = (G_x, k, f_s^x, f_t^x)$ is a yes-instance; and hence there is a reconfiguration sequence for $\mathcal{I}_x$ whose length is at most $2|V(G_x)|^2$ by the induction hypothesis. This implies that there is a reconfiguration sequence for $\mathcal{I}$ whose length is at most $2|V(G_x)|^2 \leq 2|V(G)|^2$.

This completes our proof of Theorem 3.7. □

Chapter 4 List Coloring Reconfiguration

In this chapter, we study the complexity of LCR.

4.1 PSPACE-completeness

In this section, we show the following theorem.

Theorem 4.1 LCR *is PSPACE-complete even for threshold graphs.*

Proof. Let $\mathcal{I} = (H, L, g_s, g_t)$ be an instance of LCR which satisfies the conditions (a)-(d) in Section 3.2.1. Our goal is to transform $\mathcal{I}$ into another instance in which a graph is threshold, without changing the reconfigurability. By introducing new color sets $C_3, C_4, \ldots, C_{|X|}$ and replacing colors in the lists and the given two L-colorings, we can first obtain an equivalent instance (H, L', g'_s, g'_t) which satisfies the following two conditions:

- for each $x_i \in X$, $L'(x_i) = C_i$ holds, where $C_1, C_2, \ldots, C_{|X|}$ are pairwise disjoint;
- for each $y_i^j \in Y$, $L'(y_i) \subseteq C_i \cup C_{i+1}$ holds.

We then add edges to obtain a threshold graph as follows. Let $E' = E(H) \cup (X^2) \cup (X \times Y)$, and let $H' = (V(H), E')$. Observe that the lists of two endpoints of every added edge $vw \in E' \setminus E(H)$ have no common colors, and hence this transformation does not affect the existence of L-colorings. Moreover, H' is threshold; set the threshold $s = 1$, and the mapping $\omega(x) = 1$ for each $x \in X$ and $\omega(y) = 0$ for each $y \in Y$. Thus, this completes the proof. □

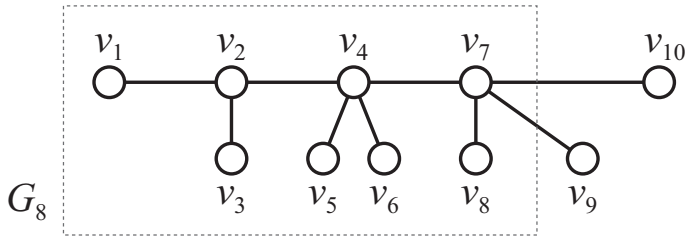


Figure 4.1: A caterpillar G and its vertex ordering, where the subgraph surrounded by a dotted rectangle corresponds to G_8.

Recall that any threshold graph is also a split graph, and observe that the graph H' constructed in the above proof is indeed a $|X|$-tree and a trivially perfect graph. Therefore, Theorems 3.5, 3.6 and 3.7 in the previous chapter is unlikely to be extended to LCR. On the other hand, as we will show in Part II, LCR becomes tractable for threshold graphs (i.e., split cographs) when parameterized by k.

4.2 A polynomial-time algorithm for graphs with pathwidth one

In contrast to the PSPACE-completeness of LCR for graphs with pathwidth two [60], we give a polynomial-time algorithm for graphs with pathwidth one.

Theorem 4.2 LCR *can be solved in polynomial time for graphs with pathwidth one.*

As a proof of Theorem 4.2, we give such an algorithm. Since every connected graph of pathwidth one is a caterpillar [53], it suffices to develop a polynomial-time algorithm for caterpillars.

A *caterpillar* G is a tree whose vertex set $V(G)$ can be partitioned into two subset V_S and V_L such that $G[V_S]$ forms a path and each vertex in V_L is adjacent with exactly one vertex in V_S. We may assume without loss of generality that the two endpoints of the path $G[V_S]$ are of degree one in G. (See v_1 and v_{10} in Figure 4.1.) We call each vertex in V_S a *spine vertex* of G, and each vertex in V_L a *leaf* of G.

We assume that all vertices in G are ordered as $v_1, v_2, \ldots, v_n$ by the breadth-first search starting with the endpoint (degree-1 vertex) of the path $G[V_S]$ with the priority to leaves; that is, when we visit a spine vertex v, we first visit all leaves of v and then visit the unvisited spine vertex. (See Figure 4.1 for example.) For each index $i \in \{1, 2, \ldots, n\}$, we let $V_i = \{v_1, v_2, \ldots, v_i\}$ and $G_i = G[V_i]$. Then, clearly $G_n = G$. For each index $i \in \{1, 2, \ldots, n\}$, let $\mathsf{sp}(i)$ be the latest spine vertex in V_i, that is, $\mathsf{sp}(i) = v_i$ if v_i is a spine vertex, otherwise $\mathsf{sp}(i)$ is the unique neighbor of v_i. Then, v_i is adjacent with only the spine vertex $\mathsf{sp}(i-1)$ in G_i for each $i \in \{2, 3, \ldots, n\}$.

We first define the *reconfiguration graph* R_G^L for a graph G with a list L, which is essentially equivalent to the solution graph, as follows. (See Figure 4.2 for an example.) The vertex set of R_G^L is the set of all L-colorings of G. We call each vertex of R_G^L a *node* in order to distinguish it from a vertex of G. Then, two nodes (i.e., L-colorings of G) f and f' of R_G^L are joined by an (undirected) edge if and only if $\mathsf{dif}(f, f')$ contains exactly one vertex $v \in V$, that is, f' can be obtained from f by changing the color assignment of a single vertex v.

We next prove that the size of each list can be restricted without loss of generality. Note that the following lemma holds for any graph.

Lemma 4.1 *For an instance (G', L', f'_s, f'_t), one can obtain another instance (G, L, f_s, f_t) in polynomial time such that $2 \leq |L(v)| \leq d(G, v) + 1$ for each vertex $v \in V(G)$, and (G', L', f'_s, f'_t) is a yes-instance if and only if (G, L, f_s, f_t) is a yes-instance.*

Proof. If $|L(v)| = 1$ for a vertex $v \in V(G')$, then any L-coloring of G' assigns the same color $c \in L(v)$ to v. Therefore, c is never assigned to any neighbor u of v. We can thus delete v from G' and set $L(u) := L(u) \setminus \{c\}$ for all neighbors u of v in G'. Clearly, this modification does not affect the reconfigurability (i.e., the existence or non-existence of a path in the reconfiguration graph).

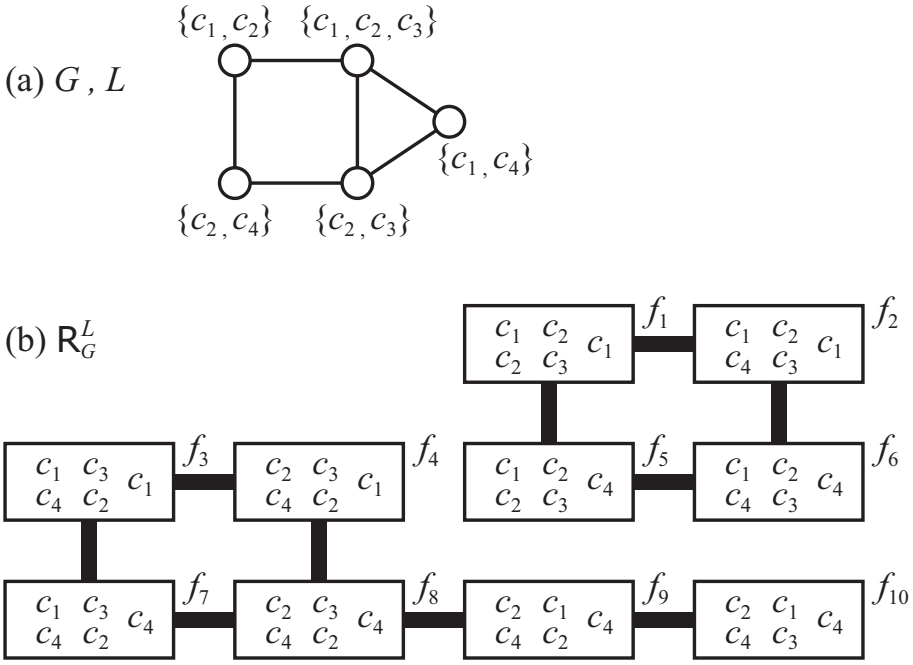


Figure 4.2: (a) A graph G and a list L, and (b) the reconfiguration graph R_G^L.

If $|L(v)| \geq d(G,v)+2$ for a vertex $v \in V(G')$, we simply delete v from G' without any modification of lists; let G be the resulting graph. Let f be any L-coloring of G, and consider a recoloring step for a neighbor u of v from the current color $c = f(u)$ to another color c'. We claim that this recoloring step can be simulated in G', as follows. If c' is not assigned to v in G', we can directly recolor u from c to c'. Thus, suppose that c' is assigned to v in G'. Then, since $|L(v)| \geq d(G,v)+2$, there is at least one color $c^* \in L(v)$ which is not c' and is not assigned to any of $d(G,v)$ neighbors of v. Therefore, we first recolor v from c' to c^*, and then recolor u from c to c'. In this way, any recoloring step in G can be simulated in G', and hence the modification does not affect the reconfigurability.

Thus, we can obtain an instance such that $2 \leq |L(v)| \leq d(G,v)+1$ holds for each vertex v without affecting the reconfigurability. Clearly, the modified instance can be constructed in polynomial time. □

Therefore, in the remainder of this chapter, we assume that G is a (connected) caterpillar and $2 \leq |L(v)| \leq d(G,v)+1$ holds for every vertex $v \in V(G)$. In

particular, $|L(v)| = 2$ for every leaf v of G.

4.2.1 Idea and definitions

The main idea of our algorithm is to extend techniques developed for SHORTEST PATH RECONFIGURATION [3], and apply them to LCR for caterpillars. Our algorithm employs a dynamic programming method based on the vertex ordering $v_1, v_2, \ldots, v_n$ of G.

For each $i \in \{1, 2, \ldots, n\}$, let $\mathsf{R}^L_{G_i}$ be the reconfiguration graph for the subgraph G_i and the list L. Then, $\mathsf{R}^L_{G_i}$ contains all L-colorings of G_i as its nodes. Our algorithm efficiently constructs $\mathsf{R}^L_{G_i}$ for each $i = 1, 2, \ldots, n$, in this order. However, of course, the number of nodes in $\mathsf{R}^L_{G_i}$ cannot be bounded by a polynomial in the input size in general. We thus use the property that the vertex v_{i+1} (will be added to G_i) is adjacent with only the spine vertex $\mathsf{sp}(i)$ in G_{i+1}; and we "encode" the reconfiguration graph $\mathsf{R}^L_{G_i}$ into a polynomial sized graph, while keeping the information of (1) the color assigned to $\mathsf{sp}(i)$ and (2) the connectivity of nodes in $\mathsf{R}^L_{G_i}$.

Before explaining the encoding methods, we first note that it suffices to focus on only one connected component in $\mathsf{R}^L_{G_i}$ which contains the restriction of f_s, where the *restriction* of an L-coloring f of a graph G to a subgraph G' is an L-coloring g of G' such that $g(v) = f(v)$ holds for all vertices $v \in V(G')$. For notational convenience, we denote by $f[V_i]$ the restriction of an L-coloring f of a caterpillar G to its subgraph $G_i = (V_i, E_i)$, that is, $f[V_i] = f|_{V_i}$. Then, we have the following lemma.

Lemma 4.2 *Let g be an L-coloring of G_i such that $f_s[V_i]$ and g are contained in the same connected component in $\mathsf{R}^L_{G_i}$. Then, for each $j \in \{1, 2, \ldots, i-1\}$, $f_s[V_j]$ and $g[V_j]$ are contained in the same connected component in $\mathsf{R}^L_{G_j}$.*

Proof. Assume that $f_s[V_i]$ and g are contained in the same connected component in $\mathsf{R}^L_{G_i}$. Then, there exists a path in $\mathsf{R}^L_{G_i}$ between $f_s[V_i]$ and g. We contract all edges

in the path that correspond to recoloring vertices in $V_i \setminus V_j$. Since each edge in the resulting path corresponds to recoloring only one vertex in V_j, the resulting path must be contained as a path in $\mathsf{R}^L_{G_j}$ between $f_s[V_j]$ and $g[V_j]$. Therefore $f_s[V_j]$ and $g[V_j]$ are contained in the same connected component in $\mathsf{R}^L_{G_j}$, and hence the lemma follows. □

From now on, we thus focus on only the connected component of $\mathsf{R}^L_{G_i}$ which contains $f_s[V_i]$. Since the list is fixed to be L in the remainder of this section, we simply denote by R_i the reconfiguration graph $\mathsf{R}^L_{G_i}$, and by R^s_i the connected component of $\mathsf{R}_i = \mathsf{R}^L_{G_i}$ containing $f_s[V_i]$.

Encoding graph

We now partition the nodes of R^s_i into several subsets with respect to (1) the color assigned to $\mathsf{sp}(i)$, and (2) the connectivity of nodes in R^s_i. For two nodes g and g' of R^s_i with $g(\mathsf{sp}(i)) = g'(\mathsf{sp}(i))$, we write $g \sim_{\mathsf{sp}(i)} g'$ if R^s_i has a path $\langle g_1, g_2, \ldots, g_\ell \rangle$ such that $g_1 = g$, $g_\ell = g'$, and $g_j(\mathsf{sp}(i)) = g(\mathsf{sp}(i)) = g'(\mathsf{sp}(i))$ holds for every $j \in \{1, 2, \ldots, \ell\}$, that is, g can be reconfigured into g' without recoloring the vertex $\mathsf{sp}(i)$. Since the adjacency relation on L-colorings is symmetric (i.e., R_i is an undirected graph), it is easy to see that $\sim_{\mathsf{sp}(i)}$ is an equivalence relation. Thus, the node set of R^s_i can be uniquely partitioned by the relation $\sim_{\mathsf{sp}(i)}$. We denote by $\mathcal{G}^s_i$ the partition of the node set of R^s_i into equivalence classes with respect to $\sim_{\mathsf{sp}(i)}$.

We finally define our dynamic programming table. For each subgraph G_i, $i \in \{1, 2, \ldots, n\}$, our algorithm keeps track of four pieces of information $(H_i, \mathsf{col}_i, \mathsf{ini}_i, \mathsf{tar}_i)$, defined as follows.

- The *encoding graph* H_i of R^s_i: This graph H_i can be obtained from R^s_i by contracting each node set in $\mathcal{G}^s_i$ into a single node. (See Figure 4.3 as an example.) We will refer to an *e-node* of H_i in order to distinguish it from a

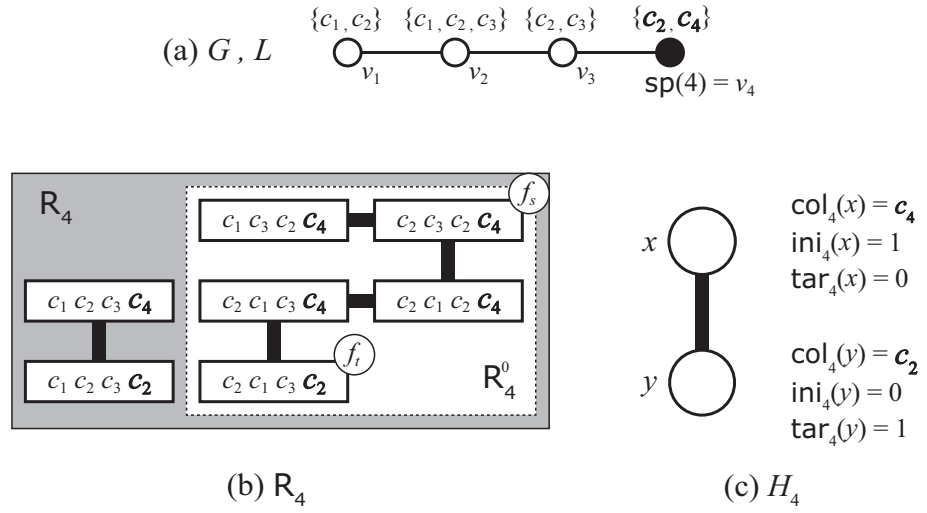


Figure 4.3: (a) A caterpillar $G = G_4$, (b) the reconfiguration graph R_4 consisting of all L-colorings of G, and (c) the encoding graph H_4 of R_4^s consisting of two e-nodes x and y, where each L-coloring in (b) is represented as the sequence of colors assigned to the vertices in G from left to right.

node of R_i^s. (Thus, each node refers to an L-coloring of G_i, and each e-node refers to a set of L-colorings of G_i.) For each e-node $x \in V(H_i)$, we denote by $\Phi_i(x)$ the set of all nodes in R_i^s that were contracted into x. (Note that we do not compute $\Phi_i(x)$, but use it only for definitions and proofs.)

- The color $\mathsf{col}_i(x) \in L(\mathsf{sp}(i))$ for each e-node $x \in V(H_i)$: This color $\mathsf{col}_i(x)$ is assigned to $\mathsf{sp}(i)$ in common by the nodes (i.e., L-colorings of G_i) in $\Phi_i(x)$.

- The label $\mathsf{ini}_i(x) \in \{0, 1\}$ for each e-node $x \in V(H_i)$: $\mathsf{ini}_i(x) = 1$ if $f_s[V_i] \in \Phi_i(x)$, otherwise $\mathsf{ini}_i(x) = 0$.

- The label $\mathsf{tar}_i(x) \in \{0, 1\}$ for each e-node $x \in V(H_i)$: $\mathsf{tar}_i(x) = 1$ if $f_t[V_i] \in \Phi_i(x)$, otherwise $\mathsf{tar}_i(x) = 0$.

To prove Theorem 4.2, we give a polynomial-time algorithm which computes $(H_i, \mathsf{col}_i, \mathsf{ini}_i, \mathsf{tar}_i)$ for each subgraph G_i, $i \in \{1, 2, \ldots, n\}$, by means of dynamic programming. Then, the problem can be solved as in the following lemma.

Lemma 4.3 *(G, L, f_s, f_t) is a yes-instance if and only if the encoding graph H_n contains a node x such that $\mathsf{tar}_n(x) = 1$.*

Proof. Since H_n contains a node x such that $\mathsf{tar}_n(x) = 1$, we have $f_t[V_n] = f_t \in \Phi_n(x)$. Recall that H_n is the encoding graph of R_n^s which contains the L-coloring f_s of G as a node. Since $\Phi_n(x) \subseteq V(\mathsf{R}_n^s)$, the lemma follows. □

4.2.2 Algorithm

As the initialization, we first consider the case where $i = 1$, that is, we compute $(H_1, \mathsf{col}_1, \mathsf{ini}_1, \mathsf{tar}_1)$. (See Figure 4.5(d) as an example.) Note that G_1 consists of a single vertex v_1, and recall that v_1 is a spine vertex of degree one. By Lemma 4.1 we have $|L(v_1)| = 2$. Therefore, the reconfiguration graph R_1 is a complete graph on $|L(v_1)| = 2$ nodes such that each node corresponds to an L-coloring of G_1 which assigns a distinct color to the vertex $\mathsf{sp}(1) = v_1$. Since R_1 is complete and contains the node $f_s[V_1]$, we have $\mathsf{R}_1^s = \mathsf{R}_1$. Furthermore, $H_1 = \mathsf{R}_1^s$ since all nodes in R_1^s assign distinct colors in $L(v_1)$ to $\mathsf{sp}(1) = v_1$. Then, for each e-node x of H_1 corresponding to the set of a single L-coloring g of G_1, we set

$$\begin{aligned} \mathsf{col}_1(x) &= g(v_1); \\ \mathsf{ini}_1(x) &= \begin{cases} 1 & \text{if } g(v_1) = f_s(v_1), \\ 0 & \text{otherwise;} \end{cases} \end{aligned}$$

and

$$\mathsf{tar}_1(x) = \begin{cases} 1 & \text{if } g(v_1) = f_t(v_1), \\ 0 & \text{otherwise.} \end{cases}$$

For $i \geq 2$, suppose that we have already computed $(H_{i-1}, \mathsf{col}_{i-1}, \mathsf{ini}_{i-1}, \mathsf{tar}_{i-1})$. Then, we compute $(H_i, \mathsf{col}_i, \mathsf{ini}_i, \mathsf{tar}_i)$, as follows.

Case (A): v_i is a leaf in V_{L}. (See Figs. 4.4(a) and 4.5(g).)

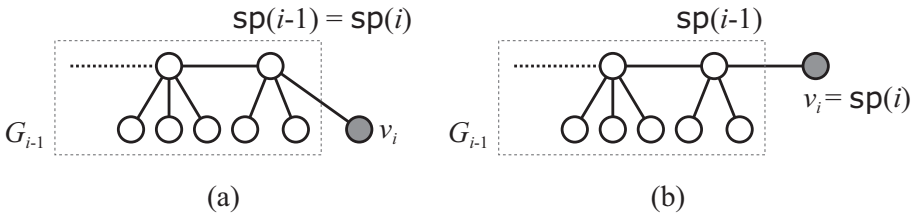


Figure 4.4: The graph G_i for (a) $v_i \in V_{\mathrm{L}}$ and (b) $v_i \in V_{\mathrm{S}}$.

By Lemma 4.1 we have $|L(v_i)| = 2$ in this case; let $L(v_i) = \{c_1, c_2\}$. Recall that v_i is adjacent with only the spine vertex $\mathsf{sp}(i-1)$ in G_i. Furthermore, $\mathsf{sp}(i) = \mathsf{sp}(i-1)$ in this case.

Let $H_{i-1}^{c_1}$ be the subgraph of H_{i-1} obtained by deleting all e-nodes y in H_{i-1} with $\mathsf{col}_{i-1}(y) = c_1$. Then, $H_{i-1}^{c_1}$ encodes all nodes of R_{i-1}^s that do not assign the color c_1 to $\mathsf{sp}(i-1)$. Thus, we can extend each L-coloring h of G_{i-1} encoded in $H_{i-1}^{c_1}$ to an L-coloring g of G_i such that $g(v_i) = c_1$ and $g(v) = h(v)$ for all vertices $v \in V_{i-1}$. Similarly, let $H_{i-1}^{c_2}$ be the subgraph of H_{i-1} obtained by deleting all e-nodes z in H_{i-1} with $\mathsf{col}_{i-1}(z) = c_2$.

We define an encoding graph $\widehat{H}_i'$ as $V(\widehat{H}_i') = V(H_{i-1}^{c_1}) \cup V(H_{i-1}^{c_2})$ and $E(\widehat{H}_i') = E(H_{i-1}^{c_1}) \cup E(H_{i-1}^{c_2})$. In other words, $\widehat{H}_i'$ can be obtained from H_{i-1} by deleting all edges $yz \in E(H_{i-1})$ such that $\mathsf{col}_{i-1}(y) = c_1$ and $\mathsf{col}_{i-1}(z) = c_2$. Let $\widehat{H}_i$ be the connected component of $\widehat{H}_i'$ that contains the e-node x such that $\mathsf{ini}_{i-1}(x) = 1$. For each e-node x in $\widehat{H}_i$, let $\widehat{\mathsf{col}}_i(x) = \mathsf{col}_{i-1}(x)$, $\widehat{\mathsf{ini}}_i(x) = \mathsf{ini}_{i-1}(x)$ and $\widehat{\mathsf{tar}}_i(x) = \mathsf{tar}_{i-1}(x)$. Then, we have the following lemma, whose proof will be given in Section 4.2.3.

Lemma 4.4 *For a leaf $v_i \in V_{\mathrm{L}}$, $(H_i, \mathsf{col}_i, \mathsf{ini}_i, \mathsf{tar}_i) = (\widehat{H}_i, \widehat{\mathsf{col}}_i, \widehat{\mathsf{ini}}_i, \widehat{\mathsf{tar}}_i)$.*

Case (B): v_i is a spine vertex in V_{S}. (See Figs. 4.4(b) and 4.5(e), (f), (h).)

In this case, notice that $\mathsf{sp}(i) = v_i$ in G_i, and hence we need to update col_i according to the color assigned to v_i.

We first define an encoding graph $\widehat{H}_i'$, as follows. For a color $c \in L(v_i)$, let H_{i-1}^{c} be the subgraph of H_{i-1} obtained by deleting all e-nodes y in H_{i-1} with $\mathsf{col}_{i-1}(y) = c$.

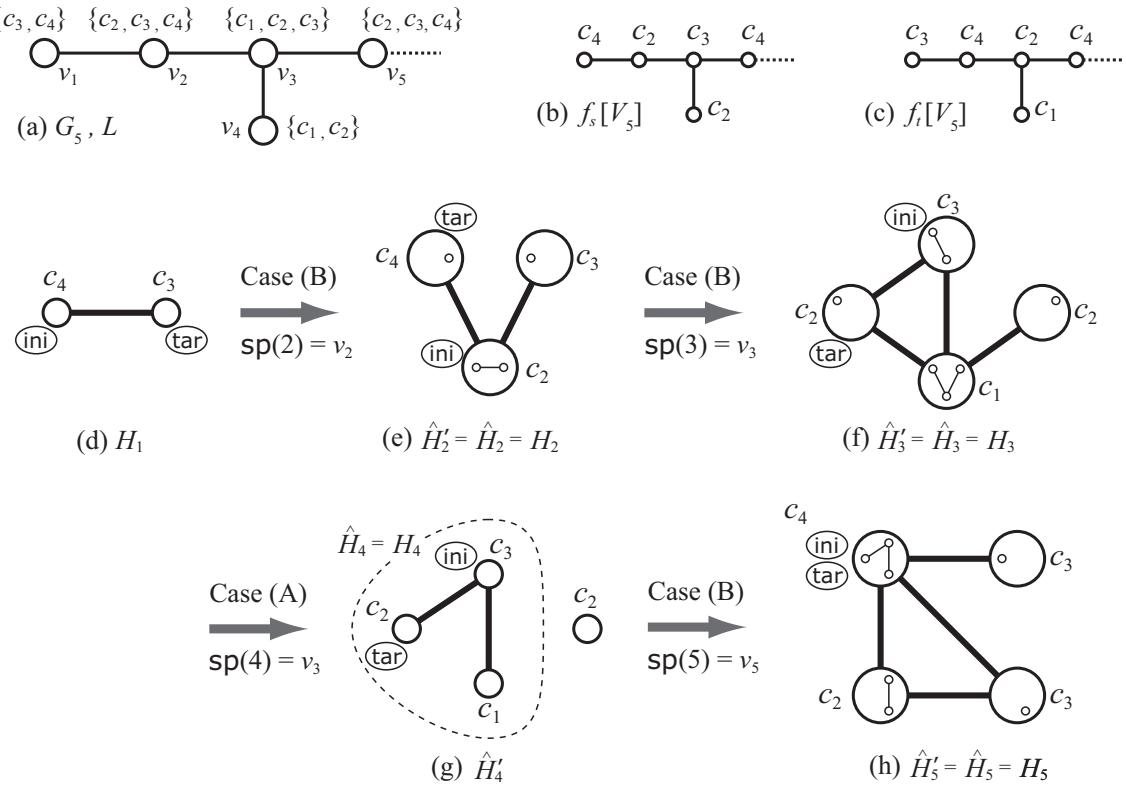


Figure 4.5: Application of our algorithm to the instance depicted in (a)–(c). In (d)–(h), $\mathsf{col}_i(x) \in L(\mathsf{sp}(i))$ is attached to each e-node x, and the e-nodes x with $\mathsf{ini}_i(x) = 1$ and $\mathsf{tar}_i(x) = 1$ have the labels "ini" and "tar," respectively. Furthermore, in (e), (f) and (h), the small graph contained in each e-node x of H_i represents the subgraph of H_{i-1} induced by $\mathsf{EN}(x)$.

For each connected component in H_{i-1}^c, we introduce a new e-node x to $\widehat{H}_i'$ such that $\widehat{\mathsf{col}}_i(x) = c$; we denote by $\mathsf{EN}(x)$ the set of all e-nodes in H_{i-1}^c that correspond to x. We apply this operation to all colors in $L(v_i)$. We then add edges to $\widehat{H}_i'$: two e-nodes x and y in $\widehat{H}_i'$ are joined by an edge if and only if $\mathsf{EN}(x) \cap \mathsf{EN}(y) \neq \emptyset$.

We now define $\widehat{\mathsf{ini}}_i(x)$ and $\widehat{\mathsf{tar}}_i(x)$ for each e-node x in $\widehat{H}_i'$, as follows:

$$\widehat{\mathsf{ini}}_i(x) = \begin{cases} 1 & \text{if } \widehat{\mathsf{col}}_i(x) = f_s(v_i) \text{ and} \\ & \mathsf{EN}(x) \text{ contains an e-node } z \\ & \text{with } \mathsf{ini}_{i-1}(z) = 1; \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\widehat{\mathsf{tar}}_i(x) = \begin{cases} 1 & \text{if } \widehat{\mathsf{col}}_i(x) = f_t(v_i) \text{ and} \\ & \mathsf{EN}(x) \text{ contains an e-node } z \\ & \text{with } \mathsf{tar}_{i-1}(z) = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Let $\widehat{H}_i$ be the connected component of $\widehat{H}'_i$ that contains the e-node x such that $\widehat{\mathsf{ini}}_i(x) = 1$. Then, we have the following lemma, whose proof will be given in Section 4.2.3.

Lemma 4.5 *For a spine vertex $v_i \in V_{\mathrm{S}}$, $(H_i, \mathsf{col}_i, \mathsf{ini}_i, \mathsf{tar}_i) = (\widehat{H}_i, \widehat{\mathsf{col}}_i, \widehat{\mathsf{ini}}_i, \widehat{\mathsf{tar}}_i)$.*

4.2.3 Correctness of the algorithm

To prove the correctness of our algorithm in Section 4.2.2, it suffices to prove Lemmas 4.4 and 4.5.

We first introduce some notation. For $i \geq 2$, let h be any node (i.e., an L-coloring of G_{i-1}) in the reconfiguration graph R_{i-1}. Recall that the vertex v_i is adjacent with only the spine vertex $\mathsf{sp}(i-1)$ in G_i. Let c be any color in $L(v_i) \setminus \{h(\mathsf{sp}(i-1))\}$. Then, we say that h can be *extended by* c to an L-coloring g of G_i such that $g(v_i) = c$ and $g(v) = h(v)$ for all vertices $v \in V_{i-1}$; we simply denote such an extension by $h + c = g$. Note that g is a node in R_i.

For any e-node y in the encoding graph H_{i-1} of R^s_{i-1}, recall that $\Phi_{i-1}(y)$ is the set of all nodes in R^s_{i-1} that were contracted into y. For a color $c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(y)\}$, let

$$\Phi_{i-1}(y) \oplus c = \{h + c : h \in \Phi_{i-1}(y)\},$$

that is, $\Phi_{i-1}(y) \oplus c$ is the set of nodes in R_i that are extended by c from nodes in $\Phi_{i-1}(y)$ $\big(\subseteq V(\mathsf{R}^s_{i-1})\big)$.

Proof of Lemma 4.4

Suppose that we have already computed $(H_{i-1}, \mathsf{col}_{i-1}, \mathsf{ini}_{i-1}, \mathsf{tar}_{i-1})$ for $i \geq 2$, and assume that $v_i \in V_{\mathrm{L}}$ and $L(v_i) = \{c_1, c_2\}$.

We first prove that $V(H_i) \subseteq V(\widehat{H}'_i)$ holds, as in the following lemma.

Lemma 4.6 *Let x be any e-node in H_i. Then, there exists an e-node $\widehat{x}$ in $V(\widehat{H}'_i) = V(H^{c_1}_{i-1}) \cup V(H^{c_2}_{i-1})$ such that*

$$\bigcup \big\{ \Phi_{i-1}(\widehat{x}) \oplus c : c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\} \big\} = \Phi_i(x).$$

Proof. Let g be any node in $\Phi_i(x) \subseteq V(\mathsf{R}^s_i)$. Then, by Lemma 4.2 the node $g[V_{i-1}]$ is contained in R^s_{i-1}, and hence there exists an e-node $\widehat{x}$ in H_{i-1} such that $g[V_{i-1}] \in \Phi_{i-1}(\widehat{x})$. Notice that, since $L(v_i) = \{c_1, c_2\}$, we have $V(H^{c_1}_{i-1}) \cup V(H^{c_2}_{i-1}) = V(H_{i-1})$. Thus, the e-node $\widehat{x}$ is contained in $\widehat{H}'_i$, too. Therefore, we will prove that $\bigcup \big\{ \Phi_{i-1}(\widehat{x}) \oplus c : c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\} \big\} = \Phi_i(x)$ for this e-node $\widehat{x}$.

Let g' be any node in $\Phi_i(x)$. Then, $g \sim_{\mathsf{sp}(i)} g'$, and hence R^s_i contains a path $\langle g_1, g_2, \ldots, g_\ell \rangle$ such that $g_1 = g$, $g_\ell = g'$ and $g_j(\mathsf{sp}(i)) = g(\mathsf{sp}(i)) = g'(\mathsf{sp}(i))$ for all $j \in \{1, 2, \ldots, \ell\}$. Thus, we can obtain a path $\langle g_1[V_{i-1}], g_2[V_{i-1}], \ldots, g_\ell[V_{i-1}] \rangle$ in which the spine vertex $\mathsf{sp}(i)$ always receives the same color $g(\mathsf{sp}(i)) = g'(\mathsf{sp}(i))$; note that $g_j[V_{i-1}] = g_{j+1}[V_{i-1}]$ may hold if v_i is recolored, but we can simply drop $g_{j+1}[V_{i-1}]$ in such a case. Since $\mathsf{sp}(i) = \mathsf{sp}(i-1)$ for the case where $v_i \in V_{\mathrm{L}}$, we have $g[V_{i-1}] \sim_{\mathsf{sp}(i-1)} g'[V_{i-1}]$. Therefore, $g'[V_{i-1}] \in \Phi_{i-1}(\widehat{x})$. Since $\mathsf{col}_{i-1}(\widehat{x})$ represents the color assigned to $\mathsf{sp}(i) = \mathsf{sp}(i-1)$ and v_i is adjacent with $\mathsf{sp}(i-1)$ in G_i, the color $g'(v_i)$ is clearly contained in $L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\}$. We thus have $g'[V_{i-1}] + g'(v_i) = g' \in \bigcup \big\{ \Phi_{i-1}(\widehat{x}) \oplus c : c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\} \big\}$.

Let g'' be any node in $\bigcup \big\{ \Phi_{i-1}(\widehat{x}) \oplus c : c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\} \big\}$ such that $g'' = h + c$ for some node h in $\Phi_{i-1}(\widehat{x})$ and $c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\}$. Since $h \in \Phi_{i-1}(\widehat{x})$ and $g[V_{i-1}] \in \Phi_{i-1}(\widehat{x})$, we have $g[V_{i-1}] \sim_{\mathsf{sp}(i-1)} h$ and hence R^s_{i-1} contains a path $\langle h_1, h_2, \ldots, h_\ell \rangle$ such that $h_1 = g[V_{i-1}]$, $g_\ell = h$ and $h_j(\mathsf{sp}(i-1)) = h(\mathsf{sp}(i-1))$ for all $j \in \{1, 2, \ldots, \ell\}$. Since $\mathsf{sp}(i) = \mathsf{sp}(i-1)$ and $g(v_i) \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\}$, the sequence $\langle h_1 + g(v_i), h_2 + g(v_i), \ldots, h_\ell + g(v_i) \rangle$ is a path in R^s_i. If $g(v_i) \neq c$, we add one more adjacent node $h_\ell + c$ to the last. Since $h_1 + g(v_i) = g$ and $h_\ell + c = g''$, we thus have

$g \sim_{\mathsf{sp}(i)} g''$ and hence $g'' \in \Phi_i(x)$. □

By Lemma 4.6 we identify each e-node x in H_i with the corresponding e-node $\widehat{x}$ in $\widehat{H}'_i$. We then prove the following lemma.

Lemma 4.7 *Let x and y be two e-nodes in H_i, and let $\widehat{x}$ and $\widehat{y}$ be two e-nodes in $\widehat{H}'_i$ corresponding to x and y, respectively. Then, $\widehat{x}\widehat{y} \in E(\widehat{H}'_i) = E(H^{c_1}_{i-1}) \cup E(H^{c_2}_{i-1})$ if and only if $xy \in E(H_i)$.*

Proof. We first prove the only-if direction. Suppose that $\widehat{x}\widehat{y} \in E(H^{c_1}_{i-1})$; it is symmetric for the other case where $\widehat{x}\widehat{y} \in E(H^{c_2}_{i-1})$. Then, there exist two adjacent nodes $h_x \in \Phi_{i-1}(\widehat{x})$ and $h_y \in \Phi_{i-1}(\widehat{y})$ such that $h_x(\mathsf{sp}(i-1)) \neq c_1$ and $h_y(\mathsf{sp}(i-1)) \neq c_1$. Therefore, by Lemma 4.6 we have $h_x + c_1 \in \Phi_i(x)$ and $h_y + c_1 \in \Phi_i(y)$. Note that, since $\widehat{x} \neq \widehat{y}$, we know that only the spine vertex $\mathsf{sp}(i-1) = \mathsf{sp}(i)$ is recolored between h_x and h_y. Thus, $h_x + c_1$ and $h_y + c_1$ are adjacent in R^s_i, and hence we have $xy \in E(H_i)$.

We then prove the if direction. Since $xy \in E(H_i)$, there exist two adjacent nodes $g_x \in \Phi_i(x)$ and $g_y \in \Phi_i(y)$ in R^s_i. Since $x \neq y$, only the spine vertex $\mathsf{sp}(i)$ is recolored between g_x and g_y, and hence $g_x(v_i) = g_y(v_i)$. Therefore, $g_x[V_{i-1}]$ and $g_y[V_{i-1}]$ are adjacent. We assume that $c_1 = g_x(v_i) = g_y(v_i)$ without loss of generality. Then, since v_i and $\mathsf{sp}(i)$ are adjacent, $g_x(\mathsf{sp}(i)) \neq c_1$ and $g_y(\mathsf{sp}(i)) \neq c_1$. By Lemma 4.6 we have $g_x \in \bigcup\{\Phi_{i-1}(\widehat{x}) \oplus c : c \in L(v_i) \setminus \{\mathsf{col}_{i-1}(\widehat{x})\}\}$; this implies that $g_x[V_{i-1}] \in \Phi_{i-1}(\widehat{x})$. Similarly, $g_y[V_{i-1}] \in \Phi_{i-1}(\widehat{y})$. Since $g_x[V_{i-1}]$ and $g_y[V_{i-1}]$ are adjacent, $\widehat{x}\widehat{y} \in E(H_{i-1})$. Furthermore, since $\mathsf{col}_{i-1}(\widehat{x}) \neq c_1$ and $\mathsf{col}_{i-1}(\widehat{y}) \neq c_1$, we have $\widehat{x}\widehat{y} \in E(H^{c_1}_{i-1})$. Therefore, $\widehat{x}\widehat{y} \in E(H^{c_1}_{i-1}) \subseteq E(\widehat{H}'_i)$. □

We now prove the following lemma.

Lemma 4.8 $\widehat{H}_i = H_i$.

Proof. Recall that H_i consists of a single connected component which contains the e-node z such that $f_s[V_i] \in \Phi_i(z)$. Consider the set of all e-nodes $\widehat{x}$ in $\widehat{H}'_i$ that

correspond to the e-nodes x in H_i. By Lemma 4.7 the e-node set forms a connected subgraph of a single connected component in $\widehat{H}'_i$. Furthermore, by Lemma 4.6 the component in $\widehat{H}'_i$ contains the e-node $\widehat{z}$ such that $f_s[V_i] \in \Phi_i(\widehat{z})$; by the construction, $\widehat{z}$ is contained in $\widehat{H}_i$. We thus have $V(H_i) \subseteq V(\widehat{H}_i)$.

Therefore, to show $H_i = \widehat{H}_i$, by Lemma 4.7 it suffices to prove that there exists no edge $x\widehat{y} \in E(\widehat{H}_i)$ which joins two e-nodes $x \in V(H_i)$ and $\widehat{y} \in V(\widehat{H}_i) \setminus V(H_i)$. Suppose for a contradiction that there exists such an edge $x\widehat{y} \in E(\widehat{H}_i)$. By the construction, $x\widehat{y} \in E(H_{i-1}^{c_1}) \cup E(H_{i-1}^{c_2})$; we may assume that $x\widehat{y} \in E(H_{i-1}^{c_1})$ without loss of generality. Then, there exist two adjacent nodes $h_x \in \Phi_{i-1}(x)$ and $h_{\widehat{y}} \in \Phi_{i-1}(\widehat{y})$ such that $h_x(\mathsf{sp}(i-1)) \neq c_1$ and $h_{\widehat{y}}(\mathsf{sp}(i-1)) \neq c_1$. Therefore h_x and $h_{\widehat{y}}$ can be extended by c_1, and $h_x + c_1 \in \Phi_i(x)$ and $h_{\widehat{y}} + c_1 \in \Phi_i(\widehat{y})$ are adjacent in R_i^s. By Lemma 4.6 we then have $\widehat{y} \in V(H_i)$; this contradicts the assumption that $\widehat{y} \in V(\widehat{H}_i) \setminus V(H_i)$. □

Finally, we show the following lemma.

Lemma 4.9 *$\widehat{\mathsf{col}}_i = \mathsf{col}_i$, $\widehat{\mathsf{ini}}_i = \mathsf{ini}_i$ and $\widehat{\mathsf{tar}}_i = \mathsf{tar}_i$.*

Proof. Recall that $\mathsf{sp}(i) = \mathsf{sp}(i-1)$ if $v_i \in V_{\mathrm{L}}$. Then, the lemma follows immediately from Lemma 4.6. □

This completes the proof of Lemma 4.4.

Proof of Lemma 4.5

Suppose that we have already computed $(H_{i-1}, \mathsf{col}_{i-1}, \mathsf{ini}_{i-1}, \mathsf{tar}_{i-1})$ for $i \geq 2$, and assume that $v_i \in V_{\mathrm{S}}$.

We first prove that $V(H_i) \subseteq V(\widehat{H}'_i)$ holds, as in the following lemma.

Lemma 4.10 *Let x be any e-node in H_i with $\mathsf{col}_i(x) = c$. Then, there exists exactly one connected component H in H_{i-1}^c such that*

$$\bigcup \{\Phi_{i-1}(y) \oplus c : y \in V(H)\} = \Phi_i(x).$$

Proof. Let g be any node in $\Phi_i(x) \subseteq V(\mathsf{R}_i^s)$. Then, by Lemma 4.2 the node $g[V_{i-1}]$ is contained in R_{i-1}^s, and hence there exists an e-node z in H_{i-1} such that $g[V_{i-1}] \in \Phi_{i-1}(z)$. Since $\mathsf{sp}(i) \neq \mathsf{sp}(i-1)$ and $\mathsf{sp}(i)$ is adjacent with $\mathsf{sp}(i-1)$, the assumption $\mathsf{col}_i(x) = c$ implies that $g(\mathsf{sp}(i)) = c$ and hence $g(\mathsf{sp}(i-1)) \neq c$. Therefore, we have $\mathsf{col}_{i-1}(z) \neq c$, and hence H_{i-1}^c has exactly one connected component H that contains z. We thus prove that $\bigcup\{\Phi_{i-1}(y) \oplus c : y \in V(H)\} = \Phi_i(x)$ for the connected component H.

Let g' be any node in $\Phi_i(x)$. Then, $g'(\mathsf{sp}(i)) = c$ and hence it suffices to show that H contains an e-node y such that $g'[V_{i-1}] \in \Phi_{i-1}(y)$. Since $g \sim_{\mathsf{sp}(i)} g'$, there exists a path $\langle g_1, g_2, \ldots, g_\ell \rangle$ in R_i^s such that $g_1 = g$, $g_\ell = g'$ and $g_j(\mathsf{sp}(i)) = g(\mathsf{sp}(i)) = g'(\mathsf{sp}(i)) = c$ for all $j \in \{1, 2, \ldots, \ell\}$. Since $\mathsf{sp}(i)$ is adjacent with $\mathsf{sp}(i-1)$, $g_j(\mathsf{sp}(i-1)) \neq c$ holds for all $j \in \{1, 2, \ldots, \ell\}$. Therefore, there exists a connected component H' in H_{i-1}^c such that the path $\langle g_1[V_{i-1}], g_2[V_{i-1}], \ldots, g_\ell[V_{i-1}] \rangle$ is contained in $\bigcup\{\Phi_{i-1}(y') : y' \in V(H')\}$. Because $z \in V(H)$ and $g_1[V_{i-1}] = g[V_{i-1}] \in \Phi_{i-1}(z)$, we have $H' = H$. Thus, H contains an e-node y such that $g'[V_{i-1}] \in \Phi_{i-1}(y)$. Then, we have $g' \in \bigcup\{\Phi_{i-1}(y) \oplus c : y \in V(H)\}$.

Conversely, let g'' be any node in $\bigcup\{\Phi_{i-1}(y) \oplus c : y \in V(H)\}$. Since H is connected, the subgraph of R_{i-1}^s induced by $\bigcup\{\Phi_{i-1}(y) : y \in V(H)\}$ is connected, too. Then, the induced subgraph contains a path $\langle h_1, h_2, \ldots, h_\ell \rangle$ such that $h_1 = g[V_{i-1}]$ and $h_\ell = g''[V_{i-1}]$. Furthermore, since H is a connected component in H_{i-1}^c, we know that $h_j(\mathsf{sp}(i-1)) \neq c$ for all nodes h_j, $j \in \{1, 2, \ldots, \ell\}$. Therefore, we can extend each node h_j by c, and obtain a path $\langle h_1 + c, h_2 + c, \ldots, h_\ell + c \rangle$. Since $h_1 + c = g$ and $h_\ell + c = g''$, we thus have $g \sim_{\mathsf{sp}(i)} g''$. Since $g \in \Phi_i(x)$, we have $g'' \in \Phi_i(x)$. □

For each e-node $x \in V(H_i)$, let H_x be the connected component in $H_{i-1}^{\mathsf{col}_i(x)}$ which satisfies Lemma 4.10. Then, we can identify the e-node x in H_i with the e-node $\widehat{x}$ in $\widehat{H}_i'$ such that $\mathsf{EN}(\widehat{x}) = V(H_x)$. We then prove the following lemma.

Lemma 4.11 *Let x and y be two e-nodes in H_i, and let $\widehat{x}$ and $\widehat{y}$ be two e-nodes in*

$\widehat{H}'_i$ *corresponding to* x *and* y*, respectively. Then,* $\mathsf{EN}(\widehat{x}) \cap \mathsf{EN}(\widehat{y}) \neq \emptyset$ *if and only if* $xy \in E(H_i)$.

Proof. We first prove the only-if direction. Suppose that the set $\mathsf{EN}(\widehat{x}) \cap \mathsf{EN}(\widehat{y})$ contains an e-node a in H_{i-1}. Choose an arbitrary node $h \in \Phi_{i-1}(a)$, then two nodes $h + \mathsf{col}_i(x) \in \Phi_i(x)$ and $h + \mathsf{col}_i(y) \in \Phi_i(y)$ are adjacent in R_i^s. Thus, $xy \in E(H_i)$.

We then prove the if direction. Suppose that $xy \in E(H_i)$, then there exist two adjacent nodes $g_x \in \Phi_i(x)$ and $g_y \in \Phi_i(y)$ in R_i^s. Since g_x and g_y are adjacent and $\mathsf{col}_i(x) \neq \mathsf{col}_i(y)$, we know that $g_x[V_{i-1}] = g_y[V_{i-1}]$. By Lemma 4.10, there exists an e-node $a \in \mathsf{EN}(\widehat{x})$ such that $g_x[V_{i-1}] \in \Phi_{i-1}(a)$. Similarly, there exists an e-node $b \in \mathsf{EN}(\widehat{y})$ such that $g_y[V_{i-1}] \in \Phi_{i-1}(b)$. Since $g_x[V_{i-1}] = g_y[V_{i-1}]$, we have $a = b$. Therefore, $a = b \in \mathsf{EN}(\widehat{x}) \cap \mathsf{EN}(\widehat{y}) \neq \emptyset$. □

By Lemmas 4.10 and 4.11, we have $H_i \subseteq \widehat{H}'_i$ and $\mathsf{col}_i = \widehat{\mathsf{col}}_i$. We now prove the following lemma.

Lemma 4.12 $\mathsf{ini}_i = \widehat{\mathsf{ini}}_i$ *and* $\mathsf{tar}_i = \widehat{\mathsf{tar}}_i$.

Proof. We prove only $\mathsf{ini}_i = \widehat{\mathsf{ini}}_i$; it is similar to prove $\mathsf{tar}_i = \widehat{\mathsf{tar}}_i$.

Let x be any e-node in H_i such that $\mathsf{ini}_i(x) = 1$. Then, $f_s[V_i] \in \Phi_i(x)$, and $\mathsf{col}_i(x) = \widehat{\mathsf{col}}_i(x) = f_s(v_i)$ holds. By Lemma 4.10 there exists an e-node $y \in \mathsf{EN}(x)$ such that $f_s[V_{i-1}] \in \Phi_{i-1}(y)$ and $\mathsf{ini}_{i-1}(y) = 1$. Since $f_s[V_i] = f_s[V_{i-1}] + \mathsf{col}_i(x)$, we thus have $\widehat{\mathsf{ini}}_i(x) = 1$.

Conversely, let $\widehat{x}$ be any e-node in $\widehat{H}'_i$ such that $\widehat{\mathsf{ini}}_i(\widehat{x}) = 1$. Then, $\mathsf{EN}(\widehat{x})$ contains an e-node y such that $f_s[V_{i-1}] \in \Phi_{i-1}(y)$ and $\mathsf{ini}_{i-1}(y) = 1$. Note that y is in $H_{i-1}^{\mathsf{col}_i(\widehat{x})}$, and $f_s[V_{i-1}] + \mathsf{col}_i(\widehat{x}) = f_s[V_i]$. By Lemma 4.10 we thus have $f_s[V_i] \in \Phi_i(\widehat{x})$ and hence $\mathsf{ini}_i(\widehat{x}) = 1$. □

Finally, we show following lemma.

Lemma 4.13 $\widehat{H}_i = H_i$.

Proof. Recall that H_i consists of a single connected component which contains the e-node z such that $f_s[V_i] \in \Phi_i(z)$. By Lemma 4.11 H_i is contained in one connected component of $\widehat{H}'_i$ as a subgraph, and the connected component contains an e-node $\widehat{z}$ such that $f_s[V_i] \in \Phi_i(\widehat{z})$. By Lemma 4.12 we have $\widehat{\mathsf{ini}}_i(\widehat{z}) = 1$, and hence the connected component is indeed $\widehat{H}_i$. We thus have $H_i \subseteq \widehat{H}_i$.

Therefore, to show $H_i = \widehat{H}_i$, by Lemma 4.11 it suffices to prove that there exists no edge $x\widehat{y} \in E(\widehat{H}_i)$ which joins two e-nodes $x \in V(H_i)$ and $\widehat{y} \in V(\widehat{H}_i) \setminus V(H_i)$. Suppose for a contradiction that there exists such an edge $x\widehat{y} \in E(\widehat{H}_i)$. Then, the set $\mathsf{EN}(x) \cap \mathsf{EN}(\widehat{y})$ contains an e-node z. Choose an arbitrary node $h \in \Phi_{i-1}(z)$, then two nodes $h + \mathsf{col}_i(x)$ and $h + \mathsf{col}_i(\widehat{y})$ are adjacent in R^s_i. Then, $h + \mathsf{col}_i(\widehat{y}) \in \bigcup_{z' \in \mathsf{EN}(\widehat{y})} (\Phi_{i-1}(z') \oplus \mathsf{col}_i(\widehat{y}))$. By Lemma 4.10 the corresponding e-node y should be contained in H_i; this contradicts the assumption that $\widehat{y} \in V(\widehat{H}_i) \setminus V(H_i)$. □

This completes the proof of Lemma 4.5.

4.2.4 Running time

We now estimate the running time of our algorithm in Section 4.2.2. The following is the key lemma for the estimation.

Lemma 4.14 *For each index $i \in \{1, 2, \ldots, n\}$,*

$$|V(H_i)| \leq \begin{cases} 2 & \text{if } i = 1; \\ |V(H_{i-1})| + d(G, v_i) & \text{otherwise.} \end{cases}$$

In particular, $|V(H_n)| = O(n)$, where n is the number of vertices in G.

We now prove Lemma 4.14. Since $|V(\widehat{H}_i)| \leq |V(\widehat{H}'_i)|$ holds, it suffices to prove the following inequality: for each index $i \in \{1, 2, \ldots, n\}$,

$$|V(\widehat{H}'_i)| \leq \begin{cases} 2 & \text{if } i = 1; \\ |V(H_{i-1})| + d(G, v_i) & \text{otherwise.} \end{cases} \tag{4.1}$$

Consider the case where v_i is a leaf. Then, $V(\widehat{H}'_i) = V(H_{i-1})$, and hence Eq. (4.1) clearly holds. In the remainder of this subsection, we thus consider the case where v_i is a spine vertex.

For a graph $G = (V, E)$, we denote by $\mathsf{cc}(G)$ the number of connected components in G. For a connected graph G, that is, $\mathsf{cc}(G) = 1$, we denote by T_G any spanning tree of G. Since $E(T_G) \subseteq E(G)$, we clearly have the following proposition.

Proposition 4.1 *Let G be a connected graph, and let V_0 be any vertex subset of G. Then, $\mathsf{cc}(G[V_0]) \leq \mathsf{cc}(T_G[V_0])$.*

We now apply Case (B) of our algorithm to any spanning tree $T_{H_{i-1}}$ of H_{i-1}, instead of applying the operation to H_{i-1}. Let $\widehat{H}_i^T$ be the obtained encoding graph, instead of $\widehat{H}'_i$. Then, we have the following lemma.

Lemma 4.15 $|V(\widehat{H}'_i)| \leq |V(\widehat{H}_i^T)|$.

Proof. For each color $c \in L(v_i)$, let $H_{i-1}^{T,c}$ be the subgraph of $T_{H_{i-1}}$ obtained by deleting all e-nodes y in $T_{H_{i-1}}$ with $\mathsf{col}_{i-1}(y) = c$. Then,

$$|V(\widehat{H}'_i)| = \sum_{c \in L(v_i)} \mathsf{cc}(H_{i-1}^c),$$

and

$$|V(\widehat{H}_i^T)| = \sum_{c \in L(v_i)} \mathsf{cc}(H_{i-1}^{T,c}). \tag{4.2}$$

By Proposition 4.1 we have $\mathsf{cc}(H_{i-1}^c) \leq \mathsf{cc}(H_{i-1}^{T,c})$ for each color $c \in L(v_i)$, and hence $|V(\widehat{H}'_i)| \leq |V(\widehat{H}_i^T)|$. □

We finally show the following lemma, which verifies Eq. (4.1) and hence completes the proof of Lemma 4.14.

Lemma 4.16 *If v_i is a spine vertex, then $|V(\widehat{H}'_i)| \leq |V(H_{i-1})| + d(G, v_i)$.*

Proof. We first consider the case where $|V(H_{i-1})| = 1$. Then, $\mathsf{cc}(H_{i-1}^c) \leq 1$ for any color $c \in L(v_i)$, and hence

$$|V(\widehat{H}_i')| = \sum_{c \in L(v_i)} \mathsf{cc}(H_{i-1}^c) \leq |L(v_i)|.$$

By Lemma 4.1 we have $|L(v_i)| \leq d(G, v_i) + 1$, and hence

$$|V(\widehat{H}_i')| \leq 1 + d(G, v_i) = |V(H_{i-1})| + d(G, v_i).$$

We then consider the case where $|V(H_{i-1})| \geq 2$. Recall that $T_{H_{i-1}}$ is a spanning tree of H_{i-1}, and hence $V(T_{H_{i-1}}) = V(H_{i-1})$. For each color $c \in L(v_i)$, let

$$X_{i-1}(c) = \{x \in V(T_{H_{i-1}}) : \mathsf{col}_{i-1}(x) = c\}.$$

For each vertex $x \in V(T_{H_{i-1}})$, we denote by $d(T_{H_{i-1}}, x)$ the degree of x in $T_{H_{i-1}}$. Then, by deleting x from $T_{H_{i-1}}$, the number of connected components in the resulting graph is increased by $d(T_{H_{i-1}}, x) - 1$. We thus have

$$\begin{aligned} \mathsf{cc}(H_{i-1}^{T,c}) &= \mathsf{cc}(T_{H_{i-1}}) + \sum \Big\{ d(T_{H_{i-1}}, x) - 1 : x \in X_{i-1}(c) \Big\} \\ &= 1 + \sum \Big\{ d(T_{H_{i-1}}, x) - 1 : x \in X_{i-1}(c) \Big\}. \end{aligned} \tag{4.3}$$

By Lemma 4.15 and Eq. (4.2) we have

$$|V(\widehat{H}_i')| \leq |V(\widehat{H}_i^T)| = \sum_{c \in L(v_i)} \mathsf{cc}(H_{i-1}^{T,c}).$$

Therefore, by Eq. (4.3) we have

$$\begin{aligned} |V(\widehat{H}_i')| &\leq \sum_{c \in L(v_i)} \Big(1 + \sum \Big\{ d(T_{H_{i-1}}, x) - 1 : x \in X_{i-1}(c) \Big\} \Big) \\ &\leq |L(v_i)| + \sum \Big\{ d(T_{H_{i-1}}, x) - 1 : x \in V(T_{H_{i-1}}) \Big\} \\ &= |L(v_i)| + 2|E(T_{H_{i-1}})| - |V(T_{H_{i-1}})|. \end{aligned} \tag{4.4}$$

Since $T_{H_{i-1}}$ is a tree, $|E(T_{H_{i-1}})| = |V(T_{H_{i-1}})| - 1$. Furthermore, recall that $T_{H_{i-1}}$ is a spanning tree of H_{i-1}, and hence $V(T_{H_{i-1}}) = V(H_{i-1})$. By Eq. (4.4) we thus have

$$|V(\widehat{H}_i')| \leq |L(v_i)| + |V(H_{i-1})| - 2.$$

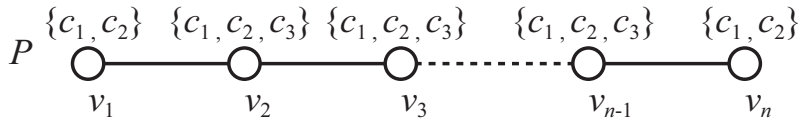


Figure 4.6: The path P with n vertices.

By Lemma 4.1 we have $|L(v_i)| \leq d(G, v_i) + 1$, and hence

$$|V(\widehat{H}'_i)| \leq |L(v_i)| + |V(H_{i-1})| - 2 \leq |V(H_{i-1})| + d(G, v_i) - 1,$$

as required. □

By Lemma 4.14 each encoding graph H_i is of size $O(n)$ for each $i \in \{1, 2, \ldots, n\}$. Therefore, our algorithm runs in polynomial time.

This completes the proof of Theorem 4.2.

We finally note that our estimation of the size of the encoding graph H_i is tight in some sense. For example, consider the path P with n vertices illustrated in Figure 4.6. Then, its encoding graph H_{n-1} is of size $\Omega(n)$.

Chapter 5 List Homomorphism Reconfiguration

In this chapter, we study the complexity of LHR.

5.1 PSPACE-completeness on paths

In this section, we show the following theorem.

Theorem 5.1 LHR *is PSPACE-complete even if* $k = O(1)$ *for paths.*

Proof. We give a polynomial-time reduction from R-WORD RECONFIGURATION, which is defined as follows. Let R be a (possibly non-simple) directed graph. An *R-word* (of *length* ρ) is a string $\boldsymbol{w} \in V(R)^\rho$ such that $w_i w_{i+1} \in E(R)$ for every pair of consecutive symbols w_i and w_{i+1} in $\boldsymbol{w}$; in other words, $\boldsymbol{w}$ can be seen as a directed walk in R. For an integer $\rho \geq 1$, an *R-word graph* $\mathscr{W}_\rho(R)$ is a graph such that $V(\mathscr{W}_\rho(R))$ is a set of all R-word of length ρ, and two R-words $\boldsymbol{w}$ and $\boldsymbol{w}'$ are adjacent if and only if the hamming distance between them is exactly one. Then, R-WORD RECONFIGURATION asks for a given integer ρ, two R-words $\boldsymbol{w}_s$ and $\boldsymbol{w}_t$, whether there exists a walk between $\boldsymbol{w}_s$ and $\boldsymbol{w}_t$ in $\mathscr{W}_\rho(R)$ or not. Wrochna showed that there exists a directed graph R such that R-WORD RECONFIGURATION is PSPACE-complete [60].

We now construct an instance $(G, D, \mathcal{C}, f_s, f_t)$ of LHR corresponding to an instance $(\rho, \boldsymbol{w}_s, \boldsymbol{w}_t)$ of R-WORD RECONFIGURATION. The idea is similar to the one used in the proof of the PSPACE-completeness of HR for cycles [60]. Let G be a path with ρ vertices $v_1, v_2, \ldots, v_\rho$. An underlying graph H is constructed as follows.

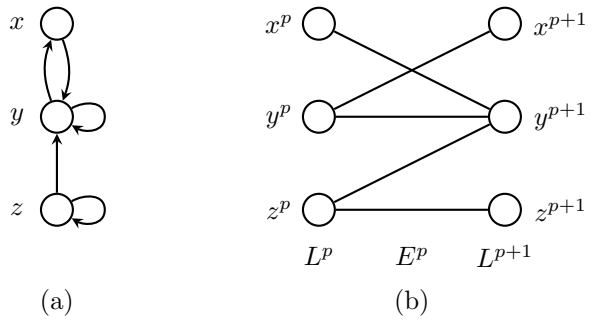


Figure 5.1: (a) A directed graph R and (b) the edge set E^p between L^p and L^{p+1}.

The vertex set $V(H)$ is a union of three sets L^0, L^1, L^2, where $L^p = \{x^p \colon x \in V(R)\}$ for each $p \in \{0, 1, 2\}$; each L^p is called a *layer*. For any integer $q > 2$, we define $x^q = x^p$ and $L^q = L^p$ if $q \equiv p \pmod 3$. The edge set $E(H)$ is a union of three sets E^0, E^1, E^2, where $E^p = \{x^p y^{p+1} \colon xy \in E(R)\}$ for each $p \in \{0, 1, 2\}$. (See Figure 5.1 for an example of E^p.) Let $D = V(H)$, and let $L(v_i) = L^i$ for each vertex $v_i \in V(G)$. For each edge $v_i v_{i+1} \in E(G)$, we construct the constraint $\mathcal{C}(v_i v_{i+1})$ so as to respect H, $L(v_i)$ and $L(v_{i+1})$. Finally, we define f_r for each $r \in \{s, t\}$ as follows. For each $v_i \in V(G)$, let $f_r(v_i) = w_i^p$ if w_i is the i-th symbol of $\boldsymbol{w}_r$. This completes the construction of $(G, D, \mathcal{C}, f_s, f_t)$, which can be done in polynomial time.

In order to show the correctness, we show that there exists a bijection between all solutions of $(G, D, \mathcal{C})$ and all R-words of length ρ. Let $f \colon V(G) \to D$ be any mapping which respects all lists such that $(f(v_1), f(v_2), \ldots, f(v_\rho)) = (w_1^1, w_2^2, \ldots, w_\rho^\rho)$ for some $w_1, w_2, \ldots, w_\rho \in V(R)$. We now define $\omega(f)$ as a string $w_1 w_2 \cdots w_\rho$. For each $i \in \{1, 2, \ldots, \rho - 1\}$, $f(v_i)f(v_{i+1}) = x^i y^{i+1} \in E(H)$ if and only if $w_i w_{i+1} = xy \in E(R)$. Therefore, f is a solution of $(G, D, \mathcal{C})$ if and only if $\omega(f)$ is an R-word of length ρ. Let ω' be the restriction of ϕ on $V(\mathscr{S}((G, D, \mathcal{C})))$, that is, $\omega' = \omega|_{V(\mathscr{S}((G,D,\mathcal{C})))}$. Then, by the definition of ω, ω' is a bijection between $V(\mathscr{S}((G, D, \mathcal{C})))$ and $V(\mathscr{W}_\rho(R))$. Moreover, this bijection preserves the adjacency relation of the solution graph, and $f_r = \omega'(\boldsymbol{w}_r)$, $r \in \{s, t\}$. Thus, $(G, D, \mathcal{C}, f_s, f_t)$ is a yes-instance if and only if $(\rho, \boldsymbol{w}_s, \boldsymbol{w}_t)$ is. □

This theorem implies that the polynomial-time algorithm for caterpillars in the previous chapter is unlikely to be extended to LHR, since a caterpillar has the pathwidth one.

5.2 A polynomial-time algorithm

In this section, we show the following theorem, which generalizes the known tractability results for LCR [17] and HR [59].

Theorem 5.2 LHR *can be solved in polynomial time if* $k = 3$.

Proof. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be a given instance of LHR such that H is the underlying graph with $|V(H)| \leq |D| = 3$ and L is a list assignment. We assume without loss of generality that G is connected and has at least two vertices. Since G is connected, for any homomorphism f from G to H, there exists exactly one connected component C such that $f(v) \in V(C)$ holds for every vertex $v \in V(G)$. Moreover, for any two homomorphisms f and f' from G to H which correspond to different connected components, $|\mathsf{dif}(f, f')| = |V(G)| \geq 2$ holds; and hence they are not adjacent in the solution graph. Because the relation of reconfigurability between homomorphisms is transitive, $C_s = C_t$ holds if f_s and f_t are reconfigurable, where C_s and C_t are connected components of H corresponding to f_s and f_t, respectively. Thus, we can assume that $C_s = C_t$, and let $H := C_s$ and $D := V(C_s)$. If H is complete, $\mathcal{I}$ is also an instance of LCR with $|D| \leq 3$, which is solvable in polynomial time [17]. Otherwise, H is a path $(\{1, 2, 3\}, \{12, 23\})$ of length two. Let V_s (resp. V_t) be the set of all vertices $v \in V(G)$ with $f_s(v) \in \{1, 3\}$ ($f_s(v) \in \{1, 3\}$). We now claim that $\mathcal{I}$ is a yes-instance if and only if $V_s = V_t$, which can be checked in polynomial time.

If $V_s = V_t$, then $\mathsf{dif}(f_s, f_t) \subseteq V_s$ holds. Because H contains no edge between 1 and 3 and f_s is a homomorphism from G to H, V_s must be an independent set of G.

Therefore, we can independently change the value of each vertex in $\mathsf{dif}(f_s, f_t) \subseteq V_s$ to obtain f_t; and hence $\mathcal{I}$ is a yes-instance.

We next assume that $\mathcal{I}$ is a yes-instance but $V_s \neq V_t$. Then, there exist two consecutive homomorphisms f and f' in the reconfiguration sequence such that $(f(v), f'(v)) \in \{(1,2),(2,1),(3,2),(2,3)\}$ holds, where v is the unique vertex in $\mathsf{dif}(f, f')$; that is, f' is obtained from f by changing the value of v along an edge of H. Since G is connected and has at least two vertices, v has at least one neighbor w in G. Because f is a homomorphism, $f(v) = 2$ if and only if $f(w) \neq 2$. Similarly, because f' is a homomorphism, $f'(v) = 2$ if and only if $f'(w) = f(w) \neq 2$. From the definition of v, $f(v) = 2$ if and only if $f'(v) \neq 2$. We thus have that $f(w) \neq 2$ if and only if $f(w) = 2$, which is a contradiction. Therefore, $V_s \neq V_t$ if $\mathcal{I}$ is a yes-instance.

□

Chapter 6 Binary Constraint Satisfiability Reconfiguration

In this chapter, we study the complexity of 2-CSR (BINARY CSR).

6.1 PSPACE-completeness

In contrast to Theorem 5.2, we show the following theorem in this section.

Theorem 6.1 2-CSR *is PSPACE-complete for bipartite planar graphs even if* $k = 3$.

Proof. We give a polynomial-time reduction from LCR to 2-CSR. It is known that LCR is PSPACE-complete for bipartite planar graphs even if each list has size at most three [4]. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be such an instance of LCR, and let L is a list assignment. Without loss of generality, we assume that $D = \{1, 2, 3, 4\}$. We then construct an instance $(G, \{1, 2, 3\}, \mathcal{C}', f'_s, f'_t)$ of 2-CSR as follows. The idea is to simply replace a value 4 with some value from $\{1, 2, 3\}$ for each vertex without changing the graph G. Let $v \in V(G)$ be a vertex such that $4 \in L(v)$. Since $|L(v)| \leq 3$, there exists a value i in $\{1, 2, 3\} \setminus L(v)$. Let $\pi \colon D \to D$ be a permutation such that $\pi(i) = 4$, $\pi(4) = i$ and $\pi(j) = j$ for each $j \notin \{i, 4\}$, and we update $(L, \mathcal{C}, f_s, f_t)$ as follows:

- $L(v) := L(v) \setminus \{4\} \cup \{i\}$;
- $g := (\pi(g(v)), g(w))$ for each neighbor w of v and each mapping $g \in \mathcal{C}(vw)$; and

- $f_r(v) := \pi(f_r(v))$ for each $r \in \{s, t\}$.

We repeat this operation until there is no vertex v such that $4 \in L(v)$, and let $\mathcal{I}' = (G, \{1, 2, 3\}, \mathcal{C}', f'_s, f'_t)$ be the resulting instance. Observe that the construction can be done in polynomial time.

Because we only replace values, $\mathcal{I}'$ is a valid instance of 2-CSR which is essentially equivalent to $\mathcal{I}$. Moreover, G is bipartite planar and the domain has only three values, and thus this completes the proof. □

We also have the following corollary.

Corollary 6.1 2-CSR *is PSPACE-complete even if* $k = 3$ *for complete graphs.*

Proof. Let $(G, D, \mathcal{C}, f_s, f_t)$ be an instance of 2-CSR constructed in Theorem 6.1. We then add an edge between every non-adjacent pair, and give a trivial constraint $\mathcal{C}(vw) = D^2$ to every added edge vw. Notice that this modification does not change the solution graph, and thus the reconfigurability. □

This implies that a fixed-parameter algorithm parameterized by $k + \mathsf{mw}$ is unlikely to exist, for a complete graph has the modular-width zero.

6.2 A polynomial-time algorithm

In contrast to Theorem 6.1, we show the following theorem in this section.

Theorem 6.2 2-CSR *can be solved in polynomial time if* $k = 2$.

Proof. We reduce the problem to BIJUNCTIVE BCSR, which is solvable in polynomial time [26]. BIJUNCTIVE BCSR is a special case of BCSR where $D = \{0, 1\}$ and there exists a 2-CNF formula $\phi(v_1, \ldots, v_r)$ such that $\mathcal{C}(\{v_1 \ldots, v_r\})$ is exactly the set of all satisfying assignments of ϕ for every hyperedge $\{v_1 \ldots, v_r\} \in E(G)$. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be a given instance of 2-CSR where G is a graph and $D = \{0, 1\}$. We now show that for every edge $vw \in E(G)$ there exists a 2-CNF

formula $\phi(v, w)$ such that $\mathcal{C}(vw)$ is exactly the set of all satisfying assignments of ϕ. For each $i \in D$ and $u \in \{v, w\}$, we denote by u^i a literal u if $i = 0$ or $\bar{u}$ if $i = 1$. Then we define a 2-CNF formula $\phi(v, w)$ as follows:

$$\phi(v, w) = \bigwedge_{(a,b) \in D^2 \setminus \mathcal{C}(vw)} (v^a \vee w^b).$$

Notice that a clause $(v^a \vee w^b)$ corresponds to a set $D^2 \setminus \{(a, b)\}$. Therefore, $\phi(v, w)$ corresponds to the set

$$\bigcap_{(a,b) \in D^2 \setminus \mathcal{C}(vw)} D^2 \setminus \{(a, b)\} = D^2 \setminus (D^2 \setminus \mathcal{C}(vw)) = \mathcal{C}(vw)$$

as required. □

Part II

Parameterized Complexity

Chapter 7 Homomorphism Reconfiguration and List Coloring Reconfiguration

In this chapter, we study the parameterized complexity of HR and LCR, and show that fixed-parameter algorithms are unlikely to exist for almost all graph parameters in Figure 1.4.

7.1 Homomorphism reconfiguration

In this section, we show the following theorem.

Theorem 7.1 HR *is* $W[1]$*-hard when parameterized by the number* n *of vertices in a graph.*

Proof. We give a parameterized reduction from LABELED CLIQUE RECONFIGURATION, which is defined as follows. Let G' be a simple graph and let τ be a positive integer. A *τ-labeled clique* (τ-*LC* for short) of G' is a vector $(u_1, u_2, \ldots, u_\tau)$ consisting of τ distinct vertices $u_1, u_2, \ldots, u_\tau \in V(G')$ which form a clique. A *τ-labeled clique graph* $\mathscr{C}_\tau(G')$ is a graph such that $V(\mathscr{C}_\tau(G'))$ is a set of all τ-LCs of G', and two τ-labeled cliques C and C' are adjacent if and only if they differ on exactly one component. Then, LABELED CLIQUE RECONFIGURATION asks for a given graph G', an integer τ, two τ-LCs C_s and C_t, whether there exists a walk between C_s and C_t in $\mathscr{C}_\tau(G')$ or not. It is known that LABELED CLIQUE RECONFIGURATION is $W[1]$-hard

when parameterized by τ [35].[1] We now construct an instance $(G, D, \mathcal{C}, f_s, f_t)$ of HR corresponding to an instance (G', τ, C_s, C_t) of LABELED CLIQUE RECONFIGURATION. Let G be a complete graph K_τ with τ vertices $\{v_1, v_2, \ldots, v_\tau\}$, and let $D = V(G')$. We define constraints for edges so that G' is an underlying graph; that is, for each $v_iv_j \in E(G)$, we define $\mathcal{C}(v_iv_j) = \{(u_p, u_q) \colon u_pu_q \in E(G')\}$. Observe that $(G, D, \mathcal{C})$ is an instance of HOMOMORPHISM with τ vertices. The remaining components, two solutions f_s and f_t, are defined as follows. For any τ-LC $C = (u_1, u_2, \ldots, u_\tau)$ of G', we define ϕ_C be a mapping such that $\phi_C(v_i) = u_i$ for each $i \in \{1, 2, \ldots, \tau\}$. Since $\phi_C(v_i)\phi_C(v_j) = u_iu_j \in E(G')$ holds for each distinct $i, j \in \{1, 2, \ldots, \tau\}$, ϕ_C is a solution of $(G, D, \mathcal{C})$. Thus, let $f_s = \phi_{C_s}$ and $f_t = \phi_{C_t}$. This completes the construction of $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$. Finally, (G', τ, C_s, C_t) is a yes-instance if and only if $\mathcal{I}$ is, because $\mathscr{C}_\tau(G')$ and $\mathscr{S}(\mathcal{I})$ are isomorphic under a mapping ϕ. □

The following also follows as a corollary.

Corollary 7.1 HR *is $W[1]$-hard when parameterized by p, where p is any parameter which is polynomially bounded in the number n of vertices in a graph.*

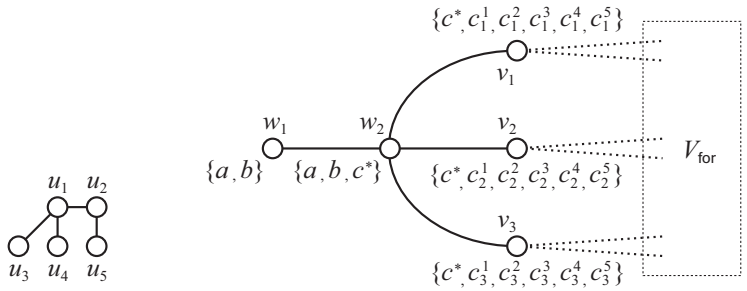
7.2 List coloring reconfiguration

In this section, we show the following theorem.

Theorem 7.2 LIST COLORING RECONFIGURATION *is $W[1]$-hard when parameterized by* vc*, where* vc *is an upper bound on the size of a minimum vertex cover of an input graph.*

Recall that LIST COLORING RECONFIGURATION is PSPACE-complete even for a fixed constant $k \geq 4$. Therefore, the problem is intractable if we take only one parameter, either k or vc.

[1] Although they actually show the similar result for (unlabeled) INDEPENDENT SET RECONFIGURATION, the proof can be applied to LABELED CLIQUE RECONFIGURATION.



(a) H , $s = 3$ (b) G , L

Figure 7.1: (a) An instance H of INDEPENDENT SET, and (b) the graph G and the list L. The set V_{for} contains vertices of $(i, j; p, q)$-forbidding gadgets for all $(i, j) \in \{(1, 2), (1, 3), (2, 3)\}$ and all $(p, q) \in \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 2), (2, 1), (1, 3), (3, 1), (1, 4), (4, 1), (2, 5), (5, 2)\}$; thus $|V_{\mathsf{for}}| = 39$.

In order to prove Theorem 7.2, we give a parametrized reduction from the INDEPENDENT SET problem when parameterized by the solution size s, in which we are asked whether a given graph H has an independent set of size at least s. This problem is known to be $W[1]$-hard [22].

7.2.1 Construction

Let H be a graph with n vertices $u_1, u_2, \ldots, u_n$, and s be an integer as the solution size parameter for INDEPENDENT SET. Then, we construct the corresponding instance (G, L, f_s, f_t) of LIST COLORING RECONFIGURATION with the parameter vc as follows. (See also Figure 7.1.)

We first create s vertices $v_1, v_2, \ldots, v_s$, which are called *selection vertices*; let V_{sel} be the set of all selection vertices. For each $i \in \{1, 2, \ldots, s\}$, we set $L(v_i) = \{c^*, c_i^1, c_i^2, \ldots, c_i^n\}$. In our reduction, we will construct G and L so that assigning the color c_i^p, $p \in \{1, 2, \ldots, n\}$, to $v_i \in V_{\mathsf{sel}}$ corresponds to choosing the vertex $u_p \in V(H)$ as a vertex in an independent set of H. Then, in order to make a correspondence between a color assignment to V_{sel} and an independent set of size s in H, we need to construct the following properties:

- For each $p, q \in \{1, 2, \ldots, n\}$ with $u_p u_q \in E(H)$, we use at most one color from $\{c_1^p, c_2^p, \ldots, c_s^p, c_1^q, c_2^q, \ldots, c_s^q\}$. This property ensures the independence of chosen vertices from H, that is, no two adjacent vertices in H are chosen in an independent set.

- For each $p \in \{1, 2, \ldots, n\}$, we use at most one color from $\{c_1^p, c_2^p, \ldots, c_s^p\}$. This property will be used to ensure the size of the constructed independent set of H, that is, each vertex $u_p \in V(H)$ can be chosen at most once in an independent set.

To do this, we define an $(i, j; p, q)$-forbidding gadget for $i, j \in \{1, 2, \ldots s\}$ and $p, q \in \{1, 2, \ldots, n\}$. The $(i, j; p, q)$-*forbidding gadget* is a vertex w which is adjacent to v_i and v_j and has a list $L(w) = \{c_i^p, c_j^q\}$. Observe that the vertex w forbids that v_i and v_j are simultaneously colored with c_i^p and c_j^q, respectively. In order to satisfy the desired properties above, we now add our gadgets as follows: for all $i, j \in \{1, 2, \ldots s\}$ with $i < j$,

- add $(i, j; p, q)$- and $(i, j; q, p)$-forbidding gadgets for every edge $u_p u_q \in E(H)$; and

- add an $(i, j; p, p)$-forbidding gadget for every vertex $u_p \in V(H)$.

We denote by V_{for} the set of all vertices in the forbidding gadgets. We create an edge consisting of two vertices w_1 and w_2 such that $L(w_1) = \{a, b\}$ and $L(w_2) = \{a, b, c^*\}$, and connect w_2 with all selection vertices in V_{sel}.

Finally, we construct two L-colorings f_s and f_t of G as follows:

- for each $v_i \in V_{\mathsf{sel}}$, $f_s(v_i) = f_t(v_i) = c^*$;

- for each $w \in V_{\mathsf{for}}$, $f_s(w)$ and $f_t(w)$ are arbitrary chosen colors from $L(w)$; and

- $f_s(w_1) = f_t(w_2) = a$, and $f_t(w_1) = f_s(w_2) = b$.

Note that both f_s and f_t are proper L-colorings of G. This completes the construction of (G, L, f_s, f_t).

7.2.2 Correctness of the reduction

In this subsection, we prove the following three statements:

- (G, L, f_s, f_t) can be constructed in time polynomial in the size of H.
- The upper bound vc on the size of a minimum vertex cover of G depends only on s.
- H is a yes-instance of INDEPENDENT SET if and only if (G, L, f_s, f_t) is a yes-instance of LCR.

In order to prove the first statement, it suffices to show that the size of (G, L, f_s, f_t) is bounded polynomially in $n = |V(H)|$. From the construction, we have $|V(G)| = |V_{\mathsf{sel}}| + |V_{\mathsf{for}}| + |\{w_1, w_2\}| \leq s + s^2 \times (|V(H)| + 2|E(H)|) + 2 = O(n^4)$. In addition, each list contains $O(n)$ colors. Therefore, the construction can be done in time polynomial in n.

The second statement immediately follows from the fact that $\{w_2\} \cup V_{\mathsf{sel}}$ is a vertex cover in G of size $s + 1$; observe that $G \setminus (\{w_2\} \cup V_{\mathsf{sel}}) = G[\{w_1\} \cup V_{\mathsf{for}}]$ contains no edge.

Finally, we prove the third statement as follows.

Lemma 7.1 *H is a yes-instance of* INDEPENDENT SET *if and only if (G, L, f_s, f_t) is a yes-instance of* LCR*.*

Proof. We first prove the if direction. Assume that there exists a reconfiguration sequence $\mathcal{W}$ for (G, L, f_s, f_t). Then, $\mathcal{W}$ must contain at least one L-coloring f such that $f(w_2) = c^*$ in order to recolor w_1 from a to b. Since w_2 is adjacent to all vertices

in V_{sel}, $f(v_i) \neq c^*$ holds for every $v_i \in V_{\mathsf{sel}}$. Then, by the construction, the vertex set $\{u_p \in V(H) \colon c_i^p = f(v_i), v_i \in V_{\mathsf{sel}}\}$ forms an independent set in H of size $|V_{\mathsf{sel}}| = s$.

We then prove the only-if direction. We construct a reconfiguration sequence for (G, L, f_s, f_t) which passes through two L-colorings f'_s and f'_t defined as follows.

We first define f'_s, and show that f_s and f'_s are reconfigurable. From the assumption, H has an independent set I of size s, say, $I = \{u_1, u_2, \ldots, u_s\}$. Then, we define f'_s as follows:

- for each $v_i \in V_{\mathsf{sel}}$, $f'_s(v_i) = c_i^i$;
- for each $(i, j; p, q)$-forbidding vertex $w \in V_{\mathsf{for}}$, $f'_s(w)$ is an arbitrary chosen color from $L(w) \setminus \{c_i^i, c_j^j\}$; and
- $f'_s(w_1) = f_s(w_1) = a$ and $f'_s(w_2) = f_s(w_2) = b$.

Note that f'_s is a proper L-coloring of G. We now show that f_s and f'_s are reconfigurable. We first recolor all vertices $w \in V_{\mathsf{for}}$ to the colors $f'_s(w)$ $(\neq c^*)$ in an arbitrary order. These recoloring steps can be done, since $f_s(v_i) = c^*$ for all $v_i \in V_{\mathsf{sel}}$ and V_{for} is an independent set in G. We then recolor all vertices $v_i \in V_{\mathsf{sel}}$ to the colors $f'_s(v_i)$ in an arbitrary order. These recoloring steps also can be done, since f'_s is a proper L-coloring and V_{sel} is an independent set in G. Thus, f_s and f'_s are reconfigurable.

We then define f'_t, and show that f'_t and f_t are reconfigurable. Similarly as f'_0, we define f'_t as follows:

- for each $v_i \in V_{\mathsf{sel}}$, $f'_t(v_i) = c_i^i$;
- for each $(i, j; p, q)$-forbidding vertex $w \in V_{\mathsf{for}}$, $f'_t(w)$ is an arbitrary chosen color from $L(w) \setminus \{c_i^i, c_j^j\}$; and
- $f'_t(w_1) = f_t(w_1) = b$ and $f'_t(w_2) = f_t(w_2) = a$.

Then, the similar arguments above yield that f'_t and f_t are reconfigurable.

Finally, we prove that f'_s and f'_t are reconfigurable. Recall that $f'_s(w_1) = f'_t(w_2) = a$, $f'_t(w_1) = f'_s(w_2) = b$, and $f'_s(v_i) = f'_t(v_i) \neq c^*$ for all $v_i \in V_{\mathsf{sel}}$. Then, we can swap the colors a and b by the following three steps:

- recolor w_2 from b to c^*;
- recolor w_1 from a to b; and
- recolor w_2 from c^* to a.

After this color swap, we can recolor all vertices $w \in V_{\mathsf{for}}$ to the colors $f'_t(w)$ in the arbitrary order, since V_{for} is an independent in G.

In this way, we can construct a reconfiguration sequence for (G, L, f_s, f_t) which passes through two L-colorings f'_s and f'_t, and hence (G, L, f_s, f_t) is a yes-instance of LCR. □

This completes the proof of Theorem 7.2.

Chapter 8 List Homomorphism Reconfiguration

In this chapter, we show the following theorem.

Theorem 8.1 LHR *is fixed-parameter tractable when parameterized by* $k + \mathsf{mw}$.

Since any cograph has the modular-width zero, we also have the following corollary.

Corollary 8.1 LHR *is fixed-parameter tractable for cographs when parameterized by* k.

8.1 Reduction rule

In order to prove several theorems including Theorem 8.1, we give fixed-parameter algorithms which are based on the concept of "kernelization". That is, we compute from the given instance into another instance whose size depends only on the parameter. After that, we can solve the problem by using Theorem 2.1.

In this subsection, we show some useful lemma, which compresses an input hypergraph into a smaller hypergraph with keeping the reconfigurability. The main idea is to "identify" two subgraphs which behave in the same way with respect to the reconfigurability.

We now formally characterize such subhypergraphs and explain how to identify them. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be an instance of CSR. For each vertex $v \in V(G)$, we define $\mathcal{A}(v)$ as a pair $(f_s(v), f_t(v))$ consisting of the initial and the target value assignments of v. Let V_1 and V_2 be two non-empty vertex subsets of G such that

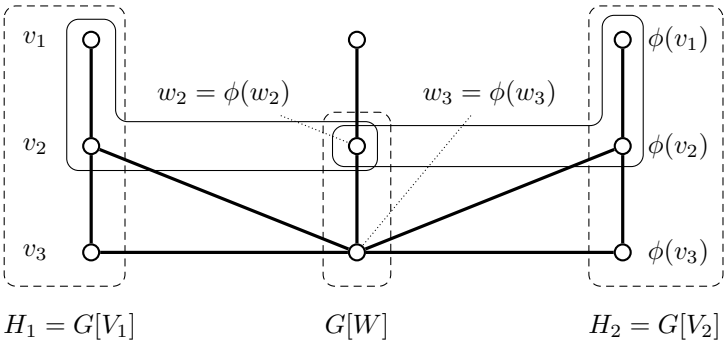


Figure 8.1: An example of two subhypergraphs H_1 and H_2 of G which satisfies the conditions (1) and (2). We draw each hyperedge of size two as a solid line, and omit the bijection $\pi\colon E(H_1') \to E(H_2')$ since it is uniquely defined from $\phi\colon V(H_1') \to V(H_2')$. If $\mathcal{A}$ and $\mathcal{C}$ satisfy the conditions (3) and (4), H_1 and H_2 are identical.

$|V_1| = |V_2|$, and $V_1 \cap V_2 = \emptyset$. Assume that $N(G, V_1) = N(G, V_2) = W$. Let $H_1 = G[V_1]$, $H_2 = G[V_2]$, $H_1' = G[V_1 \cup W]$ and $H_2' = G[V_2 \cup W]$.

Definition 8.1 *Two induced subhypergraphs H_1 and H_2 are* identical *if there exist two bijections $\phi\colon V(H_1') \to V(H_2')$ and $\pi\colon E(H_1') \to E(H_2')$ which satisfy the following four conditions:*

(1) H_1' and H_2' are isomorphic under ϕ and π.

(2) for every vertex $v \in W$, $\phi(v) = v$;

(3) for every vertex $v \in V_1$, $\mathcal{A}(v) = \mathcal{A}(\phi(v))$, that is, $f_s(v) = f_s(\phi(v))$ and $f_t(v) = f_t(\phi(v))$; and

(4) for every hyperedge $X \in E(H_1)$, $\mathcal{C}(\pi(X)) = \mathcal{C}[\widehat{\phi}](X)$, where $\widehat{\phi} = \phi|_X$.

See Figure 8.1 for an example.

We next define another instance $\mathcal{I}' = (G', D, \mathcal{C}', f_s', f_t')$ as follows:

- $G' = G \setminus V_2$;
- $f_s' = f_s|_{V(G')}$ and $f_s' = f_s|_{V(G')}$; and

- for each $X' \in E(G')$, $\mathcal{C}'(X') = \bigcap_{X \in E'} \mathcal{G}(X)$, where $E' = \{X \in E(G) \colon X \setminus V_2 = X'\}$ and $\mathcal{G}(X) = \{g|_{X'} \colon g \in \mathcal{C}(X)\}$.

Intuitively, $\mathcal{I}'$ is obtained by restricting all components (hypergraphs, mappings in constraints, and two solutions) of $\mathcal{I}$ on $V(G) \setminus V_2$. We say that $\mathcal{I}'$ is obtained from $\mathcal{I}$ by *identifying* H_1 with H_2.

Then, we have the following lemma, which says that $\mathcal{I}$ and $\mathcal{I}'$ are equivalent with respect to the feasibility.

Lemma 8.1 *Let $f' \colon V(G') \to D$ be a mapping from $V(G')$ to D. Then, f' is a solution for $(G', D, \mathcal{C}')$ if and only if there exists a solution f for $(G, D, \mathcal{C})$ such that $f' = f|_{V(G')}$.*

Proof. We first prove the if direction. Assume that f is a solution for $(G, D, \mathcal{C})$. In order to show that $f' = f|_{V(G')}$ is a solution for $(G', D, \mathcal{C}')$, it suffices to check that f' satisfies all constraints. For each $X' \in E(G')$, consider the hyperedge set $E' = \{X \in E(G) \colon X \setminus V_2 = X'\}$. Recall that $\mathcal{C}'(X') = \bigcap_{X \in E'} \mathcal{G}(X)$, where $\mathcal{G}(X) = \{g|_{X'} \colon g \in \mathcal{C}(X)\}$. For every hyperedge $X \in E'$, $f|_X \in \mathcal{C}(X)$ holds since f is a solution for $(G, D, \mathcal{C})$. Notice that $f'|_{X'} = (f|_X)|_{X'}$, and hence it is contained in $\mathcal{G}(X)$. Therefore, $f'|_{X'} \in \mathcal{C}'(X')$ holds, and hence f' is a solution for $(G', D, \mathcal{C}')$.

We next prove the only-if direction. Assume that f' is a solution for $(G', D, \mathcal{C}')$. We claim that a mapping $f \colon V(G) \to D$ obtained by extending f' as follows is a solution for $(G, D, \mathcal{C})$:

$$f(v) = \begin{cases} f'(\phi^{-1}(v)) & \text{if } v \in V_2; \\ f'(v) & \text{otherwise.} \end{cases}$$

To this end, we show that for each hyperedge $X \in E(G)$, $f|_X \in \mathcal{C}(X)$ holds. Briefly, this follows the conditions (2) and (4) of Definition 8.1. If $X \in E(G')$, $\mathcal{C}'(X)$ contains $f'|_X = f|_X$. Because $\mathcal{C}'(X)$ is a subset of $\mathcal{C}(X)$, $\mathcal{C}(X)$ also contains $f|_X$. Otherwise, X is a hyperedge in $H'_2 = G[V_2 \cup W]$. By the condition (4) of

Definition 8.1, $\mathcal{C}(X) = \mathcal{C}[\widehat{\phi}](X')$ holds, where $\widehat{\phi} = \phi|_{X'}$ and $\pi(X') = X$. Because W, V_1 and V_2 are disjoint each other, $X' \subseteq V_1 \cup W$ does not intersect V_2. Therefore, X' is contained as a hyperedge in G', and hence f' satisfies the constraint of X'; that is, $f'|_{X'} \in \mathcal{C}(X')$. Recall that $\mathcal{C}[\widehat{\phi}](X') = \{g \circ \widehat{\phi}^{-1} \colon g \in \mathcal{C}(X')\}$. Thus, $(f'|_{X'}) \circ \widehat{\phi}^{-1}$ is in $\mathcal{C}(X) = \mathcal{C}[\widehat{\phi}](X')$. It now suffices to show that $(f'|_{X'}) \circ \widehat{\phi}^{-1} = f|_X$. From the definition of f, $f(v) = f'(\phi^{-1}(v))$ holds for each $v \in V_2 \cap X$, and $f(v) = f'(v)$ holds or each $v \in W \cap X$. In addition, by the condition (2) of Definition 8.1, $f(v) = f'(\phi^{-1}(v))$ also holds in the later case; and hence $f|_X = (f' \circ \phi^{-1})|_X$. Since $\widehat{\phi}^{-1} = (\phi|_{X'})^{-1} = \phi^{-1}|_X$ is a bijection from X to X', we have $f|_X = (f' \circ \phi^{-1})|_X = (f'|_{X'}) \circ (\phi^{-1}|_X) = (f'|_{X'}) \circ \widehat{\phi}^{-1}$ as required. □

Notice that Lemma 8.1 ensures that the "restricted" instance $\mathcal{I}'$ is a valid instance of CSR.

We now give the following key lemma, which says that $\mathcal{I}$ and $\mathcal{I}'$ are equivalent with respect to even the reconfigurability.

Lemma 8.2 (Reduction rule) *Let $\mathcal{I}$ and $\mathcal{I}'$ be instances of* CSR *defined as above. Then, $\mathcal{I}'$ is a yes-instance if and only if $\mathcal{I}$ is.*

Proof. We first prove the if direction. Suppose that $\mathcal{I}$ is a yes-instance. Then, there exists a reconfiguration sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ for $\mathcal{I}$, where $f_\ell = f_t$. For each $i \in \{0, 1, \ldots, \ell\}$, $f_i|_{V(G')}$ is a solution for $(G', D, \mathcal{C}')$ by Lemma 8.1. Therefore, by removing all duplicate solutions appearing consecutively in the resulting sequence, $\langle f_0|_{V(G')}, f_1|_{V(G')}, \ldots, f_{\ell'}|_{V(G')} \rangle$ is a reconfiguration sequence for $\mathcal{I}'$. Thus $\mathcal{I}'$ is a yes-instance.

We now prove the only-if direction. Suppose that $\mathcal{I}'$ is a yes-instance. Then, there exists a reconfiguration sequence $\mathcal{W}' = \langle f'_0, f'_1, \ldots, f'_\ell \rangle$ for $\mathcal{I}'$ with $f'_0 = f_s|_{V(G')}$ and $f'_\ell = f_t|_{V(G')}$. Our goal is to construct a reconfiguration sequence $\mathcal{W}$ for $\mathcal{I}$ from $\mathcal{W}'$.

For each $i \in \{0, 1, \ldots, \ell\}$, we first extend f'_i to f_i as follows:

$$f_i(v) = \begin{cases} f'_i(\phi^{-1}(v)) & \text{if } v \in V(H_2); \\ f'_i(v) & \text{otherwise.} \end{cases}$$

Notice that f_i corresponds to the mapping defined in the only-if proof of Lemma 8.1, and hence it is a solution for $(G, D, \mathcal{C})$. Therefore, $\langle f_0, f_1, \ldots, f_\ell \rangle$ is a sequence of solutions for $(G, D, \mathcal{C})$. However, there may exist several indices $i \in \{0, 1, \ldots, \ell - 1\}$ such that f_i and f_{i+1} are not adjacent, that is, $|\mathsf{dif}(f_i, f_{i+1})| > 1$ may hold. Recall that f'_i and f'_{i+1} are adjacent for each $i \in \{0, 1, \ldots, \ell - 1\}$, that is, $\mathsf{dif}(f'_i, f'_{i+1}) = \{w\}$ for some vertex $w \in V(G')$. Therefore we know that

- if $w \notin V(H_1)$, then $\mathsf{dif}(f_i, f_{i+1}) = \{w\}$ holds, that is, f_i and f_{i+1} are adjacent; and
- if $w \in V(H_1)$, then $\mathsf{dif}(f_i, f_{i+1}) = \{w, \phi(w)\}$ holds, that is, f_i and f_{i+1} are not adjacent.

In the latter case, between f_i and f_{i+1}, we insert a solution $\widetilde{f}_i$ of G defined as follows:

$$\widetilde{f}_i(v) = \begin{cases} f_{i+1}(v) & \text{if } v = w; \\ f_i(v) & \text{if } v = \phi(w); \\ f_i(v) & \text{otherwise.} \end{cases}$$

Observe that $\widetilde{f}_i$ is a solution for $(G, D, \mathcal{C})$. Moreover, both $\mathsf{dif}(f_i, \widetilde{f}_i) = \{w\}$ and $\mathsf{dif}(\widetilde{f}_i, f_{i+1}) = \{\phi(w)\}$ hold. Thus, we obtain a proper reconfiguration sequence $\mathcal{W}$ for $\mathcal{I}$ as claimed. □

8.2 Modified reduction rule

Observe that an instance obtained by applying Lemma 8.2 is of CSR in general. Therefore, in this section, we modify the reduction rule so that it always produces an instance of LHR from one of LHR.

To this end, we first give a simpler characterization of identical subgraphs. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be a given instance of LHR such that H is the underlying graph

and L is a list assignment. Let H_1 and H_2 are two induced subgraphs such that $|V(H_1)| = |V(H_2)|$, $V(H_1) \cap V(H_2) = \emptyset$.

Definition 8.2 *H_1 and H_2 are* LHR-identical *if there exists a bijection $\phi\colon V(H_1) \to V(H_2)$ which satisfies the following two conditions:*

(i) $vw \in E(H_1)$ if and only if $\phi(v)\phi(w) \in E(H_2)$.

(ii) for every vertex $v \in V_1$,

a $N(G, v) \setminus V(H_1) = N(G, \phi(v)) \setminus V(H_2)$;

b $\mathcal{A}(v) = \mathcal{A}(\phi(v))$, that is, $f_s(v) = f_s(\phi(v))$ and $f_t(v) = f_t(\phi(v))$; and

c $L(v) = L(\phi(v))$.

Then, we have the following lemma.

Lemma 8.3 *H_1 and H_2 are identical if they are* LHR*-identical for a bijection ϕ.*

Proof. We first prove that the assumption that $N(G, V(H_1)) = N(G, V(H_2))$ is satisfied. By the condition (ii)-a, we have $N(G, V(H_1)) = \bigcup_{v \in V(H_1)} (N(G, v) \setminus V(H_1)) = \bigcup_{v \in V(H_1)} (N(G, \phi(v)) \setminus V(H_2))$. Since ϕ is a bijection, $\bigcup_{v \in V(H_1)} (N(G, \phi(v)) \setminus V(H_2)) = \bigcup_{v \in V(H_2)} (N(G, v) \setminus V(H_2)) = N(G, V(H_2))$, as required.

Let $W = N(G, V(H_1)) = N(G, V(H_2))$, $H_1' = G[V(H_1) \cup W]$ and $H_2' = G[V(H_2) \cup W]$. In order to prove the lemma, it suffices to give two bijections which satisfy all conditions of Definition 8.1. We define two mappings ϕ' and π' as follows.

- For each $v \in V(H_1')$, $\phi'(v) = \phi(v)$ if $v \in V(H_1)$ and $\phi'(v) = v$ if $v \in W$.
- For each $vw \in E(H_1')$, $\pi'(vw) = \phi'(v)\phi'(w)$.

We now prove the condition (1) for ϕ' and π'. Since ϕ is a bijection between $V(H_1)$ and $V(H_2)$, ϕ' is a bijection between $V(H_1')$ and $V(H_2')$. By the definition of π', we suffice to show that π' is a bijection between $E(H_1')$ and $E(H_2')$. From the

definition and the condition (i), $\pi'|_{E(G[W])}$ is a bijection (the identity mapping) between $E(G[W])$ and $E(G[W])$ and $\pi'|_{E(H_1)}$ is a bijection between $E(H_1)$ and $E(H_2)$. Let $E'_1 = E(H'_1)\setminus(E(H_1)\cup E(G[W]))$ and $E'_2 = E(H'_2)\setminus(E(H_2)\cup E(G[W]))$. Then, it suffices to show that $\pi'|_{E'_1}$ is a bijection between E'_1 and E'_2. For each vertices $v \in V(H_1)$ and $w \in W$, $\phi'(v)\phi'(w) = \phi(v)w$ holds. From the condition (ii)-a, $vw \in E'_1$ if and only if $\phi(v)w \in E'_2$. Therefore, $\pi'|_{E'_1}$ is a bijection between E'_1 and E'_2, and hence π' is a bijection between $E(H'_1)$ and $E(H'_2)$.

The conditions (2) and (3) directly follows the definition of ϕ' and the condition (ii)-b.

We finally prove the condition (4). Recall that each constraint $\mathcal{C}(vw) = \{g \in D^{\{v,w\}} \colon g(v)g(w) \in E(H) \cap (L(v) \times L(w))\}$ depends only on $L(v)$ and $L(w)$. From the condition (ii)-c, for every $vw \in E(H_1)$, $L(v) \times L(w) = L(\phi'(v)) \times L(\phi'(w))$ holds. Therefore, $\mathcal{C}(vw)$ and $\mathcal{C}(\pi'(vw))$ are the same when we see them as the sets of vectors. □

By the above lemma, we can apply Lemma 8.2 to obtain a new instance $\mathcal{I}' = (G', D, \mathcal{C}', f'_s, f'_t)$. However, G' may contain hyperedges of size one and it is unclear that the constraint $\mathcal{C}'$ corresponds to HOMOMORPISML. Let $\mathcal{I}^* = (G^*, D, \mathcal{C}'|_{E(G^*)}, f'_s, f'_t)$ be an instance obtained by removing all size-one hyperedges from $\mathcal{I}'$. We then have the following two lemmas.

Lemma 8.4 *$\mathcal{I}^*$ is a valid instance of* LHR*, where H is an underlying graph and $L|_{V(G^*)}$ is a list assignment.*

Proof. First of all, f'_s and f'_t are solutions, because $\mathcal{I}'$ is a valid instance of 2-CSR and all constraints in $\mathcal{I}^*$ is contained in $\mathcal{I}'$. We next show that the constraint $\mathcal{C}'|_{E(G^*)}$ corresponds to HOMOMORPISML. Since G^* is an induced subgraph of G, each edge $vw \in E(G^*)$ is also contained in G. From the definition of $\mathcal{I}'$, $\mathcal{C}(vw) = \mathcal{C}'(vw)$ holds. Moreover, $\mathcal{C}(vw) = E(H)\cap(L(v)\times L(w)) = E(H)\cap(L|_{V(G^*)}(v)\times L|_{V(G^*)}(w))$. Therefore, $\mathcal{I}^*$ is a valid instance of LHR, where H is an underlying graph and $L|_{V(G^*)}$

is a list assignment. □

Lemma 8.5 *$\mathcal{I}^*$ is a yes-instance if and only if $\mathcal{I}'$ is.*

Proof. In order to prove the lemma, it suffices to show that a mapping $f\colon V(G') \to D$ is a solution for $\mathcal{I}^*$ if and only if it is a solution for $\mathcal{I}'$. If direction is trivial; as we have seen in the proof of the previous lemma. Thus, we assume that f is a solution for $\mathcal{I}^*$. Let $X \in E(G')$ be a hyperedge. If $|X| = 2$, X is an edge in G^*, and hence $\mathcal{C}(X) = \mathcal{C}'|_{E(G^*)}(X)$. Therefore f satisfies $\mathcal{C}(X)$ by the assumption. If $|X| = 1$, say $X = \{v\}$, the set $E' = \{vw \in E(G)\colon w \in V(H_2)\}$ is non-empty. Recall that $\mathcal{C}'(\{v\}) = \bigcap_{vw \in E'} \mathcal{G}(vw)$, where $\mathcal{G}(vw) = \{g|_{\{v\}}\colon g \in \mathcal{C}(vw)\}$. Let $vw \in E'$ be any edge. Because H_1 and H_2 are LHR-identical (and hence identical) for a bijection ϕ, $\mathcal{C}(vw) = \mathcal{C}[\widehat{\phi}](vw')$ holds, where $\widehat{\phi} = \phi|_{\{v,w\}}$ and $w' = \widehat{\phi}^{-1}(w)$. Since f satisfies $\mathcal{C}'|_{E(G^*)}(vw') = \mathcal{C}(vw')$, $f \circ \phi^{-1}|_{\{v,w\}}$ is in $\mathcal{C}(vw)$. Observe that $(f \circ \phi^{-1}|_{\{v,w\}})|_{\{v\}} = f|_{\{v\}}$ is contained in $\mathcal{G}(vw)$. Therefore, $f|_{\{v\}}$ is also contained in $\mathcal{C}'(\{v\})$, and hence f is a solution for $\mathcal{I}'$. □

In this way, we have the following modified reduction rule.

Lemma 8.6 (Modified reduction rule) *Let $\mathcal{I}$ and $\mathcal{I}^*$ be instances of* LHR *defined as above. Then, $\mathcal{I}^*$ is a yes-instance if and only if $\mathcal{I}$ is.*

8.3 Kernelization

Let $\mathcal{I} = (G, \mathcal{C}, f_s, f_t)$ be an instance of LHR such that H is the underlying graph and L is a list assignment.. Suppose that $|D| \leq k$, G is a connected graph with $\mathsf{pmw}(G) \leq \mathsf{pmw}$, and all vertices of G are totally ordered according to an arbitrary binary relation $\prec$.

8.3.1 Sufficient condition for identical subgraphs

We first give a sufficient condition for which two nodes in a PMD-tree $\mathsf{PMD}(G)$ for G correspond to identical subgraphs. Let $x \in V(\mathsf{PMD}(G))$ be a node, let $p := |V(\mathsf{CG}(x))|$, and assume that all vertices in $V(\mathsf{CG}(x))$ are labeled as $v_1, v_2, \ldots, v_p$ according to $\prec$; that is, $v_i \prec v_j$ holds for each i, j with $1 \leq i < j \leq p$. Let $m \geq p$ be some integer which will be defined later. We now define an $(m+1) \times m$ matrix $\mathcal{M}_m(x)$ as follows:

$$(\mathcal{M}_m(x))_{i,j} = \begin{cases} 1 & \text{if } i,j \leq p \text{ and } v_iv_j \in E(\mathsf{CG}(x)); \\ 0 & \text{if } i,j \leq p \text{ and } v_iv_j \notin E(\mathsf{CG}(x)); \\ 0 & \text{if } p < i \leq m \text{ or } p < j \leq m; \\ \mathcal{A}(v_j) & \text{if } i = m+1 \text{ and } j \leq p; \\ \emptyset & \text{otherwise,} \end{cases}$$

where $(\mathcal{M}_m(x))_{i,j}$ denotes an (i,j)-element of $\mathcal{M}_m(x)$. Notice that $\mathcal{M}_m(x)$ contains the adjacency matrix of $\mathsf{CG}(x)$ at its upper left $p \times p$ submatrix, and the bottommost row represents the vertex assignments of the vertices in $V(\mathsf{CG}(x))$. We call $\mathcal{M}_m(x)$ an *m-ID-matrix* of x. For example, consider the node x_{13} in Figure 2.12(a). Then, $p = 2$, and a 4-ID-matrix of x_{13} is as follows:

$$\mathcal{M}_4(x_{13}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathcal{A}(x_3) & \mathcal{A}(x_4) & \emptyset & \emptyset \end{bmatrix}.$$

Lemma 8.7 *Let y_1 and y_2 be two children of a parallel node x in $\mathsf{PMD}(G)$, and let m be an integer with $m \geq \max\{|V(\mathsf{CG}(y_1))|, |V(\mathsf{CG}(y_2))|\}$. If $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$ holds, then $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$ are identical.*

Proof. Let $p_1 := |V(\mathsf{CG}(y_1))|$ and $p_2 := |V(\mathsf{CG}(y_2))|$. Observe that $(\mathcal{M}_m(y_1))_{m+1,j} \neq \emptyset$ if and only if $j \leq p_1$, and $(\mathcal{M}_m(y_2))_{m+1,j} \neq \emptyset$ if and only if $j \leq p_2$. By the assumption that $(\mathcal{M}_m(y_1))_{m+1,j} = (\mathcal{M}_m(y_2))_{m+1,j}$ for all $j \in \{1, 2, \ldots, m\}$, we have $p_1 = p_2$; we denote by p this value.

We now check that $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$ are identical. The condition 1 of identical subgraphs holds, because the upper left $p \times p$ submatrices in $\mathcal{M}_m(y_1)$ and $\mathcal{M}_m(y_2)$ correspond to the adjacency matrices of $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$, respectively. The condition 2(b) holds, because the bottommost rows are the same in $\mathcal{M}_m(y_1)$ and $\mathcal{M}_m(y_2)$. Finally, we claim that the condition 2(a) holds, as follows. Since x is a parallel node, $N(G,v) \setminus V(\mathsf{CG}(y_1)) = N(G,v) \setminus V(\mathsf{CG}(x))$ holds for all vertices v in $\mathsf{CG}(y_1)$. Similarly, $N(G,w) \setminus V(\mathsf{CG}(y_2)) = N(G,w) \setminus V(\mathsf{CG}(x))$ holds for all vertices w in $\mathsf{CG}(y_2)$. Recall that $V(\mathsf{CG}(x))$ is a module of G, that is, $N(G,v) \setminus V(\mathsf{CG}(x)) = N(G,v') \setminus V(\mathsf{CG}(x))$ holds for any vertices $v, v' \in V(\mathsf{CG}(x))$. Therefore, $N(G,v) \setminus V(\mathsf{CG}(y_1)) = N(G,w) \setminus V(\mathsf{CG}(y_2))$ holds any pair of $v \in V(\mathsf{CG}(y_1))$ and $w \in V(\mathsf{CG}(y_2))$. Thus, the condition 2(a) holds. □

8.3.2 Kernelization algorithm

We now describe how to kernelize an input instance. (See Figure 8.2 as an example.) Notice $\mathsf{pmw}(G) \leq \max\{2, \mathsf{mw}(G)\}$ holds for any graph G. Therefore, in order to prove Theorem 8.1, it suffices to give a kernelization with respect to the parameter $k + \mathsf{pmw}$. Our algorithm traverses a PMD-tree $\mathsf{PMD}(G)$ of G by a depth-first search in post-order starting from the root of $\mathsf{PMD}(G)$, that is, the algorithm processes a node of $\mathsf{PMD}(G)$ after all of its children are processed.

Let $x \in V(\mathsf{PMD}(G))$ be a node which is currently visited. If x is a non-parallel node, we do nothing. Otherwise (i.e., if x is a parallel node,) let Y be the set of all children of x, and let $m := \max_{y \in Y} |V(\mathsf{CG}(y))|$. We first construct m-ID-matrices of all children of x. If there exist two nodes y_1 and y_2 such that $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$, then $\mathsf{CG}(y_1)$ and $\mathsf{CG}(y_2)$ are identical; and hence we remove $\mathsf{CG}(y_2)$ from G by Lemma 8.6. Then, we modify $\mathsf{PMD}(G)$ in order to keep it still being a PMD-tree for the resulting graph as follows. We remove a subtree rooted at y_2 from $\mathsf{PMD}(G)$, and delete a node corresponding to y_2 from a quotient graph $\mathsf{Q}(x)$ of x. If this removal

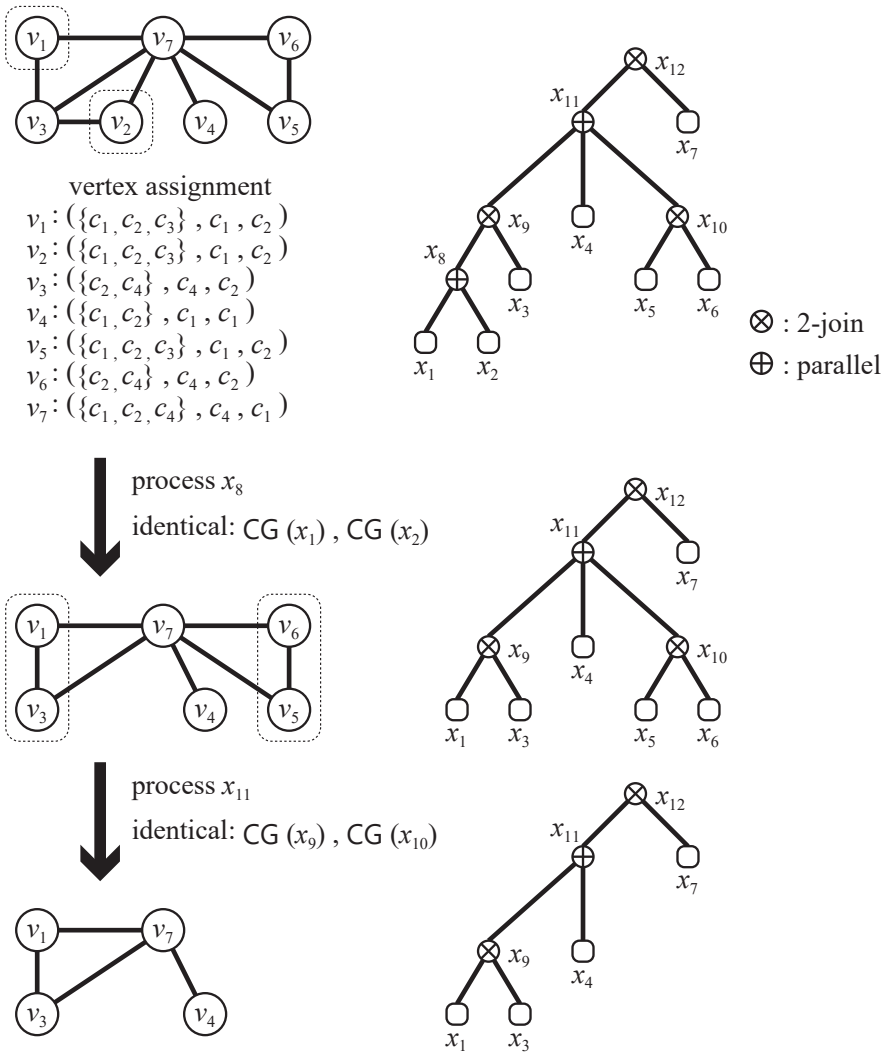


Figure 8.2: An example of an application of our algorithm. We first focus on x_8, which is a parallel node whose children are already kernelized, and find that $\mathcal{M}_1(x_1) = \mathcal{M}_1(x_2)$ holds. Therefore, we delete $\mathsf{CG}(x_1)$ from the input graph. Then, x_8 has only one child, and hence we contract the edge x_8x_9 in order to maintain being a PMD-tree. We next focus on x_{11} and find that $\mathcal{M}_2(x_9) = \mathcal{M}_2(x_{10})$ holds. We thus remove $\mathsf{CG}(x_{10})$ from the current graph and fixing a PMD-tree. Then, the algorithm terminates because we have processed all parallel nodes.

makes x having only one child y in the PMD-tree, we contract the edge xy into a new node x' such that $\mathsf{Q}(x') = \mathsf{Q}(x)$.

The running time of this kernelization can be estimated as follows. For each node $x \in V(\mathsf{PMD}(G))$, the construction of m-ID-matrices can be done in time $O(|Y| \cdot m^2) = O(|V(G)|^3)$. We can check if $\mathcal{M}_m(y_1) = \mathcal{M}_m(y_2)$ for each pair of

children y_1 and y_2 of x in time $O(m^2) = O(|V(G)|^2)$. Moreover, a modification of $\mathsf{PMD}(G)$, which follows an application of Lemma 8.6, can be done in polynomial time. Recall that the number of children of x and the size of a PMD-tree $\mathsf{PMD}(G)$ are both bounded linearly in $|V(G)|$, and hence our kernelization can be done in polynomial time.

8.3.3 Size of the kernelized instance

We finally prove that the size of the obtained instance $\mathcal{I}' = (G', L', f'_s, f'_t)$ depends only on $k+\mathsf{pmw}$; recall that pmw is an upper bound on $\mathsf{pmw}(G)$. By Observation 2.1, we can assume that the maximum clique size $\omega(G')$ is at most k. In addition, G' is connected since G is connected and an application of Lemma 8.6 does not affect the connectivity of the graph. Therefore, it suffices to prove the following lemma.

Lemma 8.8 *The graph G' has at most $h_{k,\mathsf{pmw}}(\omega(G'))$ vertices, where $h_{k,\mathsf{pmw}}(i)$ is recursively defined for an integer $i \geq 1$ as follows*:

$$h_{k,\mathsf{pmw}}(i) = \begin{cases} 1 & \text{if } i = 1; \\ \mathsf{pmw} \cdot h_{k,\mathsf{pmw}}(i-1) \cdot \sqrt{2}^{(h_{k,\mathsf{pmw}}(i-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(i-1)} & \text{otherwise.} \end{cases}$$

In particular, $h_{k,\mathsf{pmw}}(\omega(G'))$ depends only on $k + \mathsf{pmw}$.

Proof. We prove the lemma by induction on $\omega(G')$. If $\omega(G') = 1$, then we have $|V(G')| = 1 = h_{k,\mathsf{pmw}}(1)$ since G' is connected.

We thus assume in the remainder of the proof that $\omega(G') > 1$. Then, the root r of a PMD-tree for G' must be a non-parallel node since G' is connected. Because r has at most $\mathsf{pmw}(G') \leq \mathsf{pmw}$ children, it suffices to show that the corresponding graph of each child of r has at most

$$h_{k,\mathsf{pmw}}(\omega(G') - 1) \cdot \sqrt{2}^{(h_{k,\mathsf{pmw}}(\omega(G')-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(\omega(G')-1)}$$

vertices. We will prove this by showing the following two claims for any child x of r:

(A) $|V(H)| \leq h_{k,\mathsf{pmw}}(\omega(G') - 1)$ holds for any connected component H of $\mathsf{CG}(x)$; and

(B) $\mathsf{CG}(x)$ has at most $\sqrt{2}^{(h_{k,\mathsf{pmw}}(\omega(G')-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(\omega(G')-1)}$ connected components.

In order to prove the claim (A), we first claim that $\omega(\mathsf{CG}(x)) < \omega(G')$ holds. Assume for a contradiction that $\mathsf{CG}(x)$ contains a clique X of size $\omega(G')$. Let $\hat{x}$ be a node of a quotient graph $\mathsf{Q}(r)$ which corresponds to x. By the definition, $\mathsf{Q}(r)$ is connected, and hence there exists a node $\hat{y} \in V(\mathsf{Q}(r))$ which is adjacent to $\hat{x}$. Let $y \in \mathsf{PMD}(G)$ be the child of r corresponding to $\hat{y}$. Recall that all vertices in X are connected with at least one vertex v in $V(\mathsf{CG}(y))$ by the substitution operation, which means that G' has a clique $X \cup \{v\}$ of size $\omega(G') + 1$. This contradicts the assumption that the maximum clique size of G' is $\omega(G')$; this completes the proof of the claim. Note that $\omega(H) \leq \omega(\mathsf{CG}(x)) < \omega(G')$ holds for any connected component H of $\mathsf{CG}(x)$. Therefore, $|V(H)| \leq h_{k,\mathsf{pmw}}(\omega(G') - 1)$ follows from the induction hypothesis.

We next prove the claim (B). If x is a non-parallel node, $\mathsf{CG}(x)$ is connected and hence we are done. The remaining case is where x is a parallel node. Let Y be the set of all children of x. For each connected component H of $\mathsf{CG}(x)$, there exists exactly one child $y \in Y$ such that $V(\mathsf{CG}(y)) \supseteq V(H)$. Since a PMD-tree has no edge joining two parallel nodes, y is not a parallel node. Thus, $\mathsf{CG}(y)$ is connected, and hence we indeed have $V(\mathsf{CG}(y)) = V(H)$. Therefore, it suffices to bound the size of Y instead of the number of connected components in $\mathsf{CG}(x)$. Let $m := \max_{y \in Y} |V(\mathsf{CG}(y))|$. Since G' is already kernelized, $\mathcal{M}_m(y_1) \neq \mathcal{M}_m(y_2)$ holds for any two children $y_1, y_2 \in Y$. Therefore, $|Y|$ cannot exceed the number of distinct m-ID-matrices. Recall that the upper $m \times m$ submatrix consists of m^2 elements from $\{0, 1\}$, its (i, i)-element is 0 for each $i \in \{1, 2, \ldots, m\}$, and it is symmetric. Therefore, the number of such distinct $m \times m$ submatrices can be bounded by

$2^{m^2/2} = \sqrt{2}^{m^2}$. Recall that all elements of the $(m+1)$-st row are chosen from the set $2^C \times C \times C$, where C is the color set of size at most k. Therefore, the number of such distinct $1 \times m$ submatrices can be bounded by $(2^k \cdot k^2)^m$. By the claim (A), we have $m = \max_{y \in Y} |V(\mathsf{CG}(y))| \leq h_{k,\mathsf{pmw}}(\omega(G') - 1)$. Therefore, the size of Y, and hence the number of connected components in $\mathsf{CG}(x)$, can be bounded by

$$\sqrt{2}^{(h_{k,\mathsf{pmw}}(\omega(G')-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(\omega(G')-1)}.$$

From the claims (A) and (B), we have the following inequality

$$|V(G')| \leq \mathsf{pmw} \times h_{k,\mathsf{pmw}}(\omega(G') - 1) \times \sqrt{2}^{(h_{k,\mathsf{pmw}}(\omega(G')-1))^2} \cdot (2^k \cdot k^2)^{h_{k,\mathsf{pmw}}(\omega(G')-1)}$$

as claimed. In particular, we can conclude that $h_{k,\mathsf{pmw}}(\omega(G'))$ depends only on $k + \mathsf{pmw}$, because $\omega(G') \leq k$. □

Thus, Theorem 8.1 now follows from Theorem 2.1 and Lemma 8.8.

Chapter 9 Constraint Satisfiability Reconfiguration

In this chapter, we give several fixed-parameter algorithms and some lower-bound for CSR.

9.1 Graphs with bounded tree-depth

In this section, we show the following theorem.

Theorem 9.1 CSR *is fixed-parameter tractable when parameterized by* $k + \mathsf{td}$.

We remark that the maximum arity of a constraint is bounded by $\mathsf{td}(\mathcal{P}(G))$; since otherwise $\mathcal{P}(G)$ contains a clique of size more than $\mathsf{td}(\mathcal{P}(G))$, which is a contradiction. (See the next subsection for the definition of tree-depth.) Therefore, we can assume that each constraint is explicitly given as a set of mappings.

9.1.1 Definitions

We first define the notion of tree-depth of a graph. Let G be a connected graph. A *tree-depth decomposition* of G is a rooted tree T such that $V(T) = V(G)$ and if $vw \in E(G)$ then one of two endpoint is an ancestor of the other in T. The *depth* of T is the maximum number of vertices of a path in T between the root and a leaf. The *tree-depth* $\mathsf{td}(G)$ of G is the minimum depth of a tree-depth decomposition of G. In Figure 9.1, T is a tree-depth decomposition of G whose depth is three; note that T has the minimum depth, and hence $\mathsf{td}(G) = 3$. It is known that there

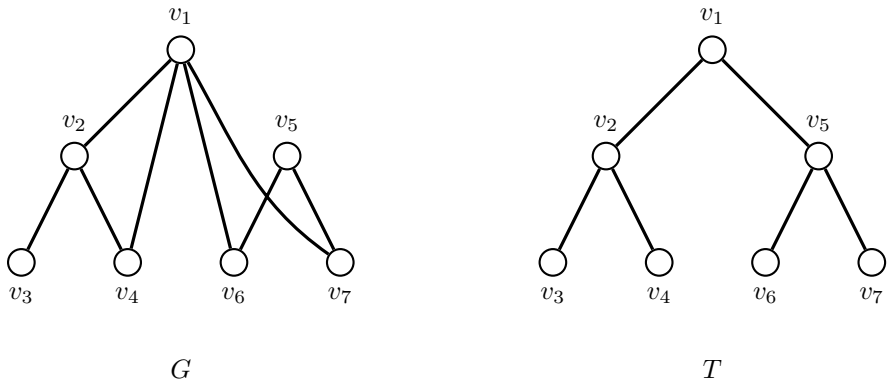


Figure 9.1: A graph G and a tree-depth decomposition T of G with the root v_1.

exists an algorithm computing the tree-depth decomposition of depth td in time $O^*(2^{O(\mathsf{td}^2)})$ [54]. For a tree-depth decomposition T of a graph G and for a vertex $v \in V(T) = V(G)$, T_v denotes the subtree of T rooted at v, and $\mathsf{Anc}(v)$ denotes the set of all ancestors of v. From the definition of a tree-depth decomposition, $N(G, v) \subseteq V(T_v) \cup \mathsf{Anc}(v)$ holds.

9.1.2 Kernelization

In order to prove Theorem 9.1, we give a kernelization with respect to $k + \mathsf{td}$. To this end, we first explain a preprocessing of our kernelization algorithm, which will simplify the discussion, and then give a sufficient condition that two subhypergraphs are identical.

Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be a given instance of CSR, and let T be a tree-depth decomposition of $\mathcal{P}(G)$ with depth td. We assume that all vertices of $V(G)$ are totally ordered in the pre-order of the depth-first search on T starting with its root; let $\prec$ be the binary relation defined by this ordering. Let $X \in E(G)$ be a hyperedge. From the definition of the primal graph, any two distinct vertices v and w in X are adjacent in $\mathcal{P}(G)$, and hence they are in ancestor-descendant relationship in T. Therefore, there exists the unique vertex which is farthest from the root. We call v the *bottommost vertex* of X. Then, for each vertex $v \in V(G)$, we modify the given

instance as follows.

- Remove all hyperedges X whose bottommost vertices are v from G.
- Add a hyperedge $X_v = \mathsf{Anc}(v) \cup \{v\}$ to G, and let $\mathcal{C}(X_v)$ be the set of all mappings $g \in D^{X_v}$ which satisfy the constraints of all removed hyperedges.

Observe that this modification can be done in time $O^*(k^{\mathsf{td}})$, since each vertex has at most td ancestors, and does not change the set of solutions. Therefore, in the remainder of this section, we assume that there exists a one-to-one correspondence between $V(G)$ and $E(G)$ such that $v \in V(G)$ corresponds to the hyperedge $\mathsf{Anc}(v) \cup \{v\}$.

Let $v \in V(T)$ be a vertex, and assume that all vertices in $V(T_v)$ are labeled as $v_1, v_2, \ldots, v_p$ according to $\prec$. Then, we define three p-tuples $\mathscr{N}(v)$, $\mathscr{A}(v)$ and $\mathscr{C}(v)$ as follows.

- The i-th component of $\mathscr{N}(v)$ is $N(T, v_i)$.
- The i-th component of $\mathscr{A}(v)$ is $\mathcal{A}(v_i)$.
- The i-th component of $\mathscr{C}(v)$ is a set of vectors of length $|X_i|$ which corresponding to $\mathcal{C}(X_i)$ under the total order $\prec$, where $X_i = \mathsf{Anc}(v_i) \cup \{v_i\}$.

We call these tuples *ID-tuples* of v. Then, we have the following lemma.

Lemma 9.1 *Let v and w be two children of a vertex u in T such that $|V(T_v)| = |V(T_w)|$. If $(\mathscr{N}(v), \mathscr{A}(v), \mathscr{C}(v)) = (\mathscr{N}(w), \mathscr{A}(w), \mathscr{C}(w))$ holds, then $G[V(T_v)]$ and $G[V(T_w)]$ are identical under some pair of two bijections.*

Proof. By the preprocessing, $N(G, V(T_v)) = N(G, V(T_w)) = \mathsf{Anc}(v) = \mathsf{Anc}(w)$ holds. We denote $H_1 = G[V(T_v)]$, $H_2 = G[V(T_w)]$, $H_1' = G[V(T_v) \cup \mathsf{Anc}(v)]$ and $H_2' = G[V(T_w) \cup \mathsf{Anc}(w)]$. We now define a pair of mappings ϕ and π as follows. ϕ

maps the i-th vertex in $V(H_1')$ to the i-th vertex in $V(H_2')$ according to $\prec$. $\pi(X) = \{\phi(x_1), \phi(x_2), \ldots, \phi(x_r)\}$ holds for each hyperedge $X = \{x_1, x_2, \ldots, x_r\} \in E(H_1')$.

Then, it suffices to show that ϕ and π satisfy the conditions (1) through (4) of Definition 8.1. From the definition of ϕ, the condition (2) holds; that is, $\phi(x) = x$ if $x \in \mathsf{Anc}(v)$. Since $\mathscr{A}(v) = \mathscr{A}(w)$, the condition (3) holds. In order to verify the condition (1), we show that π is a bijection from $E(H_1')$ to $E(H_2')$. The assumption that $\mathscr{N}(v) = \mathscr{N}(w)$ implies that T_v and T_w isomorphic under ϕ. Furthermore, $T[V(T_v) \cup \mathsf{Anc}(v)] = T[V(H_1')]$ and $T[V(T_w) \cup \mathsf{Anc}(w)] = T[V(H_2')]$ are isomorphic, too. Recall that for any vertex $x \in V(G)$, there exists the corresponding hyperedge $X \in E(G)$ such that $X = \mathsf{Anc}(v) \cup \{v\}$. Therefore, for each hyperedge $X \in E(H_1')$ corresponding to $x \in V(H_1')$, $\pi(X)$ is a hyperedge corresponding to $\phi(x) \in V(H_2')$. Thus, condition (1) holds. Finally, since $\mathscr{C}(v) = \mathscr{C}(w)$, the condition (3) holds. □

We now describe our kernelization algorithm. Our algorithm traverses T from leaves to the root, that is, the algorithm processes a vertex of T after its all children are processed.

Let $u \in V(G)$ be a vertex which is currently visited. We check if there is a pair of children v and w which satisfies the conditions of Lemma 9.1. If such a pair is found, we apply Lemma 8.2 to remove $V(T_w)$. We note that the one-to-one correspondence between the vertex set and the hyperedge set is preserved during this process. Therefore, we repeat this as long as such a pair is left.

The running time of this kernelization can be estimated as follows. For each pair of two children of a vertex, Lemma 9.1 can be checked in time polynomial in $|V(G)|$ and the maximum size of a constraint (i.e., $O(k^{\mathsf{td}})$). Since there exist at most $|V(G)|^2$ pairs to be checked, the algorithm runs in time polynomial in $|V(G)|$ and $O(k^{\mathsf{td}})$.

Finally, we prove that the obtained instance $(G', D, \mathcal{C}', f_s', f_t')$ has bounded size. We have the following lemma.

Lemma 9.2 *The graph G' has at most $h_{k,\mathsf{td}}(\mathsf{td})$ vertices, where $h_{k,\mathsf{td}}(j)$ is recursively defined for an integer $j \geq 1$ as follows*:

$$h_{k,\mathsf{td}}(j) = \begin{cases} 1 & \text{if } j = 1; \\ \alpha^2 \cdot (2^\alpha \cdot k^2 \cdot 2^{k^\alpha})^\alpha & \text{otherwise,} \end{cases}$$

where $\alpha = h_{k,\mathsf{td}}(j-1)$. In particular, $h_{k,\mathsf{td}}(\mathsf{td})$ depends only on $k + \mathsf{td}$.

Proof. In order to prove the lemma, we define the *level* of the tree-depth decomposition: each leaf has the level one, and the level of each internal vertex is the maximum level of a child plus one. Then, we prove the following stronger claim:

Claim 9.1 *Let u be a vertex of level i, and let G_u and T' be the graph and the tree-depth decomposition obtained by the algorithm after processing u. Then, $G_u[V(T'_u)]$ has at most $h_{k,\mathsf{td}}(i)$ vertices.*

We prove the claim by the induction on i. If $i = 1$, then we have $|V(G_u[V(T'_u)])| = 1 = h_{k,\mathsf{td}}(1)$ since u is a leaf.

We thus assume in the remainder of the proof that $i > 1$. Consider each child v of u in T'. Since v's level is less than i, and $G_u[T'_v]$ is the graph obtained by the algorithm after processing v, $|V(T'_v)| \leq h_{k,\mathsf{td}}(i-1)$ by the induction hypothesis. Thus, it suffices to show that r has at most $\alpha \cdot (2^\alpha \cdot k^2 \cdot 2^{k^\alpha})^\alpha$ children, where $\alpha = h_{k,\mathsf{td}}(i-1)$. Because the algorithm has processed u, there exists no pair of two children v and w of u which satisfies Lemma 9.1. Therefore, the number of children can be bounded by the number of distinct combinations of three ID-tuples with at most α components. For each $\beta \leq \alpha$, the number of distinct combinations of three ID-tuples with β components can be bounded as follows. Recall that for each child v such that $|T'_v| = \beta$, $\mathscr{N}(v)$, $\mathscr{A}(v)$, and $\mathscr{C}(v)$ consist of β subsets of T'_v, β pairs of tow values from D, and β constraints of arity at most β, respectively. Thus,

- the number of distinct $\mathscr{N}(\cdot)$'s is at most $(2^\beta)^\beta$;
- the number of distinct $\mathscr{A}(\cdot)$'s is at most $(k^2)^\beta$; and

- the number of distinct $\mathscr{C}(\cdot)$'s is at most $(2^{k^\beta})^\beta$.

Therefore, the number of children is at most

$$\sum_{\beta=1}^{\alpha} (2^\beta \cdot k^2 \cdot 2^{k^\beta})^\beta \leq \alpha \cdot (2^\alpha \cdot k^2 \cdot 2^{k^\alpha})^\alpha$$

as required. □

This completes the proof of the theorem.

9.2 Graphs with small vertex cover

In this section, we consider the size vc of a minimum vertex cover. Note that Theorem 9.1 implies CSR is fixed-parameter tractable when parameterized by $k+\mathsf{vc}$. We strengthen it as follows.

Theorem 9.2 *The shortest variant of* CSR *is fixed-parameter tractable when parameterized by* $k + \mathsf{vc}$.

Theorem 9.3 *There exists a fixed-parameter algorithm for* CSR *parameterized by* $k + \mathsf{vc}$ *which runs in time* $O^*(k^{O(\mathsf{vc})})$.

We remark that the maximum arity of a constraint is bounded by $\mathsf{vc}(\mathcal{P}(G)) + 1$; since otherwise $\mathcal{P}(G)$ contains a clique of size more than $\mathsf{vc}(\mathcal{P}(G)) + 1$, which is a contradiction. Therefore, we can again assume that each constraint is explicitly given as a set of mappings.

9.2.1 Proof of Theorem 9.2

We give a fixed-parameter algorithm for a more general "weighted variant" of CSR. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be an instance of CSR, and assume that each vertex $v \in V(G)$ has a *weight* $\omega(v) \in \mathbb{N}$, where $\mathbb{N}$ is the set of all positive integers. According to the weight function ω, we define the weight of each edge $ff' \in$

$E(\mathscr{S}((G, D, \mathcal{C})))$ in the solution graph as the weight of v, where $\{v\} = \mathsf{dif}(f, f')$. We denote by $\mathscr{S}_\omega((G, D, \mathcal{C}))$ the weighted solution graph defined in this way. The *length* $\mathsf{len}_\omega(\mathcal{W})$ of a reconfiguration sequence (i.e., a walk) $\mathcal{W}$ in $\mathscr{S}_\omega((G, D, \mathcal{C}))$ is the sum of the weight of the edges in $\mathcal{W}$. For each vertex $v \in V(G)$, we denote by $\#(\mathcal{W}, v)$ the number of edges ff' in $\mathcal{W}$ such that $\mathsf{dif}(f, f') = \{v\}$. In other words, $\#(\mathcal{W}, v)$ is the number of steps changing the value of v in $\mathcal{W}$. Notice that $\mathsf{len}_\omega(\mathcal{W}) = \sum_{v \in V(G)} \omega(v) \cdot \#(\mathcal{W}, v)$ holds. We denote by $\mathsf{OPT}(\mathcal{I}, \omega)$ the minimum length of a reconfiguration sequence between f_s and f_t; we define $\mathsf{OPT}(\mathcal{I}, \omega) = +\infty$ if $\mathcal{I}$ is a no-instance of CSR. The *weighted variant* of CSR is to determine whether $\mathsf{OPT}(\mathcal{I}, \omega) \leq \ell$ or not for a given instance $(\mathcal{I}, \omega)$ and an integer $\ell \geq 0$. Notice that the shortest variant is equivalent to the weighted variant where every vertex has weight one. Thus, in order to prove Theorem 9.2, it suffices to construct a fixed-parameter algorithm for the weighted variant parameterized by $k + \mathsf{vc}$.

The central idea is the same as the previous section; that is, we again kernelize the input instance.

Reduction rule for the weighted variant

In this subsection, we give the counterpart of Lemma 8.2 for the weighted version.

Let $(\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t), \omega)$ be an instance of the weighted version, and assume that there exist two identical subgraphs H_1 and H_2 of G, both of which consist of single vertices, say, $V(H_1) = \{v_1\}$ and $V(H_2) = \{v_2\}$. We now define a new instance $(\mathcal{I}', \omega')$ as follows:

- $\mathcal{I}'$ is the instance obtained by applying Lemma 8.2 for H_1 and H_2; and
- $\omega'(v_1) = \omega(v_1) + \omega(v_2)$ and $\omega'(v) = \omega(v)$ for any $v \in V(G) \setminus \{v_1, v_2\}$.

Intuitively, v_2 is merged into v_1 together with its weight. Then, we have the following lemma.

Lemma 9.3 $\mathsf{OPT}(\mathcal{I}, \omega) = \mathsf{OPT}(\mathcal{I}', \omega')$.

Proof. Let $\mathcal{I}' = (G', D, f'_s, f'_t)$. By Lemma 8.2, $\mathsf{OPT}(\mathcal{I}, \omega) = +\infty$ if and only if $\mathsf{OPT}(\mathcal{I}', \omega') = +\infty$. Therefore, we assume that $\mathsf{OPT}(\mathcal{I}', \omega') \neq +\infty$ and $\mathsf{OPT}(\mathcal{I}, \omega) \neq +\infty$.

We first show that $\mathsf{OPT}(\mathcal{I}, \omega) \leq \mathsf{OPT}(\mathcal{I}', \omega')$. Since $\mathsf{OPT}(\mathcal{I}, \omega) \leq \mathsf{len}_\omega(\mathcal{W})$ holds for any reconfiguration sequence $\mathcal{W}$ for $\mathcal{I}$, it suffices to show that there exists a reconfiguration sequence for $\mathcal{I}$ whose length is at most $\mathsf{OPT}(\mathcal{I}', \omega')$. Let $\mathcal{W}'$ be a shortest reconfiguration sequence for $\mathcal{I}'$ such that $\mathsf{len}_{\omega'}(\mathcal{W}') = \mathsf{OPT}(\mathcal{I}', \omega')$. Following the only-if direction proof of Lemma 8.2, we can construct a reconfiguration sequence $\mathcal{W}$ for $\mathcal{I}$ such that $\#(\mathcal{W}, v_1) = \#(\mathcal{W}, v_2) = \#(\mathcal{W}', v_1)$ and $\#(\mathcal{W}, v) = \#(\mathcal{W}', v)$ for any $v \in V(G) \setminus \{v_1, v_2\}$. Therefore,

$$\begin{aligned}
\mathsf{len}_\omega(\mathcal{W}) &= \textstyle\sum_{v \in V(G)} \omega(v) \cdot \#(\mathcal{W}, v) \\
&= \omega(v_1) \cdot \#(\mathcal{W}, v_1) + \omega(v_2) \cdot \#(\mathcal{W}, v_2) + \textstyle\sum_{v \in V(G) \setminus \{v_1, v_2\}} \omega(v) \cdot \#(\mathcal{W}, v) \\
&= (\omega(v_1) + \omega(v_2)) \cdot \#(\mathcal{W}, v_1) + \textstyle\sum_{v \in V(G) \setminus \{v_1, v_2\}} \omega(v) \cdot \#(\mathcal{W}, v) \\
&= \omega'(v_1) \cdot \#(\mathcal{W}', v_1) + \textstyle\sum_{v \in V(G) \setminus \{v_1, v_2\}} \omega'(v) \cdot \#(\mathcal{W}', v) \\
&= \textstyle\sum_{v \in V(G')} \omega'(v) \cdot \#(\mathcal{W}', v) \\
&= \mathsf{len}_{\omega'}(\mathcal{W}') \\
&= \mathsf{OPT}(\mathcal{I}', \omega').
\end{aligned}$$

Thus, $\mathcal{W}$ is a desired reconfiguration sequence for $\mathcal{I}$.

We next show that $\mathsf{OPT}(\mathcal{I}', \omega') \leq \mathsf{OPT}(\mathcal{I}, \omega)$. Since $\mathsf{OPT}(\mathcal{I}', \omega') \leq \mathsf{len}_{\omega'}(\mathcal{W}')$ holds for any reconfiguration sequence $\mathcal{W}'$ for $\mathcal{I}'$, it suffices to show that there exists a reconfiguration sequence for $\mathcal{I}'$ whose length is at most $\mathsf{OPT}(\mathcal{I}, \omega)$. Let $\mathcal{W}$ be a shortest reconfiguration sequence for $\mathcal{I}$ such that $\mathsf{len}_\omega(\mathcal{W}) = \mathsf{OPT}(\mathcal{I}, \omega)$. We now construct a reconfiguration sequence for $\mathcal{I}'$ from $\mathcal{W}$ such that $\mathsf{len}_{\omega'}(\mathcal{W}') \leq \mathsf{OPT}(\mathcal{I}, \omega)$ as follows.

Case 1. $\#(\mathcal{W}, v_1) \leq \#(\mathcal{W}, v_2)$.

In this case, we restrict all solutions in $\mathcal{W}$ on $V(G')$ to obtain a reconfiguration sequence $\mathcal{W}_1$ for $\mathcal{I}'$; recall the if direction proof of Lemma 8.2. From the construction, $\#(\mathcal{W}_1, v_1) = \#(\mathcal{W}, v_1) \leq \#(\mathcal{W}, v_2)$ and $\#(\mathcal{W}_1, v) = \#(\mathcal{W}, v)$ holds for any vertex

$v \in V(G') = V(G) \setminus \{v_2\}$. Therefore, we have

$$\begin{aligned}
\mathsf{len}_{\omega'}(\mathcal{W}_1) &= \textstyle\sum_{v \in V(G')} \omega'(v) \cdot \#(\mathcal{W}_1, v) \\
&= \omega'(v_1) \cdot \#(\mathcal{W}_1, v_1) + \textstyle\sum_{v \in V(G') \setminus \{v_1\}} \omega'(v) \cdot \#(\mathcal{W}_1, v) \\
&= (\omega(v_1) + \omega(v_2)) \cdot \#(\mathcal{W}, v_1) + \textstyle\sum_{v \in V(G) \setminus \{v_1, v_2\}} \omega(v) \cdot \#(\mathcal{W}, v) \\
&\leq \omega(v_1) \cdot \#(\mathcal{W}, v_1) + \omega(v_2) \cdot \#(\mathcal{W}, v_2) + \textstyle\sum_{v \in V(G) \setminus \{v_1, v_2\}} \omega(v) \cdot \#(\mathcal{W}, v) \\
&= \textstyle\sum_{v \in V(G)} \omega(v) \cdot \#(\mathcal{W}, v) \\
&= \mathsf{len}_{\omega}(\mathcal{W}) \\
&= \mathsf{OPT}(\mathcal{I}, \omega).
\end{aligned}$$

Thus, $\mathcal{W}_1$ is a desired reconfiguration sequence for $\mathcal{I}'$.

Case 2. $\#(\mathcal{W}, v_1) > \#(\mathcal{W}, v_2)$.

In this case, instead of restricting solutions in $\mathcal{W}$ on $V(G') = V(G) \setminus \{v_2\}$, we restrict them on $V(G) \setminus \{v_1\}$ and obtain a reconfiguration sequence $\mathcal{W}_2$ for an instance obtained by restricting on $V(G) \setminus \{v_1\}$. Then, because H_1 and H_2 are identical, we can easily "rephrase" $\mathcal{W}_2$ as a reconfiguration sequence $\mathcal{W}_2'$ for $\mathcal{I}'$. By the same arguments as the case 1 above, we have $\mathsf{len}_{\omega'}(\mathcal{W}_2') < \mathsf{len}_{\omega}(\mathcal{W}) = \mathsf{OPT}(\mathcal{I}, \omega)$. Thus, $\mathcal{W}_2'$ is a desired reconfiguration sequence for $\mathcal{I}'$.

In this way, we have shown that $\mathsf{OPT}(\mathcal{I}, \omega) = \mathsf{OPT}(\mathcal{I}', \omega')$ as claimed. □

Kernelization

Finally, we give a kernelization algorithm as follows.

Let $(\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t), \omega)$ be an instance of the weighted version such that the primal graph $\mathcal{P}(G)$ has a vertex cover of size at most vc. Because such a vertex cover can be computed in time $O(2^{\mathsf{vc}} \cdot n)$ [22], assume that we are given a vertex cover C of size at most vc. Notice that $I := V(G) \setminus C$ forms an independent set of $\mathcal{P}(G)$. In order to simplify the proof, we first modify $\mathcal{I}$ without changing the set of solutions as follows. For each vertex v in I, we remove all hyperedges containing v and add a new hyperedge $C_v = C \cup \{v\}$; note that each removed hyperedges are subsets of C_v since I is an independent set of $\mathcal{P}(G)$. We then define a constraint of C_v as the set of all mappings $g \in D^{C_v}$ satisfying the constraints of all removed

hyperedges. Observe that this modification can be done in time $O^*(k^{O(\mathsf{vc})})$ and does not change the set of solutions. Thus, we can assume that for each vertex $v \in I$, there exists the unique hyperedge C_v such that $C_v = C \cup \{v\}$.

Let v_1 and v_2 be two vertices in I. We now define two mappings $\phi\colon C_{v_1} \to C_{v_2}$ and $\pi\colon \{C_{v_1}\} \to \{C_{v_2}\}$ as follows: $\phi(w) = w$ if $w \in C$, $\phi(v_1) = v_2$, and $\pi(C_{v_1}) = C_{v_2}$. Suppose that $\mathcal{A}(v_1) = \mathcal{A}(v_2)$ and $\mathcal{C}(C_{v_2}) = \mathcal{C}[\phi](C_{v_1})$ hold. Then, induced subgraphs $G[\{v_1\}]$ and $G[\{v_2\}]$ are identical. Therefore, we can apply Lemma 9.3 to remove v_2 from G, and modify the weight function without changing the optimality. As a kernelization, we repeatedly apply Lemma 9.3 for all such pairs of vertices in I, which can be done in polynomial time. Let G' be the resulting subgraph of G, and let $I' := V(G') \setminus C$. Since C is of size at most vc, it suffices to prove the following lemma.

Lemma 9.4 $|I'| \leq k^2 \cdot 2^{k^{\mathsf{vc}+1}}$.

Proof. Recall that I' contains no pair of vertices which correspond to identical subgraphs, and hence any pair of vertices $v_1, v_2 \in I'$ does not satisfy at least one of $\mathcal{A}(v_1) = \mathcal{A}(v_2)$ and $\mathcal{C}(C_{v_2}) = \mathcal{C}[\phi](C_{v_1})$. Therefore, $|I'|$ can be bounded by the number of distinct combinations of a vertex assignment and a constraint of arity $\mathsf{vc} + 1$. Since the domain has size k, the number of (possible) vertex assignments can be bounded by k^2. Since a constraint of arity $\mathsf{vc} + 1$ can be seen as a subset of $D^{\mathsf{vc}+1}$, the number of (possible) constraints can be bounded by $2^{k^{\mathsf{vc}+1}}$. We thus have $|I'| \leq k^2 \cdot 2^{k^{\mathsf{vc}+1}}$ as claimed. □

This completes the proof of Theorem 9.2.

9.2.2 Proof of Theorem 9.3

In order to prove the theorem, we first introduce the notion of a "contracted solution graph", which was used to solve some reconfiguration problems [3, 7].

Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of CSR, where $\mathcal{J} = (G, D, \mathcal{C})$, and let $\mathscr{P}$ be a partition of the vertex set of the solution graph $\mathscr{S}(\mathcal{J})$. The *contracted solution graph* (or *CSG* for short) $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ is defined as follows. The vertex set $V(\mathsf{CSG}(\mathcal{J}, \mathscr{P}))$ is exactly $\mathscr{P}$; we call each vertex of the CSG a *node*. Each pair of distinct nodes (i.e., sets of solutions) $P, P' \in \mathscr{P}$ are adjacent in the CSG if and only if there exist two solutions $f \in P$ and $f' \in P'$ such that $ff' \in E(\mathscr{S}(\mathcal{J}))$. In other words, $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ is obtained by contracting a (possibly disconnected) subgraph of $\mathscr{S}(\mathcal{J})$ induced by each set $P \in \mathscr{P}$ into one node. A partition $\mathscr{P}$ is *proper* if every set $P \in \mathscr{P}$ induces a connected subgraph of $\mathscr{S}(\mathcal{J})$. Since the contraction of a connected subgraph maintains the connectivity of a graph, we have the following proposition.

Proposition 9.1 *Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of* CSR*, where $\mathcal{J} = (G, D, \mathcal{C})$, and let $\mathscr{P}$ be a proper partition of $V(\mathscr{S}(\mathcal{J}))$. Then, $\mathcal{I}$ is a yes-instance if and only if there exists a walk between P_s and P_t in $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$, where $f_s \in P_s$ and $f_t \in P_t$. Moreover, the above condition can be checked in time polynomial in $|\mathscr{P}|$.*

Therefore, we first define a proper partition $\mathscr{P}$ such that $|\mathscr{P}|$ depends only on $k + \mathsf{vc}$, and then give an algorithm constructing the CSG and specifying the nodes corresponding to f_s and f_t.

Defining a proper partition

Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of CSR, where $\mathcal{J} = (G, D, \mathcal{C})$. Assume that $\mathcal{P}(G)$ has a vertex cover C of size at most vc. For each solution $f \in V(\mathscr{S}(\mathcal{J}))$, we define $[f] = \{f' \colon f|_C = f'|_C\}$. Then, we define $\mathscr{P} = \{[f] \colon f \in V(\mathscr{S}(\mathcal{J}))\}$; that is, $\mathscr{P}$ is the set of the equivalence classes under the equivalence relation "their restrictions on C are the same". Clearly, $\mathscr{P}$ is a partition of $V(\mathscr{S}(\mathcal{J}))$ and $|\mathscr{P}|$ is bounded by the number of mappings from C to D, that is, $|\mathscr{P}| \leq k^{\mathsf{vc}}$.

In order to prove that $\mathscr{P}$ is proper, we introduce some notation. Let $S \subseteq V(G)$ be a vertex subset, and let $h \colon S \to D$ be a mapping from S to D. We define the

substitution $\mathsf{SUB}(\mathcal{J}; h)$ as an instance $(G', D, \mathcal{C}')$ of CONSTRAINT SATISFIABILITY such that:

- $G' = G \setminus S$; and
- for each $X' \in E(G')$, $\mathcal{C}'(X') = \bigcap_{X \in E'} \mathcal{G}(X)$, where $E' = \{X \in E(G) \colon X \setminus S = X'\}$ and $\mathcal{G}(X) = \{g|_{X'} \colon g \in \mathcal{C}(X), h \text{ and } g \text{ are compatible}\}$.

We have the following lemma.

Lemma 9.5 *Let $f' \colon V(G) \setminus S \to D$ and $f \colon V(G) \to D$ be two mappings such that $f|_{V(G)\setminus S} = f'$. Then, f' is a solution for $\mathsf{SUB}(\mathcal{J}; f|_S) = (G', D, \mathcal{C}')$ if and only if f is a solution for $(G, D, \mathcal{C})$.*

Proof. We first show the if direction. Assume that f is a solution for $(G, D, \mathcal{C})$. Let X' be any hyperedge of G'. For every hyperedge X of G such that $X \setminus S = X'$, $f|_X \in \mathcal{C}(X)$ holds. Since $f|_S$ and $f|_X$ are compatible, by the definition of $\mathcal{C}'$, $(f|_X)|_{X'}$ is in $\mathcal{C}'(X')$. In addition, $(f|_X)|_{X'} = f|_{X'} = f'|_{X'}$ holds, and hence, f' satisfies $\mathcal{C}'(X')$. Therefore, f' is a solution for $\mathsf{SUB}(\mathcal{J}; f|_S)$.

We next show the only-if direction. Assume that f' is a solution for $\mathsf{SUB}(\mathcal{J}; f|_S) = (G', D, \mathcal{C}')$. Let X be any hyperedge of G. Then, there exists a hyperedge X' of G' such that $X \setminus S = X'$. From the definition of $\mathcal{C}'$, there exists a mapping $g \in \mathcal{C}(X)$ such that $f'|_{X'} = g|_{X'}$, and $f|_S$ and g are compatible, that is, $f|_{S\cap X} = g|_{S\cap X}$. Since $f'|_{X'} = f|_X$ and $X' \cup (S \cap X) = X$, $f|_X = g|_X$ holds, and hence f satisfies $\mathcal{C}(X)$. Therefore, f is a solution for $(G, D, \mathcal{C})$. □

The following lemma implies that $\mathscr{P}$ is proper.

Lemma 9.6 *Let P be a solution set in $\mathscr{P}$ such that $f|_C = h$ holds for every $f \in P$. Then, $\mathscr{S}(\mathcal{J})[P]$ is connected.*

Proof. By Lemma 9.5, there exists a one-to-one correspondence between P and the solution set for $\mathsf{SUB}(\mathcal{J}; h)$ which preserves the adjacency relation. Thus, we

suffices to show that $\mathscr{S}(\mathsf{SUB}(\mathcal{J}; h))$ is connected. Since C is a vertex cover of $\mathcal{P}(G)$, $\mathcal{P}(G)[V(G) \setminus C]$ has no edges. This means that $\mathsf{SUB}(\mathcal{J}; h)$ contains only 1-ary constraints. Therefore, a value assignment of each vertex $v \in V(G) \setminus C$ can be changed independently, and hence $\mathscr{S}(\mathsf{SUB}(\mathcal{J}; h))$ is connected. □

Algorihm computing CSG

In order to give an algorithm computing $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ correctly, we first show two claims.

Claim 9.2 *Let h be a mapping from C to D. Then, $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ has a node corresponding to h if and only if $\mathsf{SUB}(\mathcal{J}; h) = (G', D, \mathcal{C}')$ has a solution.*

Proof. By Lemma 9.5, $\mathsf{SUB}(\mathcal{J}; h)$ has a solution f' if and only if there exists a solution f for $\mathcal{J}$ such that $f|_{V(G')} = f'$ and $f|_C = h$. Since $\mathscr{P}$ is a partition of the solution set, there exists a set $P \in \mathscr{P}$ which contains f; and hence $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ has a node corresponding to h. □

Claim 9.3 *Let P_1 and P_2 be two nodes of $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$, and let $h_1 \colon C \to D$ and $h_2 \colon C \to D$ be mappings corresponding to P_1 and P_2, respectively. Then, $P_1P_2 \in E(\mathsf{CSG}(\mathcal{J}, \mathscr{P}))$ if and only if both of the following conditions hold:*

- $|\mathsf{dif}(h_1, h_2)| = 1$*; and*
- $\mathsf{SUB}(\mathcal{J}; h_1)$ *and* $\mathsf{SUB}(\mathcal{J}; h_2)$ *has a common solution f'.*

Proof. We first assume that the above two conditions hold. For each $i \in \{1, 2\}$, let $f_i \colon V(G) \to C$ be a mapping such that $f_i|_{V(G')} = f'$ and $f_i|_C = h_i$. Then, by Lemma 9.5, f_1 and f_2 are adjacent solutions such that $f_1 \in P_1$ and $f_2 \in P_2$; and hence $P_1P_2 \in E(\mathsf{CSG}(\mathcal{J}, \mathscr{P}))$.

We next assume that $P_1P_2 \in E(\mathsf{CSG}(\mathcal{J}, \mathscr{P}))$, that is, there exist two solutions f_1 and f_2 such that $|\mathsf{dif}(f_1, f_2)| = 1$, $f_1 \in P_1$ and $f_2 \in P_2$. From the definition of

$\mathscr{P}$, $h_1 \neq h_2$. Therefore, $\mathsf{dif}(f_1, f_2) = \mathsf{dif}(h_1, h_2)$, and hence the first condition holds. Furthermore, $f_1|_{V(G)\setminus C} = f_2|_{V(G)\setminus C}$ holds. By Lemma 9.5, $f_1|_{V(G)\setminus C}$ and $f_2|_{V(G)\setminus C}$ are solutions for $\mathsf{SUB}(\mathcal{J}; h_1)$ and $\mathsf{SUB}(\mathcal{J}; h_2)$, respectively; and hence the second condition hold. □

From Claims 9.2 and 9.3, we can construct the following algorithm to compute $\mathsf{CSG}(\mathcal{J}, \mathscr{P})$ with nodes corresponding to f_s and f_t.

Phase 1 For each mapping h from C to D, check if $\mathsf{SUB}(\mathcal{J}; h)$ has a solution. If so, create a node corresponding to h. For each $r \in \{s, t\}$, if $h = f_r|_C$, it corresponds to f_r.

Phase 2 For each pair of two nodes P_1 and P_2, check if the two conditions of Claim 9.3 hold. If so, join them by an edge.

The correctness follows from Claims 9.2 and 9.3. The first phase can be done in polynomial time for each mapping, because the constructed instance $\mathsf{SUB}(\mathcal{J}; h)$ of CONSTRAINT SATISFIABILITY contains only 1-ary constraints. Since $|D^C| \leq k^{\mathsf{vc}}$, whole running time of this phase is $O^*(k^{\mathsf{vc}})$. In the second phase, the second condition of Claim 9.3 can be checked as follows. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ are constraint assignments in the substitutions $\mathsf{SUB}(\mathcal{J}; h_1)$ and $\mathsf{SUB}(\mathcal{J}; h_2)$. We now define for each $X' \in E(G')$ a constraint $\mathcal{C}'(X') = \mathcal{C}_1(X') \cap \mathcal{C}_2(X')$. Then, a solution for $(G', D, \mathcal{C}')$ is also a solution for both of $\mathsf{SUB}(\mathcal{J}; h_1)$ and $\mathsf{SUB}(\mathcal{J}; h_2)$. Because $(G', D, \mathcal{C}')$ is an instance of CONSTRAINT SATISFIABILITY which contains only 1-ary constraints, we can solve it in polynomial time. Therefore, whole running time of this phase is $O^*(k^{O(\mathsf{vc})})$.

We thus completed the proof of Theorem 9.3.

9.2.3 Discussions

We conclude this section by showing some corollaries and discussing hitting sets on hypergraphs, which is a well-known generalization of vertex covers on graphs.

Corollaries

We here show some results as corollaries of Theorems 9.2 and 9.3.

Corollary 9.1 *The shortest variant of* LHR *is fixed-parameter tractable for split graphs when parameterized by k. Moreover, there exists a fixed-parameter algorithm for* LHR *on split graphs parameterized by k which runs in time $O^*(k^{O(k)})$.*

Proof. Let $(G, D, \mathcal{C}, f_s, f_t)$ be an instance of LHR such that G is split, L is a list assignment, and H is an underlying graph. Recall that the vertex set of a split graph G can be partitioned into a clique V_Q and an independent set V_I. Therefore V_Q is a vertex cover of G and hence $\mathsf{vc}(G) \leq \omega(G)$ holds. By Observation 2.1 and the fact that f_s is an L-homomorphism from G to H, we have $\omega(G) \leq |V(H)| = k$. Thus, $\mathsf{vc}(G) \leq k$ always holds if G is split. □

Corollary 9.2 *There exists a fixed-parameter algorithm for* CR *parameterized by* vc *which runs in time $O^*(\mathsf{vc}^{O(\mathsf{vc})})$.*

Proof. We first consider the case where $k \leq \mathsf{vc} + 1$. Then, the statement follows directly from Theorem 9.3. We next consider the case where $k \geq \mathsf{vc} + 2$. We note that any graph is vc-degenerate. Since it is known that any instance is a yes-instance if the graph is d-degenerate and $k \geq d + 2$ [4, Theorem 11], we can conclude that it is a yes-instance. □

Hitting sets

Although a hitting set of a 2-uniform hypergraph is equivalent to a vertex cover of the graph, such an equivalence does not hold for general hypergraphs. Thus, it is worth considering the complexity of CSR with respect to the size of a hitting set of a given hypergraph. For a hypergraph G, a vertex subset $V' \subseteq V(G)$ is a *hitting set* if $V' \cap X \neq \emptyset$ holds for every hyperedge $X \in E(G)$. We have the following theorem,

which implies that a fixed-parameter algorithm for CSR is unlikely to exist when parameterized by the size of a hitting set plus k.

Theorem 9.4 3-CSR *is PSPACE-complete even for hypergraphs with a hitting set of size one and* $k = 3$.

Proof. Let $\mathcal{I} = (G, D, \mathcal{C}, f_s, f_t)$ be an instance constructed in Theorem 6.1. Briefly speaking, we add a new vertex u to G, include it in every edge $vw \in E(G)$, and modify the constraints so that a value assignment to u does not affect any other vertices. More precisely, we construct a new instance $\mathcal{I}' = (G', D, \mathcal{C}', f'_s, f'_t)$ as follows. Let $V(G') := V(G) \cup \{u\}$ and $E(G') := \{\{u, v, w\} \colon vw \in E(G)\}$, where u is a new vertex which is not in G. Then, G' has a hitting set $\{u\}$ of size one. For each hyperedge $\{u, v, w\} \in E(G')$, we let $\mathcal{C}'(\{u, v, w\}) := \{1\} \times \mathcal{C}(vw)$. We finally extend f_s (resp., f_t) to f'_s (resp., f'_t) by setting $f'_s(u) = 1$ (resp., $f'_t(u) = 1$). This completes the construction. $\mathcal{I}'$ is equivalent to $\mathcal{I}$ by ignoring a value assignment to u. □

9.3 Extension of the algorithm for binary BCSR

In this section, we show the following theorem.

Theorem 9.5 *There exists a fixed-parameter algorithm for* 2-CSR *parameterized by* $k + \mathsf{nb}$ *which runs in time* $O^*(k^{O(\mathsf{nb})})$.

We note that $k = 2$ implies that $\mathsf{nb} = 0$, and hence this generalizes Theorem 6.2.

Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of 2-CSR, where $\mathcal{J} = (G, D, \mathcal{C})$. We denote by V_{B} and V_{N} be the set of Boolean and non-Boolean vertices, respectively. We first define a partition similarly to Section 9.2.2 as follows. For each solution $f \in V(\mathscr{S}(\mathcal{J}))$, we define $[f] = \{f' \colon f|_{V_{\mathsf{N}}} = f'|_{V_{\mathsf{N}}}\}$. Then, we define $\mathscr{P} = \{[f] \colon f \in$

$V(\mathscr{S}(\mathcal{J}))\}$. Clearly, $\mathscr{P}$ is a partition of $V(\mathscr{S}(\mathcal{J}))$ and $|\mathscr{P}| \leq k^{\mathsf{nb}}$. In contrast to Section 9.2.2, however, $\mathscr{P}$ may be improper in this case.[1]

Therefore, as the first step of our algorithm, we modify the given instance so that $\mathscr{P}$ is proper by some preprocessing. More formally, we show the following lemma.

Lemma 9.7 *Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of* 2-CSR*, where $\mathcal{J} = (G, D, \mathcal{C})$. We can compute in polynomial time an instance $\mathcal{I}^* = (\mathcal{J}^*, f_s^*, f_t^*)$ of* 2-CSR *such that:*

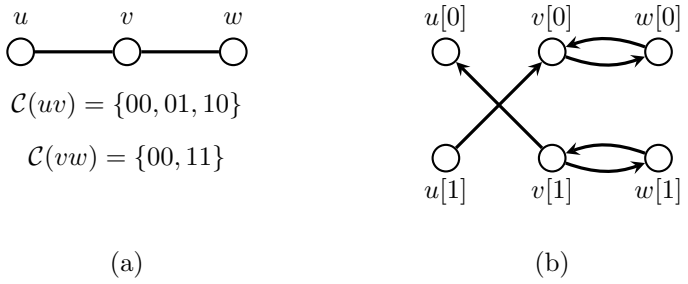
1. *the number of non-Boolean vertices in $\mathcal{J}^*$ is at most that of $\mathcal{J}$;*
2. *$\mathcal{I}$ is a yes-instance if and only if $\mathcal{I}^*$ is; and*
3. *the partition for $\mathcal{J}^*$ is proper.*

Then, we can compute the CSG in the same way as Section 9.2.2; we just replace C with V_{N}. In order to check the existence of vertices and edges in the CSG, we solve instances of Boolean 2-constraint satisfiability which are constructed by the substitution operation. Since Boolean 2-constraint satisfiability can be solved in polynomial time [56], the whole running time of the algorithm is $O^*(k^{O(\mathsf{nb})})$. In the remainder of this section, we prove Lemma 9.7.

9.3.1 Implication graphs

In order to describe a preprocessing, we first introduce the notion of "implication graph", which was first introduced in [26] in order to prove the tractability of some variant of Boolean 2-CSR. Let $\mathcal{J} = (G, D, \mathcal{C})$ be an instance of constraint satisfiability where $\{0, 1\}$ is a domain; we consider the values 0 and 1 as Boolean values. We define the *implication graph* $\mathsf{IMP}(\mathcal{J})$ for $\mathcal{J}$ as follows. (See Figure 9.2

[1] As a simple example, let us consider 2-colorings of K_2. Clearly V_{N} is empty set, and hence all (indeed, only two) 2-colorings of G are in the same set of $\mathscr{P}$. On the other hand, they are not reconfigurable each other.



(a) (b)

Figure 9.2: (a) An instance $\mathcal{J} = (G, \{0, 1\}, \mathcal{C})$ and (b) the implication graph $\mathsf{IMP}(\mathcal{J})$ for $\mathcal{J}$.

for an example.) For each vertex $v \in V(G)$ and for each value $i \in D$ such that there exists a solution f with $f(v) = i$, we add a vertex $v[i]$ to $\mathsf{IMP}(\mathcal{J})$. For each adjacent vertices $v, w \in V(G)$, add two arcs $v[i] \to w[\neg j]$ and $w[j] \to v[\neg i]$ if and only if $(i, j) \notin \mathcal{C}(vw)$, where $\neg$ denote a negation of a Boolean value. Intuitively, an arc $v[i] \to w[\neg j]$ means that if v is assigned i, then w must be assigned $\neg j$ in any solution. We note that $\mathsf{IMP}(\mathcal{J})$ can be computed in polynomial time, since the existence of a vertex $v[i]$ can be checked in polynomial time by solving Boolean 2-constraint satisfiability instance obtained by substituting $f(v) = i$. We now prove the following lemma.

Lemma 9.8 *If there exists a vertex $v \in V(G)$ such that $v[0]$ or $v[1]$ is contained in a directed cycle of $\mathsf{IMP}(\mathcal{J})$, any two solutions f_0 and f_1 for $\mathcal{J}$ such that $f_0(v) = 0$ and $f_1(v) = 1$ are not reconfigurable. On the other hand, if $\mathsf{IMP}(\mathcal{J})$ contains no directed cycles, $\mathscr{S}(\mathcal{J})$ is connected.*

Proof. The second statement can be proved by the similar argument as the proof of Lemma 4.9 in [26], although our implication graph is slightly different from the original one.

Therefore, we prove the first statement. Let $C = v_0[i_0] \to v_1[i_1] \to \cdots \to v_m[i_m] \to v_0[i_0]$ be a directed cycle in $\mathsf{IMP}(\mathcal{J})$, where $v_0 = v$ and $i_j \in \{0, 1\}$, $0 \leq j \leq m$. From the construction, $m \geq 1$ holds. Without loss of generality, assume that $i_0 = 0$. Recall that each arc $v[i] \to w[\neg j]$ means that if v is assigned i, then

w must be assigned $\neg j$. Then, $f_0(v_p) = i_p$ holds for every $p \in \{0, 1, \ldots, m\}$. Moreover, by contrapositions of the above implications, $f_1(v_p) = \neg i_p$ also holds for every $p \in \{0, 1, \ldots, m\}$. We assume for a contradiction that f_0 and f_1 are reconfigurable, and consider the first solution f in a reconfiguration sequence such that $f(v_p) = \neg i_p$ for some $p \in \{0, 1, \ldots, m\}$. Since there exists an arc $v_q[i_q] \to v_p[i_p]$ in a directed cycle C, and hence $f(v_q)$ must be $\neg i_q$. However, by the definition of f, we have $f(v_q) = i_q$, which is a contradiction. □

9.3.2 Preprocessing

We now explain a preprocessing, which eliminates all "undesirable" vertices which prevent the partition from being proper. Let $\mathcal{I} = (\mathcal{J}, f_s, f_t)$ be an instance of 2-CSR, where $\mathcal{J} = (G, D, \mathcal{C})$. Without loss of generality, we can assume that a list $L(v)$ of every Boolean vertex $v \in V_{\mathsf{B}}$ is a subset of $\{0, 1\}$ by a simple value replacement. Then, we define the instance $\mathcal{J}^{\mathsf{res}} = (G^{\mathsf{res}}, D, \mathcal{C}^{\mathsf{res}})$ of 2-CONSTRAINT SATISFIABILITY as the instance obtained by restricting all component of $\mathcal{J}$ on V_{B}. That is,

- $G^{\mathsf{res}} = G[V_{\mathsf{B}}]$; and
- for each $X' \in E(G^{\mathsf{res}})$, $\mathcal{C}^{\mathsf{res}}(X') = \bigcap_{X \in E'} \mathcal{G}(X)$, where $E' = \{X \in E(G) \colon X \cap V_{\mathsf{B}} = X'\}$ and $\mathcal{G}(X) = \{g|_{X'} \colon g \in \mathcal{C}(X)\}$.

Let V_{fix} be the set of vertices $v \in V(G)$ such that $v[0]$ or $v[1]$ are contained in a directed cycle of $\mathsf{IMP}(\mathcal{J}^{\mathsf{res}})$. By Lemma 9.8, in any solution for $\mathcal{J}^{\mathsf{res}}$, all vertices v in V_{fix} are fixed, that is, cannot be reconfigured at all. This property also holds for the original instance $\mathcal{J}$. Therefore, if $f_s|_{V_{\mathsf{fix}}} \neq f_t|_{V_{\mathsf{fix}}}$ holds, then we can immediately conclude that $\mathcal{I}$ is a no-instance. In the other case, we construct in polynomial time the substitution $\mathsf{SUB}(\mathcal{J}; h')$, where $h' = f_s|_{V_{\mathsf{fix}}} = f_t|_{V_{\mathsf{fix}}}$. Then, we have the following proposition.

Proposition 9.2 *$\mathcal{I}$ is a yes-instance if and only if $\mathcal{I}' = (\mathsf{SUB}(\mathcal{J}; h'), f_s|_{V(G')}, f_t|_{V(G')})$ is.*

Proof. Because all vertices v in V_{fix} are fixed, $\mathcal{I}$ is a yes-instance if and only if there exists a reconfiguration sequence $\mathcal{W}$ such that every solutions f in $\mathcal{W}$ satisfies $f|_{V_{\mathsf{fix}}} = h'$. By Lemma 9.5, there exists such a reconfiguration sequence if and only if there exists a reconfiguration sequence between $f_s|_{V(G')}$ and $f_t|_{V(G')}$ in $\mathscr{S}(\mathsf{SUB}(\mathcal{J}; h'))$. □

Therefore, we can obtain an equivalent instance which satisfies the conditions (1) and (2) of Lemma 9.7 by repeating the above transformation until the corresponding implication graph becomes acyclic or empty. Since the number of vertices decreases during the process, this can be done in polynomial time. Let $\mathcal{I}^* = (\mathcal{J}^*, f_s^*, f_t^*)$, where $\mathcal{J}^* = (G^*, D, \mathcal{C}^*)$, be an instance obtained by this preprocessing. Then, it is left to prove that $\mathcal{I}^*$ satisfies the condition (3).

9.3.3 Properity of the partition

Let $\mathscr{P}$ be the partition for $\mathcal{J}^*$, and let P be any solution set in $\mathscr{P}$ such that the restriction of every solution in P on V_{N} is h. By Lemma 9.8, in order to prove that $\mathscr{P}$ is proper, it suffices to show that $\mathsf{IMP}(\mathsf{SUB}(\mathcal{J}^*; h))$ has no directed cycles.

Assume for a contradiction that $\mathsf{IMP}(\mathsf{SUB}(\mathcal{J}^*; h))$ has a cycle C. Let $\mathcal{J}^{*\mathsf{res}}$ be an instance obtained by restricting all component of $\mathcal{J}^*$ on V_{B}. From the definition of the implication graph, for each vertex $v[i]$ in C, there exists a solution f' for $\mathsf{SUB}(\mathcal{J}^*; h)$ such that $f'(v) = i$. By Lemma 9.5, a mapping f such that $f|_{V_{\mathsf{N}}} = h$ and $f|_{V_{\mathsf{B}}} = f'$ is a solution for $\mathcal{J}^*$. Moreover, $f|_{V_{\mathsf{B}}} = f'$ is a solution for the restricted instance $\mathcal{J}^{*\mathsf{res}}$. Therefore, $\mathsf{IMP}(\mathcal{J}^{*\mathsf{res}})$ has a vertex $v[i]$. For each arc $v[i] \to w[j]$, a mapping $(i, \neg j)$ is not contained in the constraint $\mathcal{C}^{*\prime}(vw)$ of vw in $\mathsf{SUB}(\mathcal{J}^*; h)$. Recall that $\mathcal{C}^{*\prime}(vw)$ is the mapping set $\bigcap_{X \in E'} \mathcal{G}(X)$, where $E' = \{X \in E(G^*) \colon X \setminus V_{\mathsf{N}} = \{v, w\}\}$ and $\mathcal{G}(X) = \{g|_{\{v,w\}} \colon g \in \mathcal{C}(X), h \text{ and } g \text{ are compatible}\}$. Since G^* has a hyperedge of size at most two, E' contains exactly one edge vw.

Therefore, $\mathcal{C}^{*\prime}(vw) = \mathcal{C}^{*}(vw)$ holds, and hence $(i, \neg j)$ not contained in $\mathcal{C}^{*}(vw)$. Moreover, the constraint $\mathcal{C}^{*\mathsf{res}}(vw)$ does not contain $(i, \neg j)$, too. From the definition, $\mathsf{IMP}(\mathcal{J}^{*\mathsf{res}})$ contains the arc $v[i] \to w[j]$. By the above observations, $\mathsf{IMP}(\mathcal{J}^{*})$ has a directed cycle C, which contradicts that we have eliminated all directed cycles from the implication graph by the preprocessing.

Thus, we have proved Lemma 9.7 and hence Theorem 9.5.

9.3.4 Discussion

We conclude this section by showing the following proposition, which states that we can eliminate k from the parameter when restricted to CR

Proposition 9.3 *There exists a fixed-parameter algorithm for* CR *parameterized by* nb *which runs in time* $O^{*}(\mathsf{nb}^{O(\mathsf{nb})})$.

Proof. Let $\mathcal{I} = (G, \mathcal{C}, D, f_s, f_t)$ be an instance of CR. We can assume that $k \leq |V(G)|$ and $k \geq 3$, since otherwise it is trivial. Because a list of each vertex is exactly D in CR and $|D| = k \geq 3$, $|V(G)| = \mathsf{nb}$ holds. Therefore, we can solve it in time $O^{*}(\mathsf{nb}^{O(\mathsf{nb})})$ by Theorem 2.1. □

9.4 ETH-based lower bound

Finally, we show the following theorem.

Theorem 9.6 *Under ETH, there exists no algorithm solving* 2-CSR *in time* $O^{*}((k+n)^{o(k+n)})$.

Proof. We give a polynomial-time reduction from $(\kappa \times \kappa)$-CLIQUE, which is defined as follows. An instance of $(\kappa \times \kappa)$-CLIQUE is a graph H with the vertex set $\{u_i^p \colon 1 \leq i, p \leq \kappa\}$; we denote $U_i = \{u_i^p \colon 1 \leq p \leq \kappa\}$ for each $i \in \{1, 2, \ldots, \kappa\}$. Then, the problem asks whether there exists a clique $Q \subseteq V(H)$ such that $|Q \cap U_i| = 1$ for every

$i \in \{1, 2, \ldots, \kappa\}$. It is known that there exists no algorithm solving $(\kappa \times \kappa)$-CLIQUE in time $O^*(\kappa^{o(\kappa)})$ under ETH [20, Theorem 14.12]. We will present a polynomial-time transformation from an instance H of $(\kappa \times \kappa)$-CLIQUE to an instance $\mathcal{I}$ of 2-CSR such that

- a graph has $\kappa + 2$ vertices and a domain has $\kappa + 1$ values; and
- H is a yes-instance if and only if $\mathcal{I}$ is.

If there exists an algorithm A solving 2-CSR in time $O^*((k+n)^{o(k+n)})$, an execution of A for the transformed instance $\mathcal{I}$ yields an algorithm solving $(\kappa \times \kappa)$-CLIQUE in time

$$O^*((k+n)^{o(k+n)}) = O^*((2\kappa+3)^{o(2\kappa+3)}) = O^*((\kappa^2)^{o(2\kappa+3)}) = O^*(\kappa^{o(\kappa)}).$$

Before constructing $\mathcal{I}$, we first reformulate $(\kappa \times \kappa)$-CLIQUE as a special case of 2-ARY CONSTRAINT SATISFIABILITY by a similar idea of the proof of Theorem 7.1. Let G' be a complete graph K_κ with κ vertices $\{v_1, v_2, \ldots, v_\kappa\}$, and let $D' = \{1, 2, \ldots, \kappa\}$. We construct each constraint so that assigning a value $p \in D$ to a vertex $v_i \in V(G')$ corresponds to choosing a vertex u_i^p as a member of a clique Q. That is, we define $\mathcal{C}'(v_iv_j) := \{(p, q) \colon u_i^p u_j^q \in E(H)\}$. Observe that we can simultaneously assign p and q to v_i and v_j, respectively, if and only if u_i^p and u_j^q are adjacent in H. Therefore, H and $(G', D', \mathcal{C}')$ are equivalent. Clearly, $|V(G')| = |D'| = \kappa$ holds.

We now construct an instance $\mathcal{I} = (G, D = D' \cup \{0\}, \mathcal{C}, f_s, f_t)$ of 2-CSR as follows. A graph G is obtained from G' by adding two new vertices w_1 and w_2 and edges $\{w_1w \colon w \in V(G') \cup \{w_2\}\}$. These added vertices will form a key component which links the existence of a solution of $(G', D', \mathcal{C}')$ with the reconfigurability of $\mathcal{I}$. Clearly, $|V(G)| = \kappa + 2$ and $|D| = \kappa + 1$ hold. We first construct the constraints of the original graph G'. For each edge $v_iv_j \in E(G')$, we add to a constraint $\mathcal{C}'(v_iv_j)$, all mappings which contain 0; that is, $\mathcal{C}(v_iv_j) := \mathcal{C}'(v_iv_j) \cup (\{0\} \times D) \cup (D \times \{0\})$. We have the following observation.

Observation 9.1 *A solution of* $(G', D, \mathcal{C})$ *in which no vertices are assigned* 0 *one-to-one corresponds to a solution of the original instance* $(G', D', \mathcal{C}')$.

We next define the constraints regarding w_1 and w_2. The constraint $\mathcal{C}(w_1w_2)$ of $w_1w_2 \in E(G)$ is defined as $\mathcal{C}(w_1w_2) := \{(0,1),(0,2),(1,2),(2,1)\}$. For each $v_i \in V(G')$, we let $\mathcal{C}(w_1v_i) = D^2 \setminus \{(0,0)\}$. Then, we have the following observation.

Observation 9.2 *we can assign* 0 *to* w_1 *if and only if no other vertices are assigned* 0.

Finally, we define two solutions f_s and f_t as follows:

- for each vertex $v_i \in V(G')$, let $f_s(v_i) = f_t(v_i) = 0$; and
- let $f_s(w_1) = f_t(w_2) = 1$ and $f_s(w_2) = f_t(w_1) = 2$.

In order to show the correctness, it suffices to show the following lemma.

Lemma 9.9 $(G', D', \mathcal{C}')$ *has a solution if and only if* $\mathcal{I}$ *is a yes-instance.*

We first show the if direction. Assume that f_s and f_t are reconfigurable. Then, a reconfiguration sequence must contain a solution f such that $f(w_1) = 0$. Otherwise, values of w_1 and w_2 can never be changed because only allowed assignment to $\{w_1, w_2\}$ is either $(1,2)$ or $(2,1)$ in this case, which contradicts the reconfigurability. Since f assigns 0 to w_1, no other vertices are assigned 0 by Observation 9.2. In addition, by Observation 9.1, $f|_{V(G')}$ is a solution of $(G', D', \mathcal{C}')$.

We next prove the only-if direction. Let g be a solution of $(G', D', \mathcal{C}')$. We extend it to solutions f'_s and f'_t of $(G, D, \mathcal{C})$ as follows. For each $r \in \{s, t\}$,

- let $f'_r(v_i) = g(v_i)$ for each $v_i \in V(G')$; and
- let $f'_r(w_1) = f_r(w_1)$ and $f'_r(w_2) = f_r(w_2)$.

Notice that f'_r is a solution, and that f_r and f'_r are reconfigurable by changing values of all vertices $v_i \in V(G')$ from 0 to $f'_r(v_i)$. Furthermore, f'_s and f'_t are reconfigurable by the following three steps:

- change a value of w_1 from 1 to 0;
- change a value of w_2 from 2 to 1; and
- change a value of w_1 from 0 to 2.

We note that this yields a valid reconfiguration sequence: in particular, Observation 9.2 and the construction of f'_s justify the first step. Therefore, f_s and f_t are reconfigurable. □

Chapter 10 Conclusions

In this thesis, we studied CSR and its spacial cases, especially 3-CSR, 2-CSR, (L)HR and (L)CR from the viewpoints of polynomial-time solvability and parameterized complexity, and gave several interesting boundaries of tractable and intractable cases.

Part I

In Chapter 3, we studied CR from the viewpoint of graph classes. We first showed that the problem is PSPACE-complete even for chordal graphs and thus answered the open question posed by Bonsma and Paulusma [7]. On the other hand, we then showed that the problem is polynomial-time solvable for 2-degenerate graphs and gave linear-time algorithms for subclasses of chordal graphs, that is, q-trees with any integer $q \geq 1$, split graphs, and trivially perfect graphs.

In Chapter 4, we studied LCR from the viewpoint of graph classes. We first showed that the problem is PSPACE-complete even for threshold graphs, which have the modular-width zero. On the other hand, we gave a polynomial-time algorithm for graphs with pathwidth one using the method of dynamic programming. We note that it is known that it is PSPACE-complete for graphs with pathwidth two [60], and hence, our algorithm gave a sharp boundary of the complexity of LCR with respect to the pathwidth.

In Chapter 5, we studied LHR. We first showed the PSPACE-completeness of LHR for graphs with pathwidth one; and hence, the above algorithm for LCR is unlikely to be extended to LHR. On the other hand, we gave a polynomial time algorithm for $k = 3$.

In Chapter 6, we studied 2-CSR from the viewpoint of the size k of a domain. We showed the PSPACE-completeness of 2-CSR for $k = 3$ and gave a polynomial time algorithm for $k = 2$; these two results shows the boundary of the complexity of 2-CSR with respect to k. Indeed, in the proof of the first hardness result, there are essentially two types of constraints between vertices; the constraint corresponding LCR and the trivial one. Recall that LHR, in which all edge have the essentially same constraint corresponding to a given underlying graph, can be solved in polynomial-time for $k = 3$.

Part II

In Chapter 7, we showed that HR parameterized by the number n of vertices and LCR parameterized by vc are both $W[1]$-hard. These imply that fixed-parameter algorithms are unlikely to exist for almost all graph parameters in Figure 1.4.

In Chapter 8, we first gave some useful lemma, which we call the reduction rule, for general CSR. We then gave a fixed-parameter algorithm for LHR parameterized by $k + \mathsf{mw}$ by kernelizing an input instance using the reduction rule.

In Chapter 9, we first gave a fixed-parameter algorithm for CSR parameterized by $k + \mathsf{td}$, by again using the reduction rule. By the relationship between vc and td, this implies that CSR is fixed-parameter tractable when parameterized by $k + \mathsf{vc}$. However, we gave two stronger results; that is, we gave an algorithm for the shortest variant and a faster algorithm. We also gave a fixed-parameter algorithm for 2-CSR parameterized by $k + \mathsf{nb}$; this result generalizes the polynomial-time solvability of 2-CSR for $k = 2$. We finally showed the lower bound of the computation time for 2-CSR under the well-known exponential time hypothesis. This lower bound matches the running times of some of our algorithms.

Finally, we discuss a future direction. In 2014, Mouawad et al. [48] gave a meta-theorem to develop a fixed-parameter algorithm for the shortest variant of a reconfiguration problem on graphs when parameterized by the length ℓ of a recon-

figuration sequence and the treewidth. We then have one question whether such a meta-theorem for some parameters which do not contain ℓ can be given. Note that many reconfiguration problems are fixed-parameter intractable when parameterized only by the treewidth; recall for example that LCR is PSPACE-complete even for graphs with constant bandwidth and hence with constant treewidth [60]. Therefore, generalizing fixed-parameter algorithms parameterized by $k + \mathsf{mw}$, $k + \mathsf{td}$ and $k + \mathsf{vc}$ may be an interesting direction.

Bibliography

[1] M. Bonamy and N. Bousquet. Recoloring graphs via tree decompositions. *European Journal of Combinatorics*, 69:200–213, 2018.

[2] M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27(1):132–143, 2014.

[3] P. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017.

[4] P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.

[5] P. Bonsma, M. Kamiński, and M. Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2014)*, volume 8503 of *Lecture Notes in Computer Science (LNCS)*, pages 86–97, 2014.

[6] P. Bonsma, A. E. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2014)*, pages 110–121, 2014.

[7] P. Bonsma and D. Paulusma. Using contracted solution graphs for solving reconfiguration problems. In *41st International Symposium on Mathematical*

Foundations of Computer Science (MFCS 2016), volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:15, 2016.

[8] N. Bousquet and G. Perarnau. Fast recoloring of sparse graphs. *European Journal of Combinatorics*, 52:1–11, 2016.

[9] A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.

[10] R. C. Brewster, J.-B. Lee, B. Moore, J. A. Noel, and M. Siggers. Graph homomorphism reconfiguration and frozen H-colourings, arXiv:1712.00200, 2017.

[11] R. C. Brewster, J.-B. Lee, and M. Siggers. Recolouring reflexive digraphs. *Discrete Mathematics*, 341(6):1708–1721, 2018.

[12] R. C. Brewster, S. McGuinness, B. Moore, and J. A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theoretical Computer Science*, 639:1–13, 2016.

[13] R. C. Brewster and J. A. Noel. Mixing homomorphisms, recolorings, and extending circular precolorings. *Journal of Graph Theory*, 80(3):173–198, 2015.

[14] J. Cardinal, E. D. Demaine, D. Eppstein, R. A. Hearn, and A. Winslow. Reconfiguration of satisfying assignments and subset sums: Easy to find, hard to connect. In *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, Lecture Notes in Computer Science (LNCS), pages 365–377, 2018.

[15] M. Celaya, K. Choo, G. MacGillivray, and K. Seyffarth. Reconfiguring k-colourings of complete bipartite graphs. *Kyungpook Mathematical Journal*, 56:647–655, 2016.

[16] L. Cereceda. *Mixing Graph Colourings*. PhD thesis, The London School of Economics and Political Science, 2007.

[17] L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.

[18] G. Chen and A. Saito. Graphs with a cycle of length divisible by three. *Journal of Combinatorial Theory, Series B*, 60(2):277–292, 1994.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[20] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Lower bounds based on the exponential-time hypothesis. In *Parameterized Algorithms*, pages 467–521. Springer International Publishing, 2015.

[21] E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015.

[22] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[23] M. Dyer, A. D. Flaxman, A. M. Frieze, and E. Vigoda. Randomly coloring sparse random graphs with fewer colors than the maximum degree. *Random Structures & Algorithms*, 29(4):450–465, 2006.

[24] C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of brooks' theorem and its consequences. *Journal of Graph Theory*, 83(4):340–358, 2016.

[25] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, 1967.

[26] P. Gopalan, P. G. Kolaitis, E. N. Maneva, and C. H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.

[27] R. Haas and G. MacGillivray. Connectivity and hamiltonicity of canonical colouring graphs of bipartite and complete multipartite graphs. *Algorithms*, 11(4), 2018.

[28] R. Haas and K. Seyffarth. The k-dominating graph. *Graphs and Combinatorics*, 30(3):609–617, 2014.

[29] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[30] A. Haddadan, T. Ito, A. E. Mouawad, N. Nishimura, H. Ono, A. Suzuki, and Y. Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016.

[31] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[32] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, 2005.

[33] J. van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.

[34] 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):1054–1065, 2011.

[35] T. Ito, M. Kamiński, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka. On the parameterized complexity for token jumping on graphs. In *Proceedings of the 11th Annual Conference on Theory and Applications of Models of Computation (TAMC 2014)*, Lecture Notes in Computer Science (LNCS), pages 341–351, 2014.

[36] T. Ito, K. Kawamura, and X. Zhou. An improved sufficient condition for reconfiguration of list edge-colorings in a tree. *IEICE Transactions on Information and Systems*, 95-D(3):737–745, 2012.

[37] T. Ito, H. Nooka, and X. Zhou. Reconfiguration of vertex covers in a graph. *IEICE Transaction on Information and Systems*, 99-D(3):598–606, 2016.

[38] T. Ito and A. Suzuki. Web survey on combinatorial reconfiguration. `http://www.ecei.tohoku.ac.jp/alg/coresurvey/`. Updated on November 9, 2017.

[39] M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016.

[40] M. Kamiński, P. Medvedev, and M. Milanič. Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210, 2011.

[41] M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

[42] D. Lokshtanov and A. E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 185–195, 2018.

[43] K. Makino, S. Tamaki, and M. Yamamoto. On the Boolean connectivity problem for horn relations. *Discrete Applied Mathematics*, 158(18):2024–2030, 2010.

[44] K. Makino, S. Tamaki, and M. Yamamoto. An exact algorithm for the Boolean connectivity problem for k-CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011.

[45] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[46] R. M. McConnell and F. de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.

[47] A. E. Mouawad, N. Nishimura, V. Pathak, and V. Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. *SIAM Journal on Discrete Mathematics*, 31(3):2185–2200, 2017.

[48] A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2014)*, volume 8894 of *Lecture Notes in Computer Science (LNCS)*, pages 246–257, 2014.

[49] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2006.

[50] N. Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

[51] H. Osawa, A. Suzuki, T. Ito, and X. Zhou. Complexity of coloring reconfiguration under recolorability constraints. In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:12, 2017.

[52] H. Osawa, A. Suzuki, T. Ito, and X. Zhou. The complexity of (list) edge-coloring reconfiguration problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 101-A(1):232–238, 2018.

[53] A. Proskurowski and J. A. Telle. Classes of graphs with restricted interval models. *Discrete Mathematics and Theoretical Computer Science*, 3(4):167–176, 1999.

[54] F. Reidl, P. Rossmanith, F. S. Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, pages 931–942, 2014.

[55] N. Robertson and P. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[56] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226, 1978.

[57] K. W. Schwerdtfeger. A computational trichotomy for connectivity of Boolean satisfiability. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(3–4):173–195, 2014.

[58] M. Tedder, D. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP 2008)*, pages 634–645, 2008.

[59] M. Wrochna. Homomorphism reconfiguration via homotopy. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 730–742, 2015.

[60] M. Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018.

List of papers

Refereed papers in journals

1. Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes, *IEICE Transactions on Information and Systems*, E102-D(3), 2019, to appeared.

2. Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Parameterized complexity of the list coloring reconfiguration problem with graph parameters, *Theoretical Computer Science*, 739:65–79, 2018.

3. Tatsuhiko Hatanaka, Takehiro Ito and Xiao Zhou, The list coloring reconfiguration problem for bounded pathwidth graphs, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E98-A(6):1168–1178, 2015.

Refereed papers in international conferences

1. Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The coloring reconfiguration problem on specific graph classes, In *Proceedings of the 11th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2017)*, volume 10627 of *Lecture Notes in Computer Science (LNCS)*, pages 152–162, 2017.

2. Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Parameterized complexity of the list coloring reconfiguration problem with graph parameters, In *Pro-*

ceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017), volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:13, 2017.

3. Tatsuhiko Hatanaka, Takehiro Ito and Xiao Zhou, The list coloring reconfiguration problem for bounded pathwidth graphs, In *Proceedings of the 8th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2014)*, volume 8881 of *Lecture Notes in Computer Science (LNCS)*, pages 314–328, 2014.