On Reconfiguration Problems: Structure and Tractability

by

Amer E. Mouawad

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2015

Abstract

Given an n-vertex graph G and two vertices s and t in G, determining whether there exists a path and computing the length of the shortest path between s and t are two of the most fundamental graph problems. In the classical battle of P versus NP or "easy" versus "hard", both of these problems are on the easy side. That is, they can be solved in $poly(n)$ time, where *poly* is any polynomial function. But what if our input consisted of a 2^n-vertex graph? Of course, we can no longer assume G to be part of the input, as reading the input alone requires more than $poly(n)$ time. Instead, we are given an oracle encoded using $poly(n)$ bits and that can, in constant or $poly(n)$ time, answer queries of the form "is u a vertex in G" or "is there an edge between u and v?". Given such an oracle and two vertices of the 2^n-vertex graph, can we still determine if there is a path or compute the length of the shortest path between s and t in $poly(n)$ time?

A slightly different, but equally insightful, formulation of the question above is as follows. Given a set S of n objects, consider the graph $R(S)$ which contains one vertex for each set in the power set of S, 2^S, and two vertices are adjacent in $R(S)$ whenever the size of their symmetric difference is equal to one. Clearly, this graph contains 2^n vertices and can be easily encoded in $poly(n)$ bits using the oracle described above. It is not hard to see that there exists a path between any two vertices of $R(S)$. Moreover, computing the length of a shortest path can be accomplished in constant time; it is equal to the size of the symmetric difference of the two underlying sets. If the vertex set of $R(S)$ were instead restricted to a subset of 2^S, both of our problems can become NP-complete or even PSPACE-complete. Therefore, another interesting question is whether we can determine what types of "restriction" on the vertex set of $R(S)$ induce such variations in the complexity of the two problems.

These two seemingly artificial questions are in fact quite natural and appear in many practical and theoretical problems. In particular, these are exactly the types of questions asked under the reconfiguration framework, the main subject of this thesis. Under the reconfiguration framework, instead of finding a feasible solution to some instance $\mathcal{I}$ of a search problem $\mathcal{Q}$, we are interested in structural and algorithmic questions related to the solution space of $\mathcal{Q}$. Naturally, given some adjacency relation $\mathcal{A}$ defined over feasible solutions of $\mathcal{Q}$, size of the symmetric difference being one such relation, the solution space can be represented using a graph $R_{\mathcal{Q}}(\mathcal{I})$. $R_{\mathcal{Q}}(\mathcal{I})$ contains one vertex for each feasible solution of $\mathcal{Q}$ on instance $\mathcal{I}$ and two vertices share an edge whenever their corresponding solutions are adjacent under $\mathcal{A}$. An edge in $R_{\mathcal{Q}}(\mathcal{I})$ corresponds to a reconfiguration step, a walk in $R_{\mathcal{Q}}(\mathcal{I})$ is a sequence of such steps, a reconfiguration sequence, and $R_{\mathcal{Q}}(\mathcal{I})$ is a reconfiguration graph. Studying problems related to reconfiguration graphs has received considerable attention in recent literature, the most popular problem being to determine whether there exists a reconfiguration sequence between two given feasible solutions; for most NP-complete problems, this problem has been shown to be PSPACE-complete.

The purpose of our work is to embark on a systematic investigation of the tractability and structural properties of such problems under both classical and parameterized complexity assumptions. Parameterized complexity is another framework which has become an essential tool for researchers in computational complexity during the last two decades or so and one of its main goals is to provide a better explanation of why some hard problems (in a classical sense) can be in fact much easier than others. Hence, we are interested in what separates the tractable instances from the intractable ones and the fixed-parameter tractable instances from the fixed-parameter intractable ones. It is clear from the generic definition of reconfiguration problems that several factors affect their complexity status. Our work aims at providing a finer classification of the complexity of reconfiguration problems with respect to some of these factors, including the definition of the adjacency relation $\mathcal{A}$, structural properties of the input instance $\mathcal{I}$, structural properties of the reconfiguration graph, and the length of a reconfiguration sequence. As most of these factors can be numerically quantified, we believe that the investigation of reconfiguration problems under both parameterized and classical complexity assumptions will help us further understand the boundaries between tractability and intractability.

We consider reconfiguration problems related to Satisfiability, Coloring, Dominating Set, Vertex Cover, Independent Set, Feedback Vertex Set, and Odd Cycle Transversal, and provide lower bounds, polynomial-time algorithms, and fixed-parameter tractable algorithms. In doing so, we answer some of the questions left open in recent work and push the known boundaries between tractable and intractable even closer. As a byproduct of our initiating work on parameterized reconfiguration problems, we present a generic adaptation of parameterized complexity techniques which we believe can be used as a starting point for studying almost any such problem.

Acknowledgements

First of all, I would like to thank my supervisor Naomi Nishimura for her help, her patience, and her guidance, both with my research and with all other aspects of being a graduate student. The path (not a shortest one) that lead to this thesis has been quite an interesting one and it would not have been possible without her.

Already in advance, I would like to thank my committee members, Therese Biedl, Henning Fernau, J. Ian Munro, and Stephen Vavasis.

I am very grateful to Venkatesh Raman and Saket Saurabh for their continuous support, for generously sharing their time and expertise, and for inviting me (respectively) to the Institute of Mathematical Sciences (Chennai, India) and to the University of Bergen (Bergen, Norway), where I got to meet great people (and researchers) and explore new ideas.

Paul Bonsma, Khuzaima Daudjee, Arash Hadadan, Takehiro Ito, Daniel Lokshtanov, Neeldhara Misra, Hirotaka Ono, Fahad Panolan, Vinayak Pathak, M.S. Ramanujan, Narges Simjour, Akira Suzuki, Youcef Tebbal, and Marcin Wrochna, it was a pleasure getting to know you and working with you.

A big thank you to the Algorithms & Complexity group at the University of Waterloo and a special thank you goes to Wendy Rush.

Last but not least, I would like to thank Faisal N. Abu-Khzam for his continuous support and for pushing me towards a PhD.

Amer Mouawad
December 2014

To my father...

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many classical one-player games can be formulated as *reconfiguration problems* in which we are given a collection of *configurations* together with some transformation rules that allow us to transform one configuration into another. For example, in the n-puzzle game (Figure 1.1), n tiles numbered from 1 to n are arranged on an $m \times m$ grid, $m^2 = n + 1$, leaving one empty square. A tile can only move to the empty square if it is adjacent to it, i.e. above it, below it, to its left, or to its right. Starting from any *source configuration*, the goal is to reach the configuration in which all numbers are in order, the *target configuration*.

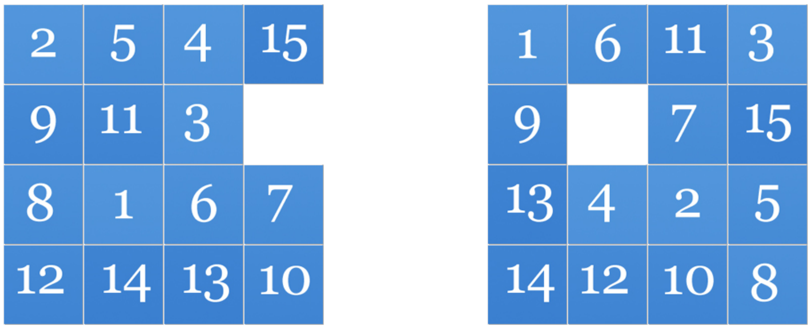

Figure 1.1: Two configurations of the 15-puzzle (source: http://www.bewitchedgames.com © 2014 Bewitched Games)

Closely related to the n-puzzle game is Rush Hour. The game is played on an $m \times m$ grid, on which a number of cars are placed, one of them being the car that wishes to escape out of the grid. The cars occupy two or three squares and can only move forwards or backwards (but not sideways) (Figure 1.2). The aim is to move the cars in such a way that the red car can be driven out of the exit.

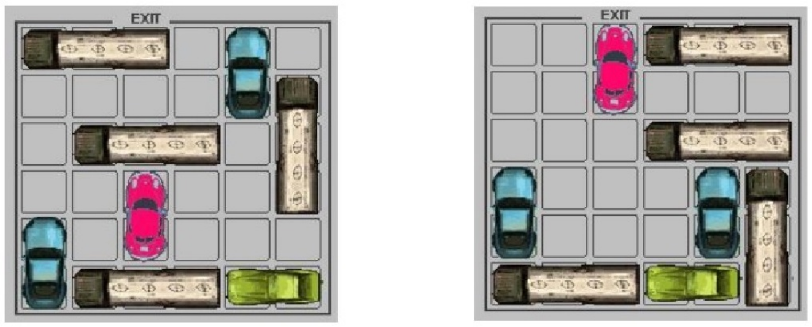

Figure 1.2: A 6×6 Rush Hour instance and solution (source: http://www.puzzles.com/products/rushhour/rushhourapp.htm © 2003–2011 ThinkFun Inc.)

Another popular example of a reconfiguration problem originating from one-player games is Rubik's cube (Figure 1.3). Here, given any configuration of the cube, the goal is to reach the configuration in which each of the six sides of the cube is *monochromatic*, i.e. of the same color. A transformation of the cube from one configuration into another consists of a rotation of one of the sides of the cube.

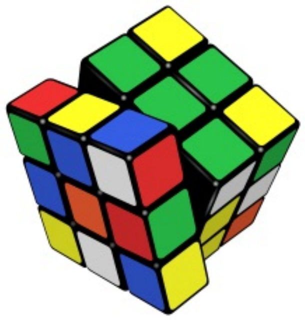

Figure 1.3: Rubik's cube (source: http://en.wikipedia.org/wiki/rubik's_cube © 2006 Booyabazooka)

More reconfiguration problems emerge from classical computational problems in different areas such as graph theory, computational geometry, artificial intelligence, and more recently quantum computing. A practical example is network security. Consider a network graph where each vertex represents an online server and edges correspond to communication links (Figure 1.4). Knowing that users u_s and u_t are malicious users sending illegal data to each other over the network, a security officer wishes to intercept any transmissions between the two at all times. To do so, the officer is given access to at most k network servers to install "firewalls" such that all communication routes between u_s and u_t are captured. After a while, the security officer realizes that fewer firewalls are needed to intercept all transmissions. And, since firewalls are a major drawback to server performance, the goal now is to change the locations of the older firewalls (one at a time) without (at any time) opening an unsecured transmission route between u_s and u_t nor having more than k firewalls installed. The security problem, as the reader might have guessed, is a reconfiguration variant of the *(u_s,u_t)-cut problem*. Note that in the example graph shown in Figure 1.4, if we assume $k = 4$ then the security officer is stuck with the initial choice of servers (gray vertices in Figure 1.4) since both black vertices need to have a firewall installed before uninstalling any of the other three firewalls.

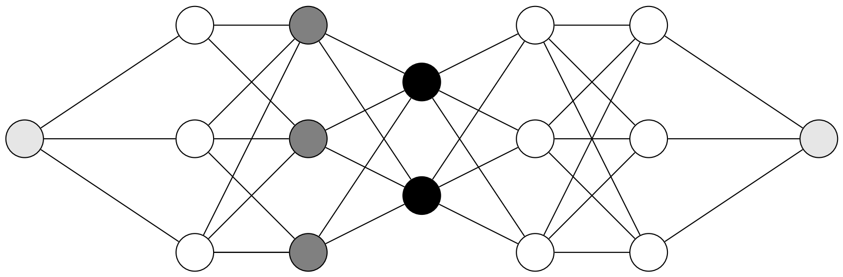

Figure 1.4: Example of a network graph with servers shown in white and malicious users shown in light gray. Dark gray servers correspond to the source configuration and black servers are the target configuration.

Viewing reconfiguration problems from a graph-theoretic perspective, the notion of a *reconfiguration graph* naturally arises. The node set of this graph consists of all possible configurations and two nodes share an edge whenever the corresponding configurations can each be obtained from the other by the application of a single transformation rule, a *reconfiguration step*. Any path or walk in the reconfiguration graph corresponds to a sequence of such steps, a *reconfiguration sequence*. Studying reconfiguration graphs is the main subject of this thesis. In fact, almost any search problem can have a corresponding reconfiguration variant. That is, instead of finding a feasible solution/configuration to some instance $\mathcal{I}$ of a search problem $\mathcal{Q}$, we are interested in structural and algorithmic questions related to the solution/configuration space of $\mathcal{Q}$. Generally, defining a reconfiguration problem requires three ingredients:

(1) A search problem $\mathcal{Q}$,

(2) an optional range, $[r_l, r_u]$, bounding some numerically quantifiable property Ψ of feasible solutions for $\mathcal{Q}$, and

(3) an adjacency relation $\mathcal{A}$, usually symmetric and polynomially-testable, on the set of feasible solutions of $\mathcal{Q}$.

With this in hand, we can construct the reconfiguration graph $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$ for each instance $\mathcal{I}$ of $\mathcal{Q}$. The nodes of $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$ correspond to the feasible solutions of $\mathcal{Q}$ having $r_l \leq \Psi \leq r_u$, and there is an edge between two nodes whenever the corresponding solutions are adjacent under $\mathcal{A}$. $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$ provides a structured view of the solution space of $\mathcal{Q}$ with respect to instance $\mathcal{I}$, adjacency relation $\mathcal{A}$, and property Ψ. There are four general types of questions which arise under the *reconfiguration framework*:

(1) $\mathcal{Q}$-Reach is the problem of determining whether two given feasible solutions belong to the same connected component of $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$.

(2) $\mathcal{Q}$-Bound is the problem of finding a shortest path or a path of length ℓ between two given feasible solutions in $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$.

(3) $\mathcal{Q}$-Conn is the problem of determining whether $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$ is connected.

(4) $\mathcal{Q}$-Diam is the problem of determining the diameter of $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$ or of its connected components.

We refer to $\mathcal{Q}$-Reach and $\mathcal{Q}$-Bound as the *reachability* and *bounded reachability* variants/problems of $\mathcal{Q}$, respectively, and we refer to all four problems as *reconfiguration problems* for $\mathcal{Q}$. To get a sense of what one might expect when studying reconfiguration problems, we go back to the n-puzzle example. The number of possible configurations of the 24-puzzle is $\frac{25!}{2} \approx 7.76 \times 10^{24}$. Clearly, constructing the full reconfiguration graph for solving the reachability or bounded reachability problem is not an option. Already in 1879, Johnson and Story [88] showed that half of the starting positions for the n-puzzle are impossible to solve, irrespective of the number moves. This implies that the reconfiguration graph of the n-puzzle game is not connected. Since there are at most four configurations "reachable" from any given configuration in a single step, one possible attempt to solve the problem is to construct what is known as the *breadth-first search-tree* [95], or *BFS-tree* for short. Initially, the BFS-tree consists of a single *root* node which corresponds to the source configuration. Building the rest of the tree is accomplished recursively using a queue as follows. First, we enqueue the root node, or equivalently the source configuration. While the queue is not empty, we dequeue a node. If this node corresponds to the target configuration, the construction ends. Otherwise, we generate the (at most four) configurations reachable from the dequeued configuration and add them to the queue, and respectively the tree, after making sure they have not been added before. Eventually, we will either reach the target configuration or exhaust all configurations reachable from the source. However, there are two crucial requirements for making this approach "computationally feasible": the target configuration must be reachable from the source configuration and the number of steps required to reach it, say ℓ, must be known. If both requirements are guaranteed and ℓ is relatively small, then we can bound the size of the BFS-tree, i.e. the number of configurations to consider, which is at most $4^{\ell+1}$. Unfortunately, even for the simple n-puzzle game, computing ℓ (or equivalently solving the bounded reachability problem) is computationally hard and the BFS-tree approach fails. As we shall see later, reachability and bounded reachability problems are, in most cases, "really hard" and sometimes surprisingly so.

Games and puzzles have been extensively studied in the literature [10, 38, 49, 58, 60, 88, 92] but new types of reconfiguration problems have also recently gained popularity. We give a detailed overview of some of these results and their motivation in Section 1.2. Before doing so, we illustrate a few examples in Section 1.1 to familiarize the reader with the problems considered throughout this work.

1.1 Examples

Graph vertex-subset problems. In a graph vertex-subset problem, given some graph G, the goal is to find a subset of the vertices of G which satisfies certain properties. Let us consider the classical VERTEX COVER problem, formally defined as follows:

VERTEX COVER (VC)
Input: A graph G with vertex set $V(G)$ and edge set $E(G)$ and an integer k.
Question: Is there a set $S \subseteq V(G)$, a *vertex cover* of G, such that $|S| \leq k$ and every edge in $E(G)$ is incident to some vertex in S?

In this case, our search problem $\mathcal{Q}$ is VC. We fix Ψ to be the size of a feasible solution and set $[r_l, r_u]$ to $[0, k]$. As a symmetric adjacency relation, $\mathcal{A}$, we let two solutions be adjacent whenever they differ in exactly one vertex. The resulting reconfiguration graph, $R_{VC}(G, 0, k)$, has one vertex for every vertex cover of G of size between 0 and k, and two vertices share an edge in $R_{VC}(G, 0, k)$ whenever the corresponding solutions differ in exactly

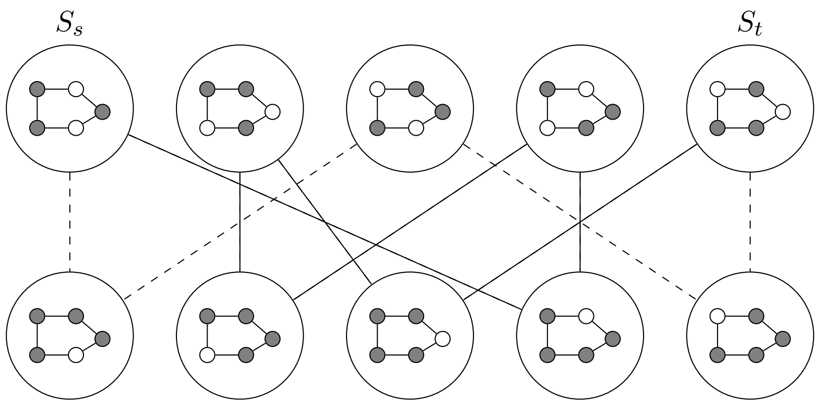

Figure 1.5: A reconfiguration sequence of length 6 (shown in solid lines) from S_s to S_t in $R_{VC}(C_5, 0, 4)$.

one vertex. Figure 1.5 illustrates the reconfiguration graph $R_{VC}(C_5, 0, 4)$ for an input graph G consisting of a cycle on five vertices, a C_5, and $k = 4$. Every feasible solution of size less than or equal to four is colored gray. Note that since the size of a minimum cardinality vertex cover of a C_5 is 3, $R_{VC}(C_5, 0, 4) = R_{VC}(C_5, 1, 4) = R_{VC}(C_5, 2, 4) = R_{VC}(C_5, 3, 4)$. The fact that $R_{VC}(C_5, 0, 4)$ is connected implies that a reconfiguration sequence exists between any two feasible solutions of this instance. If we set $k = 3$ instead, $R_{VC}(C_5, 0, 3)$ would consist of five isolated vertices (the top five vertices in Figure 1.5). Note that the bounds enforced on the size of feasible solutions greatly affect the complexity of both the reachability and bounded reachability questions. For instance, if we set $[r_l, r_u]$ to $[0, |V(G)|]$ in the case of VC then $R_{VC}(G, 0, |V(G)|)$ is always connected and of linear diameter for any graph G; $R_{VC}(G, 0, |V(G)|)$ would contain a "central" node corresponding to the vertex cover of size $|V(G)|$, which is connected to every other vertex cover S by a path of length $|V(G)| - |S|$.

There are other natural adjacency relations which can be considered when dealing with graph vertex-subset problems. If we assume that every vertex in the input graph can be either *occupied* by a single token or *free*, then we can view any solution to such a problem as a collection of tokens placed on a subset of the vertices of the input graph. When viewed as such, we can define a change in a solution by either a token jump, a token slide, or a token addition/removal. Under the *token jumping (TJ)* model [91], a reconfiguration step consists of jumping a token from some occupied vertex to some free vertex. Under the *token sliding (TS)* model [18, 77, 91], instead of jumping a token to any free vertex of the input graph, a token is only allowed to slide to some free vertex along the edges incident to the vertex it occupies. Finally, the *token addition/removal (TAR)* model [81, 91], which corresponds to the Vertex Cover example discussed above, allows for either the addition of a token on some free vertex or the removal of a token from some occupied vertex. Since every vertex can be occupied by a single token, the solution size under the TJ and TS models never changes and hence $r_l = r_u$. Unless stated otherwise, we will primarily deal with the TAR model and sometimes mention the relationships to the other two models.

Coloring. Given a graph G and a set of k colors, a *k-color assignment* of G maps each vertex of G to a color in $\{1, \ldots, k\}$. A k-color assignment is said to be a *proper k-coloring*, or *k-coloring* for short, if no two adjacent vertices receive the same color. The Coloring problem is formally defined as follows:

Coloring (COL)
Input: A graph G with vertex set $V(G)$ and edge set $E(G)$ and an integer k.
Question: Does G admit a proper k-coloring?

In contrast to the Vertex Cover problem considered above, a feasible solution to the Coloring problem assigns a color to each vertex of the input graph. Hence, in this case we let Ψ correspond to the number of colors in a feasible solution, which might be less than k since any proper k'-coloring, $k' < k$, is also a proper k-coloring. The reconfiguration graph $R_{COL}(G, 0, k)$ for a graph G therefore contains all proper k-colorings of G as its node set and two nodes are adjacent whenever the corresponding colorings differ on exactly one vertex. An example of a reconfiguration sequence between two proper 4-colorings of a graph is given in Figure 1.6. Even though the first (top-left node in Figure 1.6) and last (bottom-left node in Figure 1.6) nodes in this sequence only swap the colors of two vertices of G, six intermediate steps are required to do so.

Satisfiability. A *propositional logic formula*, also called a *Boolean formula* or *formula* for short, is built from n *(Boolean) variables* combined into m *clauses*. Each clause consists of a subset of the variables combined using the *logical operators* AND (conjunction, also denoted by $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses. A formula is said to be *satisfiable* if it can be made TRUE by assigning appropriate logical values (i.e. TRUE/1 or FALSE/0) to its variables. The (Boolean) Satisfiability problem is, given a formula, to check whether it is satisfiable.

Satisfiability (SAT)
Input: A Boolean formula ϕ.
Question: Is ϕ satisfiable?

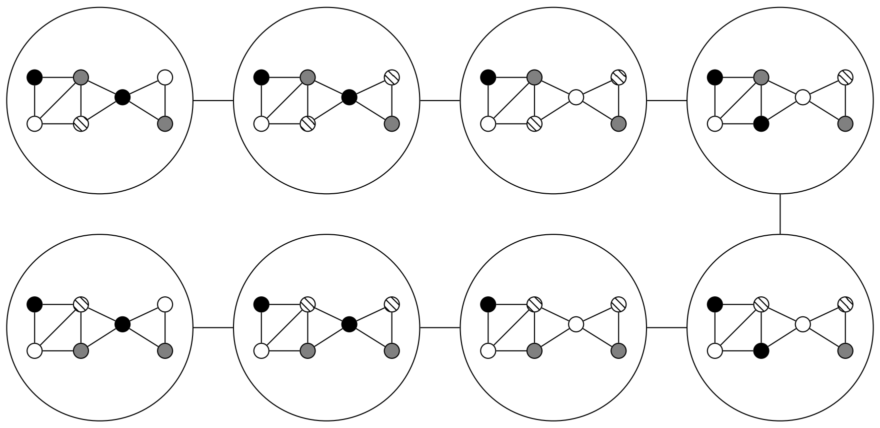
Figure 1.6: A reconfiguration sequence between two 4-colorings of a graph.

If we consider the graph whose node set consists of the collection of all possible Boolean vectors on n variables (strings of length n over the alphabet $\{0, 1\}$), where two nodes are adjacent whenever their corresponding vectors differ in exactly one variable (character), then we obtain what is known as (the graph of) the *n-dimensional hypercube*. Therefore, given a formula ϕ on n variables, the reconfiguration graph $R_{SAT}(\phi, 0, n)$ is the subgraph of the hypercube induced by the assignments which satisfy ϕ. In other words, if we delete from the hypercube all the nodes whose corresponding assignments do not satisfy ϕ along with the edges incident on them, then the resulting graph is $R_{SAT}(\phi, 0, n)$. A hypothetical example for a formula on seven variables is given in Figure 1.7. Note that for this "unweighted" version of the SAT problem, we do not impose any restrictions on the set of feasible solutions, i.e. we let Ψ be the *Hamming weight* (total number of variables set to 1 in a satisfying assignment) but we let $[r_l, r_u]$ be $[0, n]$, hence allowing all satisfying assignments of a formula in its reconfiguration graph.

1.2 Background and motivation

Triangulations and flip distance. Given a triangulation T of a convex polygon P, a *flip* of an edge e in T consists of replacing e by the other diagonal of the quadrilateral formed by the two triangles that share e, provided that this quadrilateral is convex. The *flip distance* between two triangulations T_s and T_t of P is the length of a shortest sequence of flips, or equivalently a shortest reconfiguration sequence, which transforms T_s to T_t. The Flip Distance problem asks, given T_s, T_t, and $k \in \mathbb{N}$, whether the flip distance between

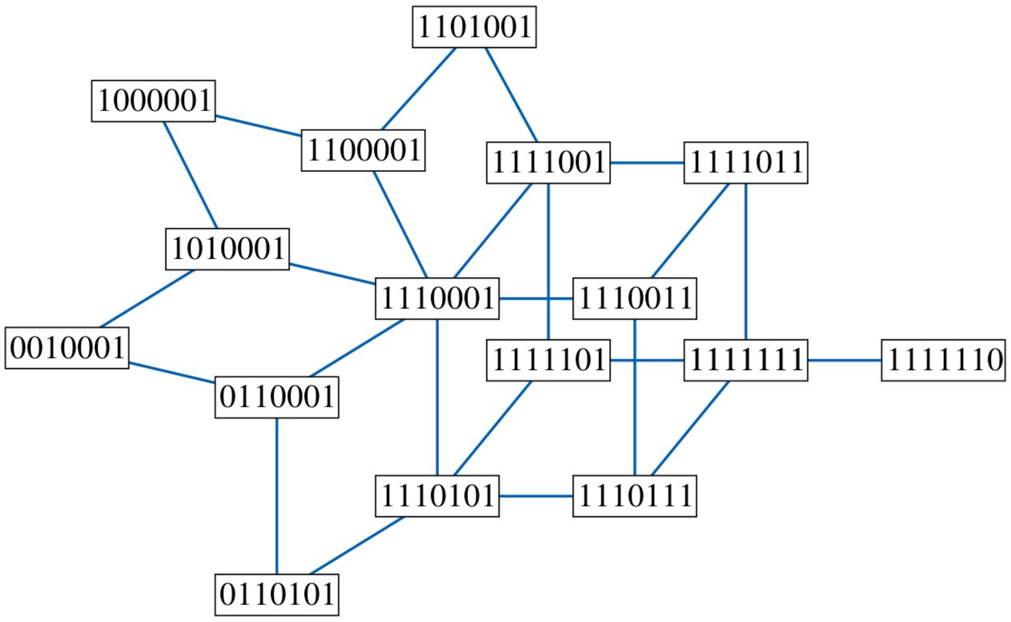

Figure 1.7: A hypothetical reconfiguration graph $R_{SAT}(\phi, 0, 7)$ (source: http://en.wikipedia.org/wiki/partial_cube © 2008 David Eppstein)

T_s and T_t is at most k. It is known that this distance is always well-defined [78], i.e. the reconfiguration graph induced by the triangulations of a convex polygon is connected. The FLIP DISTANCE problem is one of the oldest reconfiguration problems which does not originate from one-player games and is still actively studied today. The classical complexity of the problem has been open since 1987 [129]; it is not known whether finding the shortest reconfiguration sequence between two triangulations of a convex polygon is in P or is NP-complete. Sleator et al. [128] showed that for sufficiently large n, the flip distance between two triangulations of an n-gon is bounded above by $2n - 10$ and in some cases the bound is tight. Dyn et al. [47] proved that the reconfiguration graph induced by the triangulations of a simple polygon or a simple polygon with points inside it is connected, i.e. there exists a reconfiguration sequence between any two triangulations of such polygons. The diameter of this reconfiguration graph was shown to be in $\mathcal{O}(n^2)$ by Lawson [96]. Hurtado et al. [78] showed that this bound is asymptotically tight. Recently, the problem was shown to be fixed-parameter tractable parameterized by the flip distance [29].

The reason why this problem has gained particular interest is due to the close relationship between the flip distance and what is known as the *rotation distance* between rooted labeled binary trees. Binary search trees [95] are one of the most widely used data structures in computer science. Typically, search trees are more "effective" when they are balanced, and making a rooted binary tree balanced can be achieved using *rotations* [127]. As it turns out, there is a bijection between binary trees with $n - 1$ labeled leaves and triangulations of convex n-gons. In particular, a rotation in a tree is equivalent to a flip in some triangulation, making the problems of computing the flip distance and rotation distance equivalent. Computing the rotation distance is closely related to Sleator and Tarjan's famous dynamic optimality conjecture [127]. Several other variants and generalizations of the problem have also been studied in the literature. Back in 1936, Wagner [132] studied a variant where he considered triangulated planar graphs instead of convex polygons. He showed that given any two maximal planar graphs G_1 and G_2, i.e triangulated planar graphs, a reconfiguration sequence which transforms G_1 into a graph isomorphic to G_2 always exists. The best known upper bound of $5.2n - 24.4$ on the length of such a sequence is due to Bose et al. [22]. Very recently, Lubiw and Pathak showed that the more general problems of computing the flip distance between two triangulations of a polygon with holes or a set of points in the plane are NP-complete [100].

We note that the Flip Distance problem can be equivalently formulated as a reconfiguration problem on graphs. Specifically, given a convex polygon P, it is possible to construct a graph G such that there is a one-to-one correspondence between triangulations of P and vertex covers (or independent sets) of G. Therefore, studying reconfiguration variants of problems such as Vertex Cover (or Independent Set) might provide further insights into the complexity of the Flip Distance problem. We refer the reader to the extensive survey by Bose and Hurtado for more details [21].

Reconfiguration of satisfiability problems. Some key results known under the reconfiguration framework are due to Gopalan et al. [67], who studied reconfiguration of the Satisfiability problem (Schwerdtfeger [126] later published some corrections to the work of Gopalan et al. along with other results). Given a formula ϕ and two satisfying assignments, the authors studied the complexity of determining whether there exists a sequence of single variable flips, i.e. from false to true or true to false, that transforms the first assignment to the second such that each intermediate assignment still satisfies ϕ (i.e. the reachability problem). In 1978, Schaefer [125] introduced a framework for expressing variants of Satisfiability and proved a remarkable dichotomy theorem: Satisfiability is in P for certain classes of Boolean formulas, while it is NP-complete for all other classes in the framework. Schaefer's theorem states that Satisfiability is in P only for formulas built from "special" types of Boolean relations, which have since been called *Schaefer relations*. Gopalan et al. [67] extended this framework by defining *tight relations*, a superset of Schaefer relations. Using this definition, they proved the following dichotomies:

(i) Determining whether there exists a reconfiguration sequence between two satisfying assignments is in `P` for formulas built from tight relations and is `PSPACE`-complete for all other classes in the extended framework.

(ii) Determining whether the reconfiguration graph is connected is in `coNP` for formulas built from tight relations and is `PSPACE`-complete for all other classes in the extended framework.

Interestingly, the first dichotomy indicates that the set of Boolean formulas for which the reachability question is tractable is larger than the set of Boolean formulas for which the classical satisfiability question is tractable. In addition, Gopalan et al. established a structural dichotomy for the diameter of the connected components of the solution space of Boolean formulas. As expected, in the `PSPACE`-complete case the diameter can be exponential, but in all other cases it is linear. The work on reconfiguration of satisfying assignments was also motivated by structural questions regarding the solution space of random instances of SATISFIABILITY. It has been conjectured that the connectivity of the solution space has a major impact on the performance of standard satisfiability algorithms [67]. In particular, the success of Survey Propagation algorithms for solving random SATISFIABILITY instances is believed to be related to the structure of clusters of satisfying assignments (connected components in the reconfiguration graph) and frozen variables (variables whose value must remain fixed in any satisfying assignment). We refer the reader to the work of Achlioptas et al. [2] for more details.

Very recently, Gharibian and Sikora [66] studied the connectivity of ground spaces of local Hamiltonians, which captures problems in areas ranging from stabilizer codes to quantum memories. They showed that determining how "connected" the ground space of a local Hamiltonian is can range from `QCMA`-complete to `NEXP`-complete. As a result, they settled a ten years old open problem which asks for a "natural" `QCMA`-complete problem. In some sense, the work of Gharibian and Sikora (which we do not claim to fully understand) is the "quantum reconfiguration version" of satisfying assignments and they build on/extend the earlier work of Gopalan et al. [67] and Mouawad et al. [108].

Recoloring. Cereda et al. [25] formally initiated the investigation of the reconfiguration of graph colorings. As illustrated in Section 1.1, for any graph G and positive integer k, the reconfiguration graph of G, $R_{COL}(G, 0, k)$, has the proper k-colorings of G as its node set, and two k-colorings are joined by an edge if they differ in a color assignment of a single vertex. A graph G is said to be *k-mixing* if $R_{COL}(G, 0, k)$ is connected. The *mixing number* of G is the minimum value of k which guarantees that $R_{COL}(G, 0, k)$ is connected. The main result of the work of Cereda et al. was to show that when $k = \chi(G) \in \{2, 3\}$, G is not k-mixing, where $\chi(G)$ denotes the minimum number of colors needed for a proper coloring of G, i.e. the *chromatic number* of G. Moreover, for all $k \geq 4$, they show that some graphs with chromatic number k are k-mixing and some are not. The mixing number of a graph turns out to be closely related to research on rapid mixing of Markov chains [25].

The same authors later considered the complexity of finding a reconfiguration sequence when two proper k-colorings are given as part of the input, i.e. the reachability problem. This problem turns out to be solvable in polynomial time whenever $k \leq 3$ [27] and PSPACE-complete for all $k \geq 4$, even for bipartite graphs. The latter result was shown by Bonsma and Cereceda [18]. For any $k \geq 4$, examples have been explicitly constructed where any reconfiguration sequence between two colorings has exponential length [18]. On the other hand, for $k \leq 3$, the diameter of components of the reconfiguration graph is known to be polynomial [27, 131]. In recent work, Johnson et al. [87] showed that even finding the shortest reconfiguration sequence, i.e. the bounded reachability question, is solvable in polynomial time for $k \leq 3$. The case $k \leq 3$ is, in some sense, surprising since the underlying decision problem of determining whether a graph is 3-colorable is NP-complete. Note that a similar situation arises for the reconfiguration of satisfying assignments of a Boolean formula, as the set of Boolean formulas for which the reachability question is solvable in polynomial time is larger than the set of Boolean formulas for which the classical satisfiability question is solvable in polynomial time.

On the structural side, $R_{COL}(G, 0, k)$ was shown to be connected with diameter in $\mathcal{O}(n^2)$ whenever G is chordal (i.e. every cycle on four or more vertices has an edge that is not part of the cycle but connects two vertices of the cycle) and k is larger than the size of the largest clique in G [15] (and an infinite class of chordal graphs was described whose reconfiguration graphs have diameter n^2). This was further extended to show that if k is at least two greater than the treewidth (defined in Section 2.1.3) of G then, again, $R_{COL}(G, 0, k)$ is connected with diameter in $\mathcal{O}(n^2)$ [13]. Feghali et al. [52] showed that if k is one larger than the maximum degree of G then $R_{COL}(G, 0, k)$ consists of one "large" connected component whose diameter is in $\mathcal{O}(n^2)$ and possibly many isolated vertices.

A natural generalization of coloring problems is the notion of homomorphisms. A *homomorphism* from a graph G to a graph H is a mapping from the vertices of G to the vertices of H which maps each edge of G to an edge of H. When H is a clique on k vertices this is exactly the problem of finding a proper k-coloring of G. Wrochna [133] initiated the study of reconfiguration of homomorphisms and showed that when H satisfies certain properties both the reachability and bounded reachability questions are in P, generalizing the results for the coloring case where $k \leq 3$. Finally, we also point out that alternative adjacency relations were also considered in the literature. For instance, Mohar [107] studied the case where two k-colorings are adjacent if one can be obtained from the other by swapping all colors in a *Kempe chain*; given a proper k-coloring of G, with $k \geq 2$, a Kempe chain of G is a maximal connected subgraph of G whose vertices all all assigned one of two colors. We refer the reader to the recent survey by van den Heuvel for more details [131].

Moving on graphs. Ito et al. [81] were the first to introduce reconfiguration as a "framework", by considering a host of optimization problems and their corresponding reachability/bounded reachability variants. They showed that the reachability question for the NP-complete problems of Power Supply, Clique, Independent Set, Vertex Cover, Set Cover, Dominating Set, and Integer Programming are all PSPACE-complete. On the positive side, the question is shown to be in P for the Minimum Spanning Tree and Matching problems, which are both in P. It should be noted that not all problems make use of the same adjacency relation. For example, in the case of Matching, the authors use addition or removal of an edge as a single reconfiguration step while for Clique they consider vertices.

Almost every PSPACE-hardness proof under the reconfiguration framework makes direct or indirect use of the Nondeterministic Constraint Logic (NCL) model of computation introduced by Hearn and Demaine [77]. The model is based on a weighted undirected graph, the *NCL machine*, with assignments of positive integer weights to the edges and vertices of the graph. A feasible configuration of an NCL machine is an orientation of its edges (in one of two directions) such that the sum of incoming edge weights at each vertex is at least the weight of that vertex. A reconfiguration step consists of reversing the direction of a single edge, given that the resulting configuration remains feasible. It is shown, by a reduction from Quantified Boolean Satisfiability [77], that determining whether the direction of a single edge can be reversed while maintaining feasibility is PSPACE-complete. The authors first used their model to establish the complexity of several one-player games such as the sliding blocks puzzle. An alternative formulation of NCL, equivalent to sliding tokens on graphs, is also presented. Given a graph G with tokens placed on a subset of its vertices such that no two adjacent vertices receive a token, a reconfiguration step consists of sliding a token from a vertex to one of its neighbors as long as the resulting configuration remains feasible, i.e. no tokens become adjacent. The corresponding decision problem, also PSPACE-complete, is to determine whether a token can slide from one vertex to another.

Related work and applications. While the study of reconfiguration problems is primarily motivated by the need for a better understanding of the solution space of combinatorial problems, some reconfiguration problems have more practical applications [131]. One such example is the security problem discussed in Section 1.1. Another popular example is the Frequency Assignment Problem (FAP). The goal is to assign frequencies to users of a wireless network such that the interference between them is minimized and the range of used frequencies is as small as possible. Since the radio spectrum is bounded and the number of services that rely on it is constantly growing (e.g. mobile systems), it has become crucial to come up with "efficient" strategies to use it. The FAP problem can be modeled as a graph coloring problem, where frequencies are discretised appropriately and viewed as colors [111]. However, this static modeling of the problem does not capture most of the "real-world" situations, such as decay of radio waves due to distance and other external factors. Typically, in mobile systems, new transmitters are often added to meet increasing demand. Hence, optimal or near-optimal assignment of frequencies will in general not remain so for long. In addition, because finding optimal assignments is "hard", a sub-optimal assignment might need to be replaced with a recently-found better one. It thus becomes necessary to think of the assignment of frequencies as a dynamic process, where one assignment is to be replaced with another. In order to avoid interruptions of service, it becomes necessary to avoid a complete re-setting of the frequencies used on the whole network [9, 20, 75]; this gives rise to a reconfiguration problem.

Moving tokens on graphs naturally extends to motion planning of objects on a plane. For instance, in the 3D-printing industry, fused deposition modeling (FDM) is an additive manufacturing technology [89] which works by laying down material in layers; a "head" follows a predetermined path laying down material layer by layer. A major drawback of this technology is the low speed of producing large parts. One possible solution to the problem is to use multi-head 3D-printers. Planning the movement of multiple heads to avoid collisions and other unwanted interactions can also be cast as a reconfiguration problem on graphs. Moreover, to minimize printing time, minimizing the distance traveled by each head becomes important.

Finally, we also mention an area which is older but very closely related to reconfiguration: local search [3, 55, 65, 72]. Local search is a fundamental strategy used in many areas such as combinatorial optimization and artificial intelligence. Given some initial solution to a problem, the goal is to find a "better" solution which is not "too far away" from the given solution. Typically, the distance between solutions is defined in terms of the *k-exchange neighborhood*; a solution S (viewed as a set) is in the k-exchange neighborhood of a solution S' whenever the size of the symmetric difference of S and S' is at most k. As a rule of thumb, the larger the value of k, the better the chances of finding a better solution but the longer it takes to find it. For example, in the TRAVELING SALESMAN PROBLEM (TSP) problem, the goal is to find a routing (a set of edges in a weighted graph) for a salesperson who starts from a home location (a vertex), visits all cities (vertices), and returns to the original location such that the total distance traveled (edge weights) is minimized and each city is visited exactly once. Local search strategies for TSP have been extensively studied in the literature [3, 72]. In this case, a solution S is said to be in the k-exchange neighborhood of S' whenever S is obtained from S' by swapping at most k edges.

1.3 Overview

As more and more reconfiguration problems have been investigated, some general patterns have started emerging. A list of some of these results is given in Table 1.1 along with the classical complexity of the decision version of the underlying problem and known upper bounds on the diameter of the corresponding reconfiguration graphs. At first glance, Table 1.1 suggests that the reachability and bounded reachability variants of NP-complete problems are PSPACE-complete, while for problems in P both problems are also in P. However this is clearly not the case. We know that the relationship between the complexity of the underlying problem and its reachability/bounded reachability variants is more subtle. Both the reachability and bounded reachability variants of the 3-COLORING problem, which is NP-complete, are in P. On the other hand, the 4-COLORING problem is trivially in P for bipartite or planar graphs but the reachability variant is PSPACE-complete [18].

Recently, Bonsma proved that the reachability variant of SHORTEST PATH, a problem in P, is in fact PSPACE-complete [16]. The reachability variant of the SHORTEST PATH problem was first introduced by Kamiński et al. [90]. We are given a graph G, two vertices s and t, and two shortest paths between s and t in G and we are asked whether one path can be transformed into the other such that every intermediate path is also a shortest path and differs from the previous one in exactly one vertex. An important motivation for studying the reachability of shortest paths was an attempt to break a second pattern that has been observed for reachability problems in general. This pattern relates the complexity of reachability problems and the diameter of the reconfiguration graph (or of its connected components); for most reachability problems in P [14, 27, 82, 91], the diameter is polynomially bounded, and for all PSPACE-complete reachability problems, the diameter is exponential [18]. The latter is not surprising since otherwise NP = PSPACE. Even though the reachability of shortest paths gives another example of a problem in P with a reachability version which is PSPACE-complete, it is still unknown whether the general correlation between the complexity of reachability problems and the diameter of the re-

Table 1.1: Classical complexity classification of some reconfiguration problems.

Problem	Decision	Reach.	Bound.	Diam.	Ref.
Flip Distance (convex polygons)	P	P	open	poly	[21]
Flip Distance (point sets)	P	P	NPC	poly	[100]
4-Coloring	NPC	PSPACEC	PSPACEC	exp	[18]
3-Coloring	NPC	P	P	poly	[25, 27, 87]
2-Coloring	P	P	P	poly	[25, 27, 87]
3-SAT	NPC	PSPACEC	PSPACEC	exp	[67]
2-SAT	P	P	P	poly	[67]
Horn-Sat	P	P	open	poly	[67]
Vertex Cover	NPC	PSPACEC	PSPACEC	exp	[81]
Independent Set	NPC	PSPACEC	PSPACEC	exp	[81, 91]
Clique	NPC	PSPACEC	PSPACEC	exp	[81]
Dominating Set	NPC	PSPACEC	PSPACEC	exp	[81]
List Edge-Coloring	NPC	PSPACEC	PSPACEC	exp	[82]
List Edge-Coloring (in trees)	P	P	open	poly	[82]
List $L(2,1)$-Labeling	NPC	PSPACEC	PSPACEC	exp	[85]
Shortest Path	P	PSPACEC	PSPACEC	exp	[16, 90]
Matching	P	P	open	poly	[81]
Minimum Spanning Tree	P	P	open	poly	[81]

configuration graph holds for all "natural" problems; constructing artificial instances of reconfiguration problems to break this pattern can be easily accomplished, e.g. 4-colorings of a bipartite graph [18]. Note that for bounded reachability problems, i.e. finding reconfiguration sequences of bounded length, this correlation clearly does not hold; the Flip Distance problem for planar point-sets (Table 1.1) is one example where the reconfiguration graph is connected and of polynomial diameter but the bounded reachability question is NP-complete.

An important observation is that the results concerning the bounded reachability question have been in most cases a direct consequence of the study of reachability, as PSPACE-completeness of the latter implies PSPACE-completeness of the former. For the tractable cases, the relationship between reachability and bounded reachability is slightly more subtle, hence we illustrate it using the example of 2-SAT. It was shown by Gopalan et al. [67] that any reconfiguration sequence from an assignment α to another assignment β can always flip the variables which are assigned different values in α and β and never flip any of the remaining variables. Any reconfiguration sequence following this behavior will always be of shortest possible length as these flips constitute the minimum set of required flips to transform α to β. Formally, this behavior, which we call *symmetric-difference-only* behavior, occurs whenever the shortest path between any two satisfying assignments is equal to the size of their symmetric difference. The symmetric-difference-only behavior was also observed when considering some of the graph problems listed in Table 1.1 on special types of instances [91]. As we shall see later, this behavior is strongly related to the complexity of bounded reachability questions and it has been detected for many of the known problems for which finding a shortest reconfiguration sequence is solvable in polynomial time.

Reconfiguration problems have so far been studied only under classical complexity assumptions. In fact, most work has been limited to studying the complexity of reachability questions. Let us consider a typical example. Both the reachability and bounded reachability variants of the Independent Set problem are known to be PSPACE-complete. Hence, one cannot hope for any efficient algorithm for solving the problem. But should we lose all hope of solving the problem? What if, for instance, we are given a graph G on $n = 10^6$ vertices and two independent sets each of size $k = 10$ and are asked to determine whether there exists a reconfiguration sequence between the two? Is an algorithm running in $2^k n$ time possible? Note that such an algorithm would be "practical" and can actually be implemented and used to solve the problem on almost any computer available today. Another framework that has become an essential tool for researchers in computational complexity during the last two decades or so and which deals with exactly these types of questions is *parameterized complexity*. In classical complexity theory, all NP-complete (or PSPACE-complete) problems are indistinguishably hard and can have no algorithms with efficient running times unless P = NP (or P = PSPACE). Unfortunately, a large number of computational problems we need to solve in practice are hard [63], and reconfiguration problems are no exception. Several ideas have emerged to cope with this inherent limitation and one of the newest ones, introduced by Downey and Fellows [46], is parameterized complexity. At the heart of parameterized complexity lies the realization that the description of a large number of hard problems is oblivious to structural properties of both the input instance and the output solution. For example, graph instances could have bounded degree, bounded diameter, or bounded treewidth (Section 2.1). Similar restrictions could apply to the desired output, where constraints such as the maximum acceptable size of a solution are very common in practice. For properties that can be captured numerically, we can augment problem definitions to include parameters. This leads to the notion of a *parameterized problem*. After more than two decades of research [43, 44, 45, 46, 42, 61], it is now clear that, depending on the choice of parameter, problems can be very different from each other in terms of "hardness". It has been shown that a problem that is otherwise intractable can have an efficient algorithm as long as the parameter is kept small [46]. For other problems, the introduction of a parameter does little to improve the situation, and the problem remains intractable. In summary, parameterized complexity tries to confine the exponential explosion in the running time of a problem to the parameter instead of the input size.

Thus, we believe it is natural to ask about the variations in the complexity landscape of reachability and bounded reachability problems under the parameterized setting. The purpose of our work is to embark on a systematic investigation of the tractability and structural properties of such problems under both classical and parameterized complexity assumptions. In particular, we are interested in what separates the tractable instances from the intractable ones. It is clear, from the generic definition of reconfiguration problems, that several factors affect their complexity status. Our work aims at providing a "finer" classification of the complexity of reachability and bounded reachability problems with respect to some of these factors, including the definition of the adjacency relation $\mathcal{A}$, the choice of property Ψ, the values of r_l and r_u, structural properties of the input instance $\mathcal{I}$, structural properties of the reconfiguration graph, and the length of a reconfiguration sequence. As most of these factors can be numerically quantified, we believe that the investigation of reconfiguration problems under both parameterized and classical complexity assumptions will help us further understand the boundaries between tractability and intractability. Moreover, such an investigation would provide a finer classification of problems into one of three categories; the tractable, the fixed-parameter tractable, and the intractable cases.

The remainder of this document is organized as follows: Chapter 2 outlines the notation and basic definitions and provides some preliminary results. In Chapter 3, we consider the reachability and bounded reachability variants of the Dominating Set problem, or equivalently Unbounded Hitting Set problem, and prove a series of results which help build up intuition on why such problems can be hard and what types of restrictions or structural properties can make some instances easy. In Chapter 4, and in contrast to Chapter 3, we study the reachability and bounded reachability variants of the Bounded Hitting Set problem in general, and we focus on the Vertex Cover problem in particular. As we shall see, going from "unbounded" to "bounded" greatly affects the complexity of reconfiguration problems, especially in the parameterized setting. We formalize and generalize some of the techniques from earlier chapters in Chapter 5. In particular, we show how to adapt known techniques from the parameterized complexity "toolkit" such as bounded search trees, parameterized enumeration, kernelization, irrelevant vertices, and compact representations, to reachability and bounded reachability problems. In doing so, we answer some of the questions left open in recent literature. In Chapter 6, we consider reconfiguration questions related to coloring and, more generally, to constraint satisfaction problems. Concluding remarks, open problems, and directions for future research are discussed in Chapter 7. The appendix includes a complete list of problems considered throughout this work. Most of the results presented in this thesis have appeared in part or in full in one of the following conference papers and/or manuscripts:

- *On the parameterized complexity of reconfiguration problems.* In Proceedings of the 8th International Symposium on Parameterized and Exact Computation, IPEC 2013. Joint work with Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki.

- *Vertex cover reconfiguration and beyond.* In Proceedings of the 25th International Symposium on Algorithms and Computation, ISAAC 2014. Joint work with Naomi Nishimura and Venkatesh Raman.

- *Reconfiguration of dominating sets.* In Proceedings of the 20th International Computing and Combinatorics Conference, COCOON 2014. Joint work with Naomi Nishimura and Akira Suzuki.

- *Reconfiguration over tree decompositions.* In Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC 2014. Joint work with Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna.

- *The complexity of bounded length graph recoloring and CSP reconfiguration.* In Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC 2014. Joint work with Paul Bonsma, Naomi Nishimura, and Venkatesh Raman.

- *Shortest reconfiguration paths in the solution space of Boolean formulas* (manuscript). Submitted to the Computational Complexity Conference, CCC 2015. Joint work with Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman.

- *The complexity of dominating set reconfiguration* (manuscript). Join work with Arash Hadadan, Takehiro Ito, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal.

- *Reconfiguration on sparse graphs* (manuscript). Joint work with Daniel Lokshtanov, Fahad Panolan, M.S. Ramanujan, and Saket Saurabh.

Chapter 2

Preliminaries

2.1 Graph theory

For an in-depth review of general graph theoretic definitions we refer the reader to the book of Diestel [41]. Unless otherwise stated, we assume that each graph G is a simple, undirected graph with vertex set $V(G)$ and edge set $E(G)$, where $|V(G)| = n$ and $|E(G)| = m$. The *open neighborhood*, or simply *neighborhood*, of a vertex v is denoted by $N_G(v) = \{u \mid uv \in E(G)\}$, the *closed neighborhood* by $N_G[v] = N_G(v) \cup \{v\}$. Similarly, for a set of vertices $S \subseteq V(G)$, we define $N_G(S) = \{v \mid uv \in E(G), u \in S, v \notin S\}$ and $N_G[S] = N_G(S) \cup S$. The *degree* of a vertex is $|N_G(v)|$. We drop the subscript G when clear from context. A *subgraph* of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *induced subgraph* of G with respect to $S \subseteq V(G)$ is denoted by $G[S]$; $G[S]$ has vertex set S and edge set $\{uv \in E(G[S]) \mid u, v \in S, uv \in E(G)\}$. We denote by $\Delta(G)$ and $\delta(G)$ the maximum and minimum degree of G, respectively.

A *walk* of length ℓ from v_0 to v_ℓ in G is a vertex sequence $v_0, \ldots, v_\ell$, such that for all $i \in \{0, \ldots, \ell - 1\}$, $v_i v_{i+1} \in E(G)$. It is a *path* if all vertices are distinct. It is a *cycle* if $\ell \geq 3$, $v_0 = v_\ell$, and $v_0, \ldots, v_{\ell-1}$ is a path. A path from vertex u to vertex v is also called a *uv-path*. The *distance* between two vertices u and v of G, $dist_G(u, v)$, is the length of a shortest uv-path in G (positive infinity if no such path exists). The *eccentricity* of a vertex $v \in V(G)$, $ecc(v)$, is equal to $max_{u \in V(G)}(dist_G(u, v))$. The *radius* of G, $rad(G)$, is equal to $min_{v \in V(G)}(ecc(v))$. The *diameter* of G, $diam(G)$, is equal to $max_{v \in V(G)}(ecc(v))$. For $r \geq 0$, the *r-neighborhood* of a vertex $v \in V(G)$ is defined as $N^r_G[v] = \{u \mid dist_G(u, v) \leq r\}$. We write $B(v, r) = N^r_G[v]$ and call it a *ball of radius r around v*; for $S \subseteq V(G)$, $B(S, r) = \bigcup_{v \in S} N^r_G[v]$.

A vertex or edge set satisfying a specific condition is *minimal (maximal)* (with respect to the condition) if no proper subset (superset) of the set satisfies the condition. A graph G is *connected* if there is a path between every pair of vertices. A *connected component* of G is an induced subgraph C of G such that $V(C)$ is a maximal subset of $V(G)$ such that $G[V(C)]$ is connected. A set $S \subset V(G)$ is a *separator* if $G[V(G) \setminus S]$ is disconnected. Vertices s and t (vertex sets S and T) are *separated* if $s \neq t$ ($S \cap T = \emptyset$) and there is no edge $st \in E(G)$ (no edge $st \in E(G)$ for $s \in S$ and $t \in T$). Two subgraphs of G are *separated* whenever their corresponding vertex sets are separated.

A *matching* $M \subseteq E(G)$ in a graph G is a set of edges of G such that no two of them share a vertex. The largest possible matching on a graph with n vertices consists of $\lfloor \frac{n}{2} \rfloor$ edges, and such a matching is called a *perfect matching.* We write $V(M)$ to denote the set of vertices incident to edges in M. A matching M *saturates* $S \subseteq V(G)$ if $S \subseteq V(M)$.

We adopt known conventions for referring to some special graphs [23]. P_k denotes a path on k vertices and $k-1$ edges. C_k denotes a cycle on k vertices and k edges. *Trees* are connected graphs without cycles. A disconnected graph without cycles is a *forest.* K_k denotes a *clique* of size k, i.e. k pairwise adjacent vertices. $\overline{K_k}$ denotes an *independent set* of size k, i.e. k pairwise nonadjacent vertices. A graph G is *bipartite* if there exists a partition (A, B) of its vertex set such that $G[A]$ and $G[B]$ are edgeless. K_{k_1,k_2} denotes a *biclique*, i.e. a bipartite graph with k_1 vertices in the first partition, k_2 vertices in the second, and all edges between them. $K_{1,k}$ denotes a star with k leaves. We say $K_{1,k}$ is a *k-star* or a star of size k. A graph G is *d-regular* if $\Delta(G) = \delta(G) = d$. A *cactus graph* is a graph in which every edge belongs to at most one cycle. A graph is *even-hole-free* if it contains no induced cycle with an even number of vertices. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. We sometimes use the notation $G + H$ to denote the graph obtained by taking the disjoint union of two graphs G and H.

2.1.1 Graph properties

A *graph property* Π is a collection of graphs, and is *non-trivial* if it is non-empty and does not contain all graphs. A graph property is *polynomially decidable* if for any graph G, it can be decided in polynomial (in n) time whether G is in Π. The property Π is *hereditary* if for any $G \in \Pi$, any induced subgraph of G is also in Π. Examples of hereditary properties include graphs having no edges and graphs having no cycles. It is well-known [97] that every hereditary property Π has a *forbidden set* $\mathcal{F}_\Pi$, in that a graph has property Π if and only if it does not contain any graph in $\mathcal{F}_\Pi$ as an induced subgraph.

2.1.2 Graph problems

Most problems we consider in Chapters 3, 4, and 5 fall under the "umbrella" of graph vertex-subset problems. We say a graph problem $\mathcal{Q}$ is a *vertex-subset* problem whenever feasible solutions for $\mathcal{Q}$ on input G correspond to subsets of $V(G)$. $\mathcal{Q}$ is a *vertex-subset minimization (maximization)* problem whenever feasible solutions for $\mathcal{Q}$ correspond to subsets of $V(G)$ of size at most (at least) k, for some integer k.

A classical example of a vertex-subset minimization problem, for which we study the reachability and bounded reachability variants in Chapter 3, is Dominating Set. A set $D \subseteq V(G)$ is called a *dominating set* of G if $N[D] = V(G)$. The corresponding decision problem is defined as follows.

Dominating Set (DS)	
Input:	A graph G and a positive integer k
Question:	Is there $D \subseteq V(G)$ such that $|D| \leq k$ and $N[D] = V(G)$?

The property of a graph having a dominating set of size at most k is not hereditary. However, many classical vertex-subset problems can be expressed using hereditary graph properties (Section 2.1.1). We exploit the close relationship between such problems to prove both negative and positive results in Chapters 4 and 5.

Definition 2.1.1. *For any polynomially decidable hereditary graph property* Π*, graph* G*, and positive integer* k*, we define the following decision problems:*

Π-Del(G, k)*: Is there* $S \subseteq V(G)$ *such that* $|S| \leq k$ *and* $G[V(G) \setminus S] \in \Pi$*?*
Π-Sub(G, k)*: Is there* $S \subseteq V(G)$ *such that* $|S| \geq k$ *and* $G[S] \in \Pi$*?*
Π-Comp(G, S, k)*: For* $|S| = k$ *and* $G[V(G) \setminus S] \in \Pi$*, is there* $S' \subseteq V(G)$ *such that* $|S'| < |S|$ *and* $G[V(G) \setminus S'] \in \Pi$*?*

We say that Π-Del and Π-Sub are *parametric duals* of each other. In the Π-Sub problem, we seek a set of vertices of size *at least* k inducing a subgraph in Π, whereas in Π-Del, we seek a set of vertices of size *at most* k whose *complement set* induces a subgraph in Π. Examples include Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal for the latter and Independent Set, Induced Forest, and Induced Bipartite Subgraph for the former, for Π defined as the collection of all edgeless graphs, forests, and bipartite graphs, respectively. We introduce Vertex Cover and Independent Set below. The appendix contains a complete list of problem definitions.

Vertex Cover (VC)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \leq k$ and $G[V(G) \setminus S]$ is edgeless?

Independent Set (IS)
Input: A graph G and a positive integer k
Question: Is there $I \subseteq V(G)$ such that $|I| \geq k$ and $G[I]$ is edgeless?

2.1.3 Bandwidth, pathwidth, and treewidth

We give formal definitions of what constitutes "easy" and "hard" problems in Sections 2.2 and 2.3. For now, we simply note that (under classical complexity assumptions) all problems mentioned in Section 2.1.2 are hard on general graphs [63]. However, if the input graph has some "manageable structure", all of these problems become easy [12, 94]. Three related graph measures which we define below and imply such manageable structure are bandwidth, pathwidth, and treewidth.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \chi)$, where T is a tree and χ is a mapping that assigns to each node $i \in V(T)$ a vertex subset X_i (called a *bag*) such that:

(1) $\bigcup_{i \in V(T)} X_i = V(G)$,

(2) for every edge $uv \in E(G)$, there exists a node $i \in V(T)$ such that the bag $\chi(i) = X_i$ contains both u and v, and

(3) for every $v \in V(G)$, the set $\{i \in V(T) \mid v \in X_i\}$ forms a connected subgraph (subtree) of T.

The *width* of any tree decomposition $\mathcal{T}$ is equal to $\max_{i \in V(T)} |X_i| - 1$. The *treewidth* of a graph G, $tw(G)$, is the minimum width of a tree decomposition of G. For any graph of treewidth t, we can compute a tree decomposition of width t and transform it into a nice tree decomposition of the same width in time linear in n [94], where a rooted tree decomposition $\mathcal{T} = (T, \chi)$ with root *root* of a graph G is a *nice tree decomposition* if each of its nodes is either

(1) a leaf node (a node i with $|X_i| = 1$ and no children),

(2) an introduce node (a node i with exactly one child j such that $X_i = X_j \cup \{v\}$ for some vertex $v \notin X_j$; v is said to be *introduced* in i),

(3) a forget node (a node i with exactly one child j such that $X_i = X_j \setminus \{v\}$ for some vertex $v \in X_j$; v is said to be *forgotten* in i), or

(4) a join node (a node i with two children p and q such that $X_i = X_p = X_q$).

For node $i \in V(T)$, we use T_i to denote the subtree of T rooted at i and V_i to denote the set of vertices of G contained in the bags of T_i. Thus $G[V_{root}] = G$.

A *path decomposition* of a graph G is a tree decomposition of G where the underlying tree T is a path. The *pathwidth* of G, $pw(G)$, is the minimum width over all possible path decompositions of G. Intuitively, the pathwidth of a graph is a measure of how "path-like" the graph is. Similarly, the treewidth of a graph is a measure of how "tree-like" the graph is. The pathwidth of any graph is greater than or equal to its tree-width.

The *bandwidth* of a graph is the minimum over all assignments $f : V(G) \rightarrow \mathbb{N}$ of the quantity $max_{uv \in E(G)}|f(u) - f(v)|$. A *bucket arrangement* of a graph G is a partition of $V(G)$ into a sequence of buckets, such that the endpoints of any edge in $E(G)$ are either in one bucket or in two consecutive buckets. If a graph has a bucket arrangement where each bucket has at most b vertices, then it has bandwidth at most $2b$ [53]. Hence, a graph of bandwidth b can easily be seen to have pathwidth and treewidth at most b and maximum degree at most $2b$.

2.1.4 Sparse graphs

Graphs of bounded bandwidth, pathwidth, or treewidth share one common feature: they are not dense. A *dense graph* is a graph in which the number of edges is close to the maximal number of edges, i.e. $\binom{n}{2}$. A graph with only a few edges is a *sparse graph*. The distinction between sparse and dense graphs is rather vague and varies depending on the context. However, the realization that many graph problems become tractable on sparse graphs has lead to a long line of research, where the main goal has been to find the largest graph class on which algorithmic techniques such as dynamic programming, divide and conquer, and/or graph separators can be adapted to solve problems efficiently [68, 70]. Central to this research is the work of Robertson and Seymour, which ultimately led to one of the most fundamental results in graph theory, the Graph Minor Theorem [122]. More recently, and in an attempt to settle the dichotomy of what separates sparse graphs from dense graphs, Nesetril and Ossona De Mendez [114, 112] introduced the class of nowhere-dense graphs. Indeed, most familiar examples of sparse graph classes turn out to be nowhere-dense. We need a few additional definitions before discussing the details.

Contracting an edge uv of a graph G results in a new graph H in which the vertices u and v are deleted and replaced by a new vertex w that is adjacent to $N_G(u) \cup N_G(v) \setminus \{u, v\}$. If a graph H can be obtained from G by repeatedly contracting edges, H is said to be a *contraction* of G. If H is a subgraph of a contraction of G, then H is said to be a *minor* of G, denoted by $H \preceq_m G$.

Definition 2.1.2. *A class of graphs $\mathcal{C}$ is said to be H-*minor-free*, if H is not a minor of any $G \in \mathcal{C}$.*

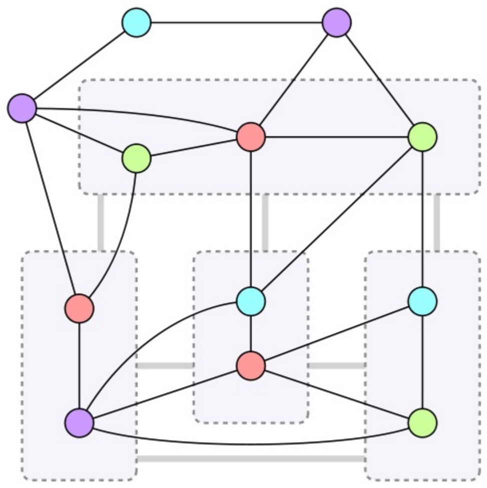

Figure 2.1: A graph G that has the complete graph K_4 as a 1-shallow minor (source: http://en.wikipedia.org/wiki/shallow_minor © 2008 David Eppstein)

Subdividing an edge uv of G results in a new graph H in which the edge uv is deleted and replaced by two edges uw and wv, where w is a new vertex. If a graph H can be obtained from G by repeatedly subdividing edges, H is said to be a *subdivision* or an *expansion* of G. A contraction of edge uv is *topological* if either u or v have degree at most two in G. Such a contraction can be viewed as an inverse operation to the subdivision operation. H is a *topological minor* of G, denoted by $H \preceq_{tm} G$, if G contains a subdivision of H as a subgraph.

Definition 2.1.3. *A class of graphs $\mathcal{C}$ is said to be H-*topological-minor-free*, if H is not a topological minor of any $G \in \mathcal{C}$.*

An equivalent characterization of minors states that H is a minor of G if there is a mapping that associates to each vertex v of H a non-empty connected subgraph G_v of G such that G_u and G_v are disjoint, for $u \neq v$, and whenever there is an edge between u and v in H there is an edge in G between a vertex in G_u and a vertex in G_v. The subgraphs G_v are called *branch sets* and vertices of H are called *supervertices*. H is a *minor at depth* r of G or an r*-shallow minor* of G [120], denoted by $H \preceq_m^r G$, if H is a minor of G which is witnessed by a collection of branch sets $\{G_v \mid v \in V(H)\}$, each of which induces a graph of radius at most r. That is, for each $v \in V(H)$, there is a $w \in V(G_v)$ such that $V(G_v) \subseteq N^r_{G_v}[w]$. Shallow minors with depth n coincide with minors, while the shallow minors with depth zero are exactly the subgraphs of a graph. Figure 2.1 shows a graph G that has the complete graph K_4 as a 1-shallow minor. Each of the four vertex subsets indicated by the dashed rectangles induces a connected subgraph with radius one, and there exists an edge between every pair of subsets.

Definition 2.1.4. *A class of graphs $\mathcal{C}$ is said to be* nowhere-dense *if for every $r \geq 0$ there exists a graph H_r such that $H_r \not\preceq_m^r G$ for all $G \in \mathcal{C}$. $\mathcal{C}$ is* effectively nowhere-dense *if the map $r \mapsto H_r$ is computable. Otherwise, $\mathcal{C}$ is said to be* somewhere-dense.

"Nowhere-density" turns out to be a very robust concept with several natural characterizations [69]. We use one such characterization in Section 5.4.2. Although planar graphs, graphs of bounded treewidth, graphs of bounded degree, H-minor-free graphs, and H-topological-minor-free graphs are nowhere-dense [114, 112], some "natural" sparse graph classes fail to satisfy Definition 2.1.4. The most notable example is the class of graphs of bounded degeneracy.

Definition 2.1.5. *A class of graphs $\mathcal{C}$ is said to be* d-degenerate *if there is an integer d such that every induced subgraph of any graph $G \in \mathcal{C}$ has a vertex of degree at most d.*

Proposition 2.1.6 ([98])**.** *The number of edges in a d-degenerate graph is at most dn and hence its average degree is at most $2d$.*

Graphs of bounded degeneracy and nowhere-dense graphs are in fact incomparable [70]. Degeneracy is a hereditary property, hence any induced subgraph of a d-degenerate graph is also d-degenerate. It is well-known that graphs of treewidth at most d are also d-degenerate. Moreover a d-degenerate graph cannot contain $K_{d+1,d+1}$ as a subgraph, which brings us to the class of biclique-free graphs. The relationship between bounded degeneracy, nowhere-dense, and $K_{d,d}$-free graphs was shown by Philip et al. and Telle and Villanger [119, 130].

Definition 2.1.7. *A class of graphs $\mathcal{C}$ is said to be* d-biclique-free*, for some $d > 0$, if $K_{d,d}$ is not a subgraph of any $G \in \mathcal{C}$, and it is said to be* biclique-free *if it is d-biclique-free for some d.*

Proposition 2.1.8 ([119, 130])**.** *Any degenerate or nowhere-dense class of graphs is biclique-free, but not vice-versa.*

Figure 2.2 illustrates some of the inclusion relationships among various well-known graph classes, including all the classes defined above [23, 114, 112].

2.2 Classical complexity

We will assume a random-access machine as the underlying machine model throughout this work [63, 117]. In the random-access machine any simple operation (arithmetic, if-statements, memory access, and so on) takes unit time. We define only the classical complexity classes we will generally encounter. The interested reader can find full details in the excellent textbooks of, e.g., Garey and Johnson [63] and Papadimitriou [117].

Definition 2.2.1. *The* time complexity *of an algorithm* ALG *is the function $t : \mathbb{N} \to \mathbb{N}$, where $t(n)$ is the maximum number of steps that* ALG *uses on any input of length n. We also say that* ALG *is a $t(n)$*-time algorithm.

Definition 2.2.2. *The* space complexity *of an algorithm* ALG *is the function $s : \mathbb{N} \to \mathbb{N}$, where $s(n)$ is the maximum amount of memory space that* ALG *uses on any input of length n. We also say that* ALG *is an $s(n)$*-space algorithm.

A system of time and space complexity classes has been devised to classify problems according to their time and space complexity. Each of these classes contains problems that are asymptotically equivalent.

Definition 2.2.3. *A* problem *or a* language *is a set L of strings of length at most n over a finite alphabet Σ, that is $L \subseteq \Sigma^n$. A string $s \in L$ is a* yes-instance *of L and a string $s \notin L$ is a* no-instance *of L. The* complement *of a problem L is the set $\Sigma^n \setminus L$.*

Definition 2.2.4. *For a function $t : \mathbb{N} \to \mathbb{N}$, the time complexity class $TIME(t(n))$ is defined as $TIME(t(n)) = \{L \mid L$ is a language decided by an $\mathcal{O}(t(n))$-time algorithm$\}$.*

Definition 2.2.5. *For a function $s : \mathbb{N} \to \mathbb{N}$, the space complexity class $SPACE(s(n))$ is defined as $SPACE(s(n)) = \{L \mid L$ is a language decided by an $\mathcal{O}(s(n))$-space algorithm$\}$.*

The class P is the class of problems that are solvable in polynomial time. Historically, P has been considered as roughly equivalent to the class of "easy" problems, i.e. problems that are computationally easy to solve, and any problem not in P is considered hard to solve.

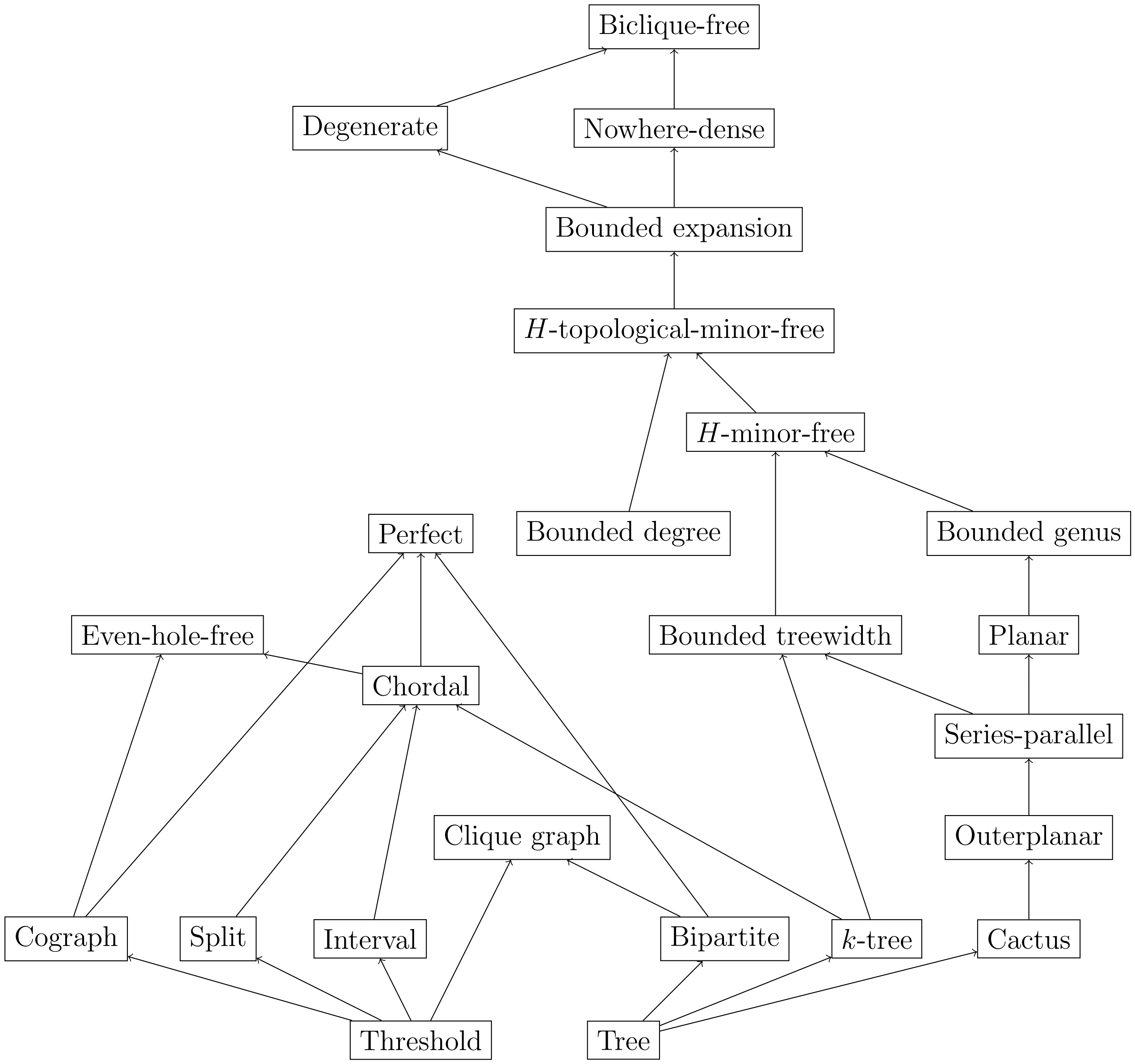

Figure 2.2: Graph classes [23, 114, 112]. Arrows indicate inclusion.

Definition 2.2.6. $\mathtt{P} = \bigcup_{k \in \mathbb{N}} TIME(n^k)$.

The class NP is another major class of decision problems in complexity theory. In NP we find all the problems that we can verify with polynomial time algorithms. That is, any yes-instance for a problem in NP has a certificate that can be checked by a *verifier* in polynomial time.

Definition 2.2.7. $\mathtt{NP} = \{L \mid \exists V$ *such that* V *is a verifier for* L *and* $V \in \mathtt{P}\}$.

Definition 2.2.8. *A language L is* NP*-complete if it satisfies the following:*

(1) L is in NP*, and*

(2) every language L' in NP *is polynomial time reducible to L.*

If a problem satisfies requirement (2), we say that it is NP*-hard. A problem is in* coNP *if and only if its complement is in* NP*.*

If we denote by $SPACE(s(n))$ the set of all problems that can be solved by Turing machines using $\mathcal{O}(s(n))$ space for some function s of the input size n, then we can define PSPACE formally as:

Definition 2.2.9. $\mathtt{PSPACE} = \bigcup_{k \in \mathbb{N}} SPACE(n^k)$.

A problem is PSPACE-complete if it can be solved using an amount of memory that is polynomial in the input length, i.e. *polynomial space*, and if every other problem that can be solved in polynomial space can be transformed to it in polynomial time. The related non-deterministic complexity class NPSPACE is similarly defined as the class of problems which can be solved by a non-deterministic algorithm that can recognize yes-instances of the problem using an amount of memory that is polynomial in the size of the input. The following relations are known between the complexity classes NPSPACE, PSPACE, NP, and P:

Proposition 2.2.10. $\mathtt{P} \subseteq \mathtt{NP} \subseteq \mathtt{PSPACE} = \mathtt{NPSPACE}$

$\mathtt{NP} \subseteq \mathtt{PSPACE}$ follows from the fact that we can try all possible solutions of a problem in NP (or coNP) in polynomial space. $\mathtt{PSPACE} = \mathtt{NPSPACE}$ follows from the celebrated result of Savitch [124]. It is widely suspected that all set containments in Proposition 2.2.10 are strict [63, 117].

When analyzing algorithms with exponential running times, we use the standard $\mathcal{O}$ notation and sometimes use the modified $\mathcal{O}^*$ notation which ignores polynomially bounded factors, i.e. polynomial in the input size.

Definition 2.2.11. *Given a function $f(n,k)$, we write $f(n,k) = \mathcal{O}^*(g(k))$ if there exist k_0, c, and n_0 such that $f(n,k) \leq g(k)n^c$ for all $k \geq k_0$ and $n \geq n_0$.*

Definition 2.2.12. *For a function $f : \mathbb{N} \to \mathbb{N}$, we write $f(n) = \mathcal{O}(g(n))$ if there exist constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.*

Definition 2.2.13. *For a function $f : \mathbb{N} \to \mathbb{N}$, we write $f(n) = \Omega(g(n))$ if there exist constants c and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.*

Definition 2.2.14. *For a function $f : \mathbb{N} \to \mathbb{N}$, we write $f(n) = \Theta(g(n))$ if there exist constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.*

2.3 Parameterized complexity

Using the framework developed by Downey and Fellows [46], we first define the general notion of a *parameterized problem.*

Definition 2.3.1. *A* parameterized problem *or a* parameterized language *is a set $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component is called the* parameter *of the problem.*

Definition 2.3.2. *For a parameterized problem L with inputs of the form (x,p), $|x| = n$ and p a positive integer:*

(i) *L is* fixed-parameter tractable, *or equivalently in* `FPT`, *if it can be decided in $f(p)n^c$ time, or equivalently* `FPT` *time, where f is an arbitrary computable function on non-negative integers and c is a constant independent of both n and p.*

(ii) *L is in the class* `XP` *if it can be decided in $n^{f(p)}$ time.*

Every problem in `FPT` has what is known as a kernel [46, 115]. Intuitively, a kernel for a problem L retains the minimum amount of information from the input L with respect to the parameter while preserving decidability. Formally:

Definition 2.3.3. Kernelization *is a polynomial time transformation that maps an instance (x,p) to an instance (x',p'), the* kernel, *such that:*

(i) *(x,p) is a yes-instance if and only if (x',p') is a yes-instance,*

(ii) $p' \leq p$, *and*

(iii) $|x'| \leq f(p)$ *for some arbitrary computable function on nonnegative integers* $f(p)$.

A kernel can be of exponential (or worse) size. If a problem has a kernelization algorithm (or equivalently a kernel) then it is in `FPT` and every `FPT` problem has a kernelization algorithm [46, 115]. We use this fact to prove the fixed-parameter tractability of several reconfiguration problems. That is, after reducing the size of an instance, we can solve the problem using exhaustive enumeration. Since the size of the kernel is bounded by a function of the parameter, such a procedure, in some cases not optimal, would still run in `FPT` time.

Showing `NP`-hardness of a parameterized problem rules out the existence of an algorithm solving the problem in $n^{\mathcal{O}(1)}$ time, assuming $\texttt{P} \neq \texttt{NP}$, but it does not rule out an algorithm running in $n^{f(p)}$ time. In other words, the problem might still be in `XP`. Many graph vertex-subset problems admit such algorithms, i.e. we simply enumerate all vertex subsets of size p and check whether they form a feasible solution. Problems that remain `NP`-hard even when the parameter is a fixed integer (above some threshold) are called *para-*`NP`*-hard*. For such problems, one cannot hope for algorithms running in `FPT` time, or even in $\mathcal{O}(n^{\mathcal{O}(f(p))})$ time. Hence, establishing whether a problem is in `XP` can serve as an initial indicator on whether it might also be in `FPT`.

However, using the classical notion of `NP`-hardness, it is not possible to distinguish between parameterized problems solvable in $n^{f(p)}$ time and parameterized problems solvable in $f(p)n^c$ time. To address this issue, Downey and Fellows [46] introduced the `W`*-hierarchy*. The hierarchy consists of a complexity class `W[t]` for every integer $t \geq 1$ such that $\texttt{W[t]} \subseteq \texttt{W[t + 1]}$ for all t. A parameterized problem L is in the class `W[t]`, $t \geq 1$, if every instance (x, p) can be transformed (in `FPT` time) to a combinatorial (Boolean) circuit that has weft at most t such that $(x, p) \in L$ if and only if there exists a satisfying assignment (to the inputs of the circuit) which assigns the value 1 (or true) to at most p inputs of the circuit [46]. Here, the *weft* denotes the largest number of logical units (or logical gates) with unbounded fan-in (i.e. unbounded number of inputs) on any path from an input unit to the output unit. Moreover, the number of logical units with bounded fan-in on any path must be bounded by a constant that holds for all instances of the problem. Downey and Fellows proved that $\texttt{FPT} \subseteq \texttt{W[1]} \subseteq \texttt{W[2]} \subseteq \ldots \subseteq \texttt{W[t]}$ and conjectured that strict containment holds. In particular, the assumption $\texttt{FPT} \subset \texttt{W[1]}$ is a natural parameterized analogue of the conjecture that $\texttt{P} \neq \texttt{NP}$. Moreover, they showed that the Independent Set problem parameterized by solution size is `W[1]`-complete and the Dominating Set problem parameterized by solution size is `W[2]`-complete. We sometimes say a problem is

W-hard, which implies that it is hard for some class in the W-hierarchy. Showing hardness in the parameterized setting is accomplished using FPT reductions.

Definition 2.3.4. *An* FPT reduction *is an* $(f(p)|x|^{\mathcal{O}(1)})$*-time transformation that maps an instance* (x, p) *to an instance* (x', p') *such that:*

(i) (x, p) *is a yes-instance if and only if* (x', p') *is a yes-instance, and*

(ii) $p' = g(p)$ *for some arbitrary computable function on nonnegative integers* $g(p)$*.*

Finally, we mention that some parameterizations can be "stronger" than others. For example, given a graph G, if G has a matching M, then the size of any vertex cover of G is least $|M|$. Hence, instead of parameterizing the VERTEX COVER problem by the solution size k, we can consider $k - |M|$ as the parameter. Such a parameterization is called an *above-guarantee parameterization* and several otherwise fixed-parameter tractable problems can become W-hard when parameterized above guarantee [101, 105]. The reader is referred to the excellent books of Niedermeier, Flum, and Grohe for more on parameterized complexity [62, 115].

2.4 Reconfiguration

For any vertex-subset problem $\mathcal{Q}$, graph G, and positive integers r_l and r_u, we consider the *reconfiguration graph* $R_{\mathcal{Q}}(G, r_l, r_u)$. A set $S \subseteq V(G)$ has a corresponding node in $V(R_{\mathcal{Q}}(G, r_l, r_u))$ if and only if S is a feasible solution for $\mathcal{Q}$ and $r_l \leq |S| \leq r_u$. We refer to *vertices* in G using lower case letters (e.g. u, v) and to the *nodes* in $R_{\mathcal{Q}}(G, r_l, r_u)$, and by extension their associated feasible solutions, using upper case letters (e.g. A, B). If $A, B \in V(R_{\mathcal{Q}}(G, r_l, r_u))$ then there exists an edge between A and B in $R_{\mathcal{Q}}(G, r_l, r_u)$ if and only if there exists a vertex $u \in V(G)$ such that $\{A \setminus B\} \cup \{B \setminus A\} = \{u\}$. Equivalently, for $A \Delta B = \{A \setminus B\} \cup \{B \setminus A\}$ the *symmetric difference* of A and B, A and B share an edge in $R_{\mathcal{Q}}(G, r_l, r_u)$ if and only if $|A \Delta B| = 1$.

We write $A \;{}_{r_l}\!\!\leftrightarrow_{r_u} B$ if there exists a path in $R_{\mathcal{Q}}(G, r_l, r_u)$, a *reconfiguration sequence*, joining A and B. We drop the subscripts when clear from context. Any reconfiguration sequence from *source* feasible solution S_s to *target* feasible solution S_t, which we sometimes denote by $\sigma = \langle S_0, S_1, \ldots, S_\ell \rangle$, for some ℓ, has the following properties:

- $S_0 = S_s$ and $S_\ell = S_t$,

- S_i is a feasible solution for $\mathcal{Q}$ for all $0 \leq i \leq \ell$,
- $|S_i \Delta S_{i+1}| = 1$ for all $0 \leq i < \ell$, and
- $r_l \leq |S_i| \leq r_u$ for all $0 \leq i \leq \ell$.

We denote the *length* of σ by $|\sigma|$. For $0 < i \leq \ell$, we say vertex $v \in V(G)$ is *added* at step/index/position/slot i if $v \notin S_{i-1}$ and $v \in S_i$. Similarly, a vertex v is *removed* at step/index/position/slot i if $v \in S_{i-1}$ and $v \notin S_i$. A vertex $v \in V(G)$ is *touched* in the course of a reconfiguration sequence if v is either added or removed at least once; it is *untouched* otherwise. A vertex is *removable* (*addable*) from feasible solution S if $S \setminus \{v\}$ ($S \cup \{v\}$) is also a feasible solution for $\mathcal{Q}$. For any pair of consecutive solutions (S_{i-1}, S_i) in σ, we say S_i (S_{i-1}) is the *successor* (*predecessor*) of S_{i-1} (S_i). A reconfiguration sequence $\sigma' = \langle S_0, S_1, \ldots, S_{\ell'} \rangle$ is a *prefix* of $\sigma = \langle S_0, S_1, \ldots, S_\ell \rangle$ if $\ell' < \ell$.

Definition 2.4.1. *For any vertex-subset problem $\mathcal{Q}$, n-vertex graph G, positive integers r_l, r_u, and ℓ, $S_s \subseteq V(G)$, and $S_t \subseteq V(G)$, we define four decision problems:*

- $\mathcal{Q}$-Reach(G, S_s, S_t, r_l, r_u): *For $S_s, S_t \in V(R_\mathcal{Q}(G, r_l, r_u))$, is there a path between S_s and S_t in $R_\mathcal{Q}(G, r_l, r_u)$?*
- $\mathcal{Q}$-Bound$(G, S_s, S_t, r_l, r_u, \ell)$: *For $S_s, S_t \in V(R_\mathcal{Q}(G, r_l, r_u))$, is there a path of length at most ℓ between S_s and S_t in $R_\mathcal{Q}(G, r_l, r_u)$?*
- $\mathcal{Q}$-Conn(G, r_l, r_u): *Is $R_\mathcal{Q}(G, r_l, r_u)$ connected?*
- $\mathcal{Q}$-Diam(G, r_l, r_u): *What is the diameter of $R_\mathcal{Q}(G, r_l, r_u)$ or of its connected components?*

When $\mathcal{Q}$ is a maximization problem, we say r_l is the *minimum allowed capacity.* For minimization problems, r_u is the *maximum allowed capacity.* Proposition 2.4.2 is a consequence of the fact that two nodes can differ by the removal or addition of a single vertex.

Proposition 2.4.2. *For any vertex-subset problem $\mathcal{Q}$ and graph G, the degree of each node in $R_\mathcal{Q}(G, r_l, r_u)$ is at most $|V(G)|$.*

Proposition 2.4.3. *For any vertex-subset problem $\mathcal{Q}$, n-vertex graph G, positive integers r_l, r_u, and ℓ, and $S_s, S_t \in V(R_\mathcal{Q}(G, r_l, r_u))$, any vertex in $\{S_s \setminus S_t\} \cup \{S_t \setminus S_s\}$ must be touched an odd number of times and any vertex not in $\{S_s \setminus S_t\} \cup \{S_t \setminus S_s\}$ must be touched an even number of times in any reconfiguration sequence of length at most ℓ from S_s to S_t. Moreover, any vertex can be touched at most $\ell - |\{S_s \setminus S_t\} \cup \{S_t \setminus S_s\}| + 1$ times.*

For a hereditary graph property Π, it will be useful to consider two reconfiguration graphs; the Π-*reconfiguration graph* of G, $R_\Pi(G, r_l, r_u)$, has a node for each $S \subseteq V(G)$ such that $r_l \leq |S| \leq r_u$ and $G[S]$ has property Π, and the $\overline{\Pi}$-*reconfiguration graph* of G, $R_{\overline{\Pi}}(G, r_l, r_u)$, has a node for each $S \subseteq V(G)$ such that $r_l \leq |S| \leq r_u$ and $G[V(G) \setminus S]$ has property Π. Given $R_\Pi(G, k, n)$, we can obtain $R_{\overline{\Pi}}(G, 0, n - k)$ by replacing the set corresponding to each node in $R_\Pi(G, k, n)$ by its (set-wise) complement.

Definition 2.4.4. *For any polynomially decidable hereditary graph property Π, n-vertex graph G, positive integers k and ℓ, $S_s \subseteq V(G)$, and $S_t \subseteq V(G)$, we define the following decision problems:*

- Π-Del-Reach(G, S_s, S_t, k): *For $S_s, S_t \in V(R_{\overline{\Pi}}(G, 0, k))$, is there a path between S_s and S_t in $R_{\overline{\Pi}}(G, 0, k)$?*
- Π-Del-Bound(G, S_s, S_t, k, ℓ): *For $S_s, S_t \in V(R_{\overline{\Pi}}(G, 0, k))$, is there a path of length at most ℓ between S_s and S_t in $R_{\overline{\Pi}}(G, 0, k)$?*
- Π-Sub-Reach(G, S_s, S_t, k): *For $S_s, S_t \in V(R_\Pi(G, k, n))$, is there a path between S_s and S_t in $R_\Pi(G, k, n)$?*
- Π-Sub-Bound(G, S_s, S_t, k, ℓ): *For $S_s, S_t \in V(R_\Pi(G, k, n))$, is there a path of length at most ℓ between S_s and S_t in $R_\Pi(G, k, n)$?*

In the parameterized setting, when dealing with the Π-Del-Reach or Π-Sub-Reach (or more generally $\mathcal{Q}$-Reach) problems, a natural parameter to consider is the maximum allowed capacity, i.e. the value k. Similarly, for Π-Del-Reach and Π-Sub-Reach (or $\mathcal{Q}$-Reach) problems, we shall consider k, now the minimum allowed capacity, as one possible parameter (since r_u will usually be equal to n for such problems, parameterizing by n provides no further insights). On the other hand, there are two natural parameters that come into play when studying bounded reachability variants: k and ℓ. Therefore, any combination of these two values can be considered as a parameter, i.e. k, ℓ, or $k + \ell$. Although these are not the only possible parameterizations that we will consider for the problems in question, they are the most "natural" ones. For example, when parameterizing the reachability problem by k, we are interested in whether there exists an algorithm which can solve the problem in $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ time, for some computable function f. When parameterizing the bounded reachability problem by k alone or by ℓ alone, the goal is an algorithm running in $\mathcal{O}(h(k)(n\ell)^{\mathcal{O}(1)})$ time or $\mathcal{O}(h'(\ell)(nk)^{\mathcal{O}(1)})$ time, respectively, for some computable functions h and h'. Finally, when parameterizing the bounded reachability problem by $k+\ell$, we seek an algorithm running in $\mathcal{O}(g(k, \ell)n^{\mathcal{O}(1)})$ time, for some computable function g. If such algorithms exist, then depending on the problem and instance at hand, it is very likely for one to be much more "efficient" than the others in practice. For instance, an algorithm running in $2^k n^\ell$ time would terminate much faster than a $2^\ell n^k$ time algorithm if $n = 100$, $k = 10$, and $\ell = 5$. Note that if a problem is W-hard parameterized by $k + \ell$, then it remains W-hard parameterized by k alone or by ℓ alone; the converse is however not true.

In most cases, the definitions of the reachability and bounded reachability variants of a problem follow naturally from the definition of the problem itself (appendix). We use Vertex Cover as an example below, which corresponds to Π-Del for Π the collection of all edgeless graphs.

VC-Reach

Input:	A graph G, positive integer k, and two vertex covers S_s and S_t of G of size at most k
Question:	Is there a path from S_s to S_t in $R_{VC}(G, 0, k)$?

VC-Bound

Input:	A graph G, positive integers k and ℓ, and two vertex covers S_s and S_t of G of size at most k
Question:	Is there a path of length at most ℓ from S_s to S_t in $R_{VC}(G, 0, k)$?

We first observe that for any polynomially decidable graph property, Π-Del-Bound and Π-Sub-Bound are in XP when parameterized by ℓ; we conduct breadth-first search on the reconfiguration graph starting at S_s, stopping either upon discovery of S_t or upon completing the exploration of ℓ levels. Proposition 2.4.2 implies a bound of at most n^ℓ vertices to explore in total.

Proposition 2.4.5. *For any polynomially decidable graph property* Π*,* Π-Del-Bound $\in$ XP *and* Π-Sub-Bound $\in$ XP *when parameterized by* ℓ*.*

The situation is different for Π-Del-Reach and Π-Sub-Reach parameterized by k, as the latter is a maximization problem while the former is a minimization problem. In other words, we can enumerate all feasible solutions of size at most k in $\mathcal{O}(n^k)$ time to solve Π-Del-Reach (or Π-Del-Bound) parameterized by k but the same "easy" argument does not hold for Π-Sub-Reach (or Π-Sub-Bound) parameterized by k.

Proposition 2.4.6. *For any polynomially decidable graph property* Π*,* Π-Del-Reach $\in$ XP *and* Π-Del-Bound $\in$ XP *when parameterized by* k*.*

Already, Proposition 2.4.6 implies that, in some sense, dealing with reachability and bounded reachability variants of maximization problems might be "harder" in the parameterized setting.

For a graph G, and $S_s, S_t \subseteq V(G)$, we partition $V(G)$ into the sets $C_{st} = S_s \cap S_t$ (vertices common to S_s and S_t), $S_{s\setminus t} = S_s \setminus S_t$ (vertices to be removed from S_s in the course of a reconfiguration sequence), $S_{t\setminus s} = S_t \setminus S_s$ (vertices to be added to form S_t), and $O_{st} = V(G) \setminus (S_s \cup S_t)$ (all other vertices). Furthermore, we can partition C_{st} into two sets C_F and $C_M = C \setminus C_F$, where a vertex is in C_F if and only if it is in every feasible solution of size bounded by k. A vertex $v \in C_F$ is said to be *fixed* or *frozen*. The following proposition is a consequence of the definitions above, the fact that Π is hereditary, and the observations that $G[S_{s\setminus t}]$ and $G[O_{st}]$ are both subgraphs of $G[V(G) \setminus S_t]$, and $G[S_{t\setminus s}]$ and $G[O_{st}]$ are both subgraphs of $G[V(G) \setminus S_s]$.

Proposition 2.4.7. *For any hereditary property Π, graph G, and $S_s, S_t \subseteq V(G)$ such that $G[V(G) \setminus S_s], G[V(G) \setminus S_t] \in \Pi$, the graphs $G[O_{st}]$, $G[S_{s\setminus t}]$, and $G[S_{t\setminus s}]$ are all in Π.*

In any reconfiguration sequence, each vertex in $S_{s\setminus t}$ must be removed and each vertex in $S_{t\setminus s}$ must be added. In fact, since ℓ implies a bound on the total number of vertices that can be touched in a reconfiguration sequence, setting $\ell = |S_{s\setminus t}| + |S_{t\setminus s}|$ drastically simplifies the bounded reachability problem.

Proposition 2.4.8. *For any polynomially decidable hereditary graph property Π, if $\ell = |S_{s\setminus t}| + |S_{t\setminus s}|$, then* Π-Del-Bound *and* Π-Sub-Bound *can be solved in $\mathcal{O}^*(2^\ell)$ time, and hence are fixed-parameter tractable when parameterized by ℓ.*

Proof. Since each vertex in $S_{t\setminus s}$ must be added and each vertex in $S_{s\setminus t}$ removed, in ℓ steps we can touch each vertex in $S_{s\setminus t} \cup S_{t\setminus s}$ exactly once; all vertices in $V(G) \setminus (S_{s\setminus t} \cup S_{t\setminus s})$ remain untouched.

Any node in the path between S_s and S_t in $R_\Pi(G, k, n)$ represents a set $S \cup B$, where B is a subset of $S_{s\setminus t} \cup S_{t\setminus s}$. As $|S_{s\setminus t}| + |S_{t\setminus s}| = \ell$, there are only 2^ℓ choices for B. Our problem then reduces to finding the shortest path between S_s and S_t in the subgraph of $R_\Pi(G, k, n)$ induced on the 2^ℓ relevant nodes; the bound follows from the fact that the number of edges is at most $2^\ell |V(G)|$, a consequence of Proposition 2.4.2. The same argument holds for $R_{\overline{\Pi}}(G, 0, k)$. □

As we shall see in subsequent chapters, this relationship between ℓ and the size of $S_{s\setminus t} \cup S_{t\setminus s}$ can have major impact on the parameterized complexity of bounded reachability problems.

Intuitively, the larger the (absolute) difference between the two, the harder the problem becomes.

We conclude with a proposition which relates the parameterized complexity of Π-Del-Bound and Π-Sub-Bound when both are parameterized by ℓ; proving fixed-parameter tractability of either problem on some graph class will be enough to imply fixed-parameter tractability of the other.

Proposition 2.4.9. *Given* Π *and a collection of graphs* $\mathcal{C}$*,* Π-Del-Bound *parameterized by* ℓ *is fixed-parameter tractable on* $\mathcal{C}$ *if and only if* Π-Sub-Bound *is.*

Proof. Given an instance (G, S_s, S_t, k, ℓ) of Π-Sub-Bound, where $G \in \mathcal{C}$, we solve the Π-Del-Bound instance $(G, V(G) \setminus S_s, V(G) \setminus S_t, n - k, \ell)$. Note that the parameter ℓ remains unchanged.

It is not hard to see that there exists a path between the nodes corresponding to S_s and S_t in $R_{\Pi}(G, k, n)$ if and only if there exists a path of the same length between the nodes corresponding to $V(G) \setminus S_s$ and $V(G) \setminus S_t$ in $R_{\overline{\Pi}}(G, 0, n - k)$. The same argument holds for the other direction. □

Chapter 3

A case study in domination

In this chapter, we consider the reachability and bounded reachability variants of the Dominating Set problem, formally defined below:

DS-Reach	
Input:	A graph G, positive integer k, and two dominating sets D_s and D_t of G of size at most k
Question:	Is there a path from D_s to D_t in $R_{DS}(G, 0, k)$?

DS-Bound	
Input:	A graph G, positive integers k and ℓ, and two dominating sets D_s and D_t of G of size at most k
Question:	Is there a path of length at most ℓ from D_s to D_t in $R_{DS}(G, 0, k)$?

The Dominating Set problem is a central problem with numerous applications in many areas and has been extensively studied from both the classical and parameterized complexity viewpoints. The problem is known to be `NP`-complete and `W[2]`-complete on general graphs [46, 63]. In Section 3.1, we show, using "standard" reductions, that both DS-Reach and DS-Bound are `PSPACE`-complete, even when restricted to planar graphs of degree at most six, bipartite graphs, and split graphs. Having established hardness, the next natural step is to investigate the complexity of both problems when restricted to other sparse graph classes, e.g. graphs of bounded bandwidth (hence pathwidth and treewidth). We show, again, that both problems remain `PSPACE`-complete. We then consider the DS-Reach problem parameterized by k as well as the DS-Bound problem parameterized by $k + \ell$ and show, once more, that both problems are `W[2]`-hard. Unfortunately, these results very likely rule out the existence of algorithms solving the former problem in $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ or $\mathcal{O}(f'(tw(G))(nk)^{\mathcal{O}(1)})$ time and the latter problem in $\mathcal{O}^*(h(k,\ell)n^{\mathcal{O}(1)})$ or $\mathcal{O}^*(h'(tw(G))(nk\ell)^{\mathcal{O}(1)})$ time, for any computable functions f, f', h, and h'. An interesting aspect of our W-hardness reduction is that it suggests, in some sense, that solving either the DS-Reach problem or the DS-Bound problem is at least as hard as solving the DS problem itself; we formalize this notion in Chapter 4 and show that it holds for a large number of graph problems.

To better understand the source of hardness, we tackle the structure of the reconfiguration graph in Section 3.2. We show that there exists an infinite family $\mathcal{F}_{bw}$ of graphs of bounded bandwidth such that for each graph G in the family there exists a value of k, namely the size of a minimum dominating set of G plus one, for which the corresponding reconfiguration graph has exponential diameter. In previous work, Haas and Seyffarth [73] demonstrated that the reconfiguration graph $R_{DS}(G, 0, k)$ is connected when $k = n - 1$ and G has at least two non-adjacent edges (the graph is trivially connected for $k = n$), or when k is one greater than the maximum cardinality of any minimal dominating set of G and G is non-trivially bipartite or chordal. They left as an open question whether the latter results could be extended to all graphs. We extend their work by showing that the reconfiguration graph is connected and of linear diameter for $k = n - \mu$ for any input graph with a matching of size least $\mu + 1$, for any nonnegative integer μ. We also answer their open question negatively by giving a series of counterexamples demonstrating that, when k is one greater than the maximum cardinality of any minimal dominating set of the input graph, the reconfiguration graph is not guaranteed to be connected, even if the input graph is restricted to be planar, of bounded treewidth, or b-partite for $b \geq 3$.

One crucial observation that follows from both the construction of the infinite family $\mathcal{F}_{bw}$ (Section 3.2) as well as the PSPACE-hardness proof for graphs of bounded bandwidth (Lemma 3.1.6) is that in both cases, the size of the dominating sets are "quite large" compared to n, the size of the input graph. That is, every bucket in a bucket arrangement of G (or bag in a tree decomposition) contains at least one vertex of any dominating set of the graph. This suggests that adding the bandwidth (or treewidth) of the graph to our parameter might make the problem fixed-parameter tractable. We show, in Section 3.3, that this is indeed the case. What we will prove is in fact much more general. We show fixed-parameter tractability of both problems on biclique-free graphs, a class of graphs which includes a large number of known sparse graph classes (Figure 2.2) and is the largest class on which the Dominating Set problem is known to be fixed-parameter tractable parameterized by k. Finally, we also consider polynomial-time solvable instances. In contrast to the fixed-parameter tractability result of Theorem 3.3.11, where the graph class under consideration is "quite large", we are only able to prove DS-Reach solvable in polynomial time on paths and trees. Even the (classical) complexity of solving DS-Bound on paths remains open. Our polynomial-time algorithms make use of what we call a canonical solution, i.e. a well-defined node in the reconfiguration graph which is connected by a path to any other node in the reconfiguration graph. Hence, solving the DS-Reach problem is simplified to proving the existence of a canonical solution.

3.1 Hardness

We use $\gamma(G)$ to denote the minimum cardinality of any dominating set of a graph G. Similarly, $\Gamma(G)$ is the maximum cardinality of any minimal dominating set of G. For a vertex $u \in V(G)$ and a dominating set D of G, we say u is *dominated* by $v \in D$ if $u \notin D$ and u is adjacent to v. For a vertex v in a dominating set D, a *private neighbor* of v is a vertex dominated by v and not dominated by any other vertex in D; the *private neighborhood* of v is the set of all private neighbors of v.

A close relative to the Dominating Set problem which will be useful in proving some of our results is known as the Red-Blue Dominating Set (RBDS) problem. We define this problem and its reachability and bounded reachability variants below. Given a bipartite graph G with $V(G)$ partitioned into two sets R (red) and B (blue), a set $D \subseteq R$ is a *red-blue dominating set* of G if $N(D) = B$.

Red-Blue Dominating Set (RBDS)

Input:	A bipartite graph G with bipartition (R, B) and a positive integer k
Question:	Is there $D \subseteq R$ such that $\|D\| \leq k$ and $N[D] = B$?

RBDS-Reach

Input:	A bipartite graph G with bipartition (R, B), positive integer k, and two red-blue dominating sets D_s and D_t of G of size at most k
Question:	Is there a path from D_s to D_t in $R_{RBDS}(G, 0, k)$?

RBDS-Bound

Input:	A bipartite graph G with bipartition (R, B), positive integers k and ℓ, and two red-blue dominating sets D_s and D_t of G of size at most k
Question:	Is there a path of length at most ℓ from D_s to D_t in $R_{RBDS}(G, 0, k)$?

DS and RBDS are in fact equivalent. In the following lemma, we show that the reachability and bounded reachability variants of both problems are also equivalent.

Lemma 3.1.1. DS-Reach *is equivalent to* RBDS-Reach *and* DS-Bound *is equivalent to* RBDS-Bound*.*

Proof. Given an instance (G, D_s, D_t, k) of DS-Reach, where $V(G) = \{v_1, \ldots, v_n\}$, we construct a bipartite graph G' with bipartition (R, B) as follows: We let $R = \{r_1, \ldots, r_n\}$, $B = \{b_1, \ldots, b_n\}$, and we make a red vertex $r_i \in R$, $1 \leq i \leq n$, adjacent to all blue vertices b_j, $1 \leq j \leq n$, such that $v_j \in N[v_i]$. We let $D'_s = \{r_i \mid v_i \in D_s \wedge 1 \leq i \leq n\}$ and $D'_t = \{r_i \mid v_i \in D_t \wedge 1 \leq i \leq n\}$, i.e. the red copies of vertices in D_s and D_t, respectively. It is not hard to see that (G, D_s, D_t, k) is a yes-instance of DS-Reach if and only if (G', D'_s, D'_t, k) is a yes-instance of RBDS-Reach. Moreover, the length of reconfiguration sequences is preserved and hence (G, D_s, D_t, k, ℓ) is a yes-instance of DS-Bound if and only if $(G', D'_s, D'_t, k, \ell)$ is a yes-instance of RBDS-Bound.

For the other direction, given an instance (G, D_s, D_t, k) of RBDS-Reach, where $V(G) = R \cup B$, $R = \{r_1, \ldots, r_{|R|}\}$, and $B = \{b_1, \ldots, b_{|B|}\}$, we construct a graph G' as follows. We let $V(G') = R_1 \cup B_1 \cup \ldots \cup B_{k+2} \cup \{v_1, \ldots, v_{k+3}\}$, where:

- R_1 is a copy of R,
- B_i is a copy of B for all i, $1 \leq i \leq k+1$ ($(k+2)$ copies in total), and
- $\{v_1, \ldots, v_{k+3}\}$ are $k+3$ new vertices.

We make every vertex in R_1 adjacent to the $k+2$ copies of its neighbors in G. Finally, we make v_1 adjacent to all vertices in $R_1 \cup \{v_2, \ldots, v_{k+3}\}$. We let $D'_s = D_s \cup \{v_1\}$ and $D'_t = D_t \cup \{v_1\}$. We claim that (G, D_s, D_t, k) is a yes-instance of RBDS-Reach if and only if $(G', D'_s, D'_t, k+1)$ is a yes-instance of DS-Reach. Since v_1 has $k+2$ degree-one neighbors, it must be part of any dominating set of G' of size $k+1$ or less, i.e. v_1 is frozen. Hence, all vertices in R_1 are dominated by v_1. Moreover, any shortest reconfiguration sequence from D'_s to D'_t does not add a vertex $u \in B_1 \cup \ldots \cup B_{k+2}$. Since the remaining $k+1$ copies of u have to be dominated by a vertex w in R_1 which also dominates u, we can ignore the addition of u and obtain a shorter reconfiguration sequence which touches only vertices in R_1. Therefore, the length of shortest reconfiguration sequences is preserved which implies that (G, D_s, D_t, k, ℓ) is a yes-instance of RBDS-Bound if and only if $(G', D'_s, D'_t, k, \ell)$ is a yes-instance of DS-Bound. □

In Lemmas 3.1.2, 3.1.3, 3.1.4, and 3.1.6, we show PSPACE-completeness of DS-Reach and DS-Bound on various graph classes by simply proving PSPACE-hardness of DS-Reach on the corresponding class. Clearly, both DS-Reach and DS-Bound are in NPSPACE, and hence in PSPACE by Savitch's theorem. Moreover, setting $\ell = 2^n$, where ℓ is assumed to be encoded in binary, gives a trivial reduction from DS-Reach to DS-Bound.

Lemma 3.1.2. DS-Reach *and* DS-Bound *are* PSPACE-*complete on planar graphs of degree at most six.*

Proof. The fact that DS-Reach is PSPACE-hard on planar graphs of degree at most six follows from the classical reduction from VC to DS [63]. That is, given an instance (G, S_s, S_t, k) of VC-Reach, for every edge uv in $E(G)$ we add a new vertex e_{uv} and two edges ue_{uv} and $e_{uv}v$ to obtain the graph G'. We let $(G', D_s = S_s, D_t = S_t, k)$ denote the corresponding DS-Reach instance. Any reconfiguration sequence from D_s to D_t which adds a vertex e_{uv} can be modified not to do so; we can replace the addition of e_{uv} by one of its two neighbors and obtain a (possibly shorter) reconfiguration sequence which only touches vertices in G. As VC-Reach is PSPACE-hard on planar graphs of degree at most three [77], we obtain the desired result. □

Lemma 3.1.3. DS-Reach *and* DS-Bound *are* PSPACE-*complete on split graphs.*

Proof. We again give a reduction from VC-Reach. Given an instance (G, S_s, S_t, k) of VC-Reach, where $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$, we construct a split graph G' as follows. We let $V(G') = A \cup B_1 \cup \ldots \cup B_{k+2} \cup \{c_1, \ldots, c_{k+2}\}$, where $A = \{a_1, \ldots, a_{n+1}\}$ and $B_i = \{b^i_1, \ldots, b^i_m\}$, $1 \leq i \leq k+2$. We add all edges between vertices in A (i.e. A is a clique). For each vertex $b^i_j \in B_i$, $1 \leq i \leq k+2$ and $1 \leq j \leq m$, we add two edges connecting b^i_j to the two vertices in A which correspond to the endpoints of the edge e_j in G. Finally, we add an edge $a_{n+1}c_i$, for all $1 \leq i \leq k+2$. In the resulting DS-Reach instance $(G', D_s = S_s \cup \{a_{n+1}\}, D_t = S_t \cup \{a_{n+1}\}, k+1)$, any dominating set of G' of size at most $k+1$ must include a_{n+1}, i.e. a_{n+1} is frozen. Hence, all vertices in A are dominated. Moreover, any reconfiguration sequence from D_s to D_t which adds a vertex $u \in B_1 \cup \ldots \cup B_{k+2}$ can be modified not to do so. Since the remaining $k+1$ copies of u have to be dominated by some vertex w in A which also dominates u, we can ignore the addition of u and obtain a shorter reconfiguration sequence which only touches vertices in A, as needed. □

Lemma 3.1.4. DS-Reach *and* DS-Bound *are* PSPACE-*complete on bipartite graphs.*

Proof. To show PSPACE-hardness on bipartite graphs, we can use a reduction from VC-Reach to RBDS-Reach, which by Lemma 3.1.1 implies PSPACE-hardness of DS-Reach. Given an instance (G, S_s, S_t, k) of VC-Reach, where $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$, we construct a bipartite graph G' with partitions R and B as follows. We let $R = \{r_1, \ldots, r_n\}$ and $B = \{b_1, \ldots, b_m\}$. Every vertex $b_i \in B$, $1 \leq i \leq m$, has two neighbors in R which correspond to the endpoints of the edge e_i in G, $1 \leq i \leq m$. It is not hard to see that we have a one-to-one correspondence between vertex covers of G and red-blue dominating sets of G'. □

To show PSPACE-hardness on graphs of bounded bandwidth, we first define a very useful problem, introduced by Wrochna [133], whose simplicity allows for easy reductions to various reachability and bounded reachability problems. Given a pair $\mathcal{H} = (\Sigma, \mathcal{R})$, where Σ is an alphabet and $\mathcal{R} \subseteq \Sigma^2$ a binary relation between symbols, we say that a word over Σ is an $\mathcal{H}$-*word* if every pair of consecutive symbols is in the relation. If one looks at $\mathcal{H}$ as a digraph (possibly with loops), a word is an $\mathcal{H}$-word if and only if it is a walk in $\mathcal{H}$. The $\mathcal{H}$-Word Reach problem asks whether two given $\mathcal{H}$-words of equal length n can be transformed into one another (in any number of steps) by changing one symbol at a time so that all intermediary steps also result in $\mathcal{H}$-words.

Lemma 3.1.5 ([110, 133])**.** *There exists a pair $\mathcal{H} = (\Sigma, \mathcal{R})$ for which $\mathcal{H}$-Word Reach is PSPACE-complete.*

Lemma 3.1.6. DS-Reach *and* DS-Bound *are* PSPACE-*complete on graphs of bounded bandwidth, pathwidth, or treewidth.*

Proof. We let $\mathcal{H} = (\Sigma, \mathcal{R})$ be the pair whose existence was shown in Lemma 3.1.5, $\Sigma = \{s_1, \ldots, s_{|\Sigma|}\}$, and $\mathcal{R} \subseteq \Sigma^2$. We give a reduction from $\mathcal{H}$-Word Reach, for $\mathcal{H}$-words of size n, to DS-Reach by constructing a graph G_n as follows.

The vertex set of G_n consists of the disjoint union of n vertex-disjoint cliques $C_1, \ldots, C_n$, each of size $|\Sigma|$, and $n-1$ vertex-disjoint independent sets $R_1, \ldots, R_{n-1}$, each of size at most $2|\Sigma|^2$. That is, for every symbol in Σ there is a corresponding vertex in each C_i, $1 \leq i \leq n$, and we let $C_i = \{c^i_1, \ldots, c^i_{|\Sigma|}\}$. Given a vertex $c^i_j \in C_i$ and a vertex $c^{i+1}_p \in C_{i+1}$, $1 \leq i < n$ and $1 \leq j, p \leq |\Sigma|$, we say c^i_j and c^{i+1}_p are *related* if $(s_j, s_p) \in \mathcal{R}$; they are *unrelated* otherwise. For every pair of sets C_i and C_{i+1}, $1 \leq i < n$, we will add a set of at most $2|\Sigma|^2$ vertices, denoted by R_i. The number of vertices in R_i is equal to two times the number of unrelated pairs of vertices in C_i and C_{i+1}. That is, if c^i_j and c^{i+1}_p are unrelated, we have two vertices $a^i_{jp}, b^i_{jp} \in R_i$ and there is an edge between a^i_{jp} (b^i_{jp}) and every vertex in $C_i \cup C_{i+1} \setminus \{c^i_j, c^{i+1}_p\}$. In other words, a^i_{jp} and b^i_{jp} dominate all but two vertices in $C_i \cup C_{i+1}$, namely c_j^i and c_p^{i+1}. The construction of G_n for $\Sigma = \{s_1, s_2, s_3\}$, $\mathcal{R} = \Sigma^2 \setminus \{(s_1, s_1)\}$, and $n = 4$ is illustrated in Figure 3.1. The sets $C_i \cup R_i$ give a bucket arrangement where each bucket has size $|\Sigma| + 2|\Sigma|^2$.

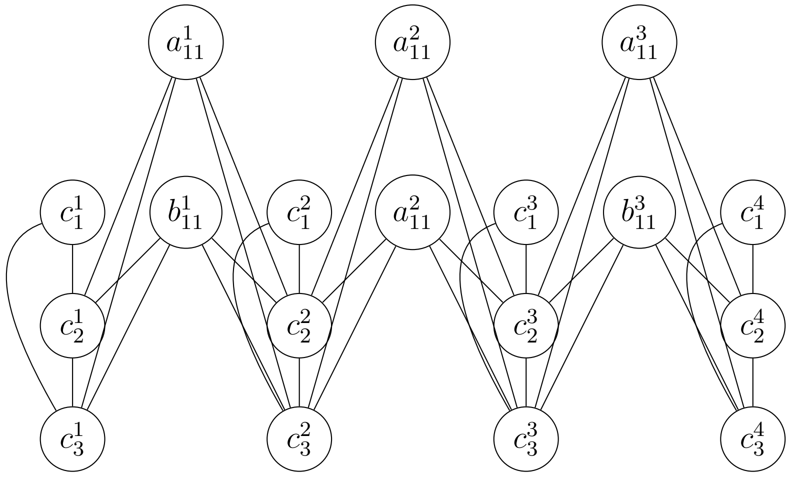

Figure 3.1: Example of a reduction from $\mathcal{H}$-Word-Reach to DS-Reach.

We start by proving the following claims.

Claim 3.1.7. *For $1 \leq i < n$ and $1 \leq p_1, p_2 \leq |\Sigma|$, any two vertices $c_{p_1} \in C_i$ and $c_{p_2} \in C_{i+1}$ form a dominating set of $G_n[C_i \cup R_i \cup C_{i+1}]$ if and only if they are related.*

Proof. When c_{p_1} and c_{p_2} are related, every vertex in R_i is adjacent to c_{p_1}, c_{p_2}, or both. Moreover, C_i and C_{i+1} are cliques and hence the claim follows.

Conversely, if c_{p_1} and c_{p_2} are not related then, by construction, there exist at least two vertices in R_i which are dominated by neither c_{p_1} nor c_{p_2}. □

Claim 3.1.8. *The cardinality of any dominating set of G_n is at least n.*

Proof. We prove the claim by induction on n, for $\mathcal{H} = (\Sigma, \mathcal{R})$ fixed. When $n = 1$, $G_1 = C_1$ is a clique of size $|\Sigma|$ and the statement trivially holds. For G_2, we note that every pair of vertices $a^1_{jp}, b^1_{jp} \in R_1$, $1 \leq j, p \leq |\Sigma|$, dominates all but two vertices in $C_1 \cup C_2$, namely c^1_j and c^2_p. Hence any dominating set of G_2 that does not include one vertex from C_1 and one vertex from C_2 must be of size three or more. Moreover, since there are no edges between vertices in C_1 and vertices in C_2, any dominating of G_2 must be of cardinality at least two, which exists whenever there is at least one pair of related vertices in C_1 and C_2

(Claim 3.1.7). Now consider the graph G_{n+1}. By the inductive hypothesis, the minimum cardinality of any dominating set of $G[V(G_{n+1}) \setminus \{R_n \cup C_{n+1}\}]$ is n. Since there are no edges between vertices in C_n and vertices in C_{n+1}, at least one additional vertex is required to dominate vertices in $R_n \cup C_{n+1}$, as needed. □

Claim 3.1.9. *For $n \geq 2$, a set $D \subseteq V(G_n)$ of size n is a dominating set of G_n if and only if the following conditions holds:*

(1) $D \cap C_i = c_{p_i}$, for $1 \leq i \leq n$ and $1 \leq p_i \leq |\Sigma|$.

(2) The vertices $c_{p_i} = D \cap C_i$ and $c_{p_{i+1}} = D \cap C_{i+1}$ are related, for $1 \leq i < n$ and $1 \leq p_i \leq |\Sigma|$.

Proof. If conditions (1) and (2) hold, then by Claim 3.1.7, every two vertices c_{p_i} and $c_{p_{i+1}}$ in D, $1 \leq i < n$, are related and are therefore a dominating set of $G_n[C_i \cup R_i \cup C_{i+1}]$. Hence, D is a dominating of G_n.

We prove the other direction by induction on n using arguments similar to those made in the proof of Claim 3.1.8. When $n = 2$, the statement follows by combining Claims 3.1.7 and 3.1.8 with the fact that no two vertices of R_1 can dominate $V(G_2)$. Now consider the graph G_{n+1}. By the inductive hypothesis, D must include exactly one vertex from each set C_i, $1 \leq i \leq n$, and $c_{p_i} = D \cap C_i$ and $c_{p_{i+1}} = D \cap C_{i+1}$ must be related. Since there are no edges between vertices in C_n and vertices in C_{n+1}, the additional vertex required to dominate the non-dominated vertices in $R_n \cup C_{n+1}$ must be in C_{n+1} and must be related to the vertex in C_n (Claim 3.1.7). This completes the proof. □

An immediate consequence of Claim 3.1.9 is that any subset of $V(G_n)$ of size n is a dominating set of G_n if and only if its vertices correspond to an $\mathcal{H}$-word, giving a bijection between dominating sets of G_n of size n and $\mathcal{H}$-words of length n. Formally, if D is a dominating set of size n, then D picks exactly one vertex from each C_i, $1 \leq i \leq n$, and we can therefore write it as $D = \{c_{p_1}, \ldots, c_{p_n}\}$, where $1 \leq p_1, \ldots, p_n \leq |\Sigma|$. Moreover, the vertices c_{p_i} and $c_{p_{i+1}}$ are related, for all $1 \leq i < n$. Hence, $s_{p_1} \ldots s_{p_n}$ is an $\mathcal{H}$-word of length n.

Given an instance of $\mathcal{H}$-Word Reach, where $\mathcal{H} = (\Sigma, \mathcal{R})$, $w_s, w_t \in \Sigma^*$ are two $\mathcal{H}$-words and $|w_s| = |w_t| = n$, we construct the instance $(G_n, D_s, D_t, n+1)$ of DS-Reach, where D_s and D_t are the dominating sets of size n that correspond to w_s and w_t, respectively. Any reconfiguration sequence between such dominating sets starts by adding a vertex (since G_n has no dominating set of size $n-1$) and then removing another (since dominating sets larger than $n+1$ are not allowed), which corresponds to changing one symbol of an $\mathcal{H}$-word.

This gives a one-to-one correspondence between reconfiguration sequences of $\mathcal{H}$-words and reconfiguration sequences (of exactly twice the length) between dominating sets of size n. The instances are thus equivalent. $\square$

The reachability and bounded reachability variants of the SAT problem are `PSPACE`-complete [67]. And, as mentioned by Ito et al. [81], many reachability and bounded reachability problems can be shown to be `PSPACE`-complete via extensions, often complicated, of the original `NP`-completeness proofs. This was exemplified in some of our earlier hardness proofs (e.g. Lemma 3.1.2). Moreover, the simplicity of the $\mathcal{H}$-Word Reach problem introduced by Wrochna [133] makes it, we believe, another central problem to add to the "toolbox" of problems to consider when attempting to prove `PSPACE`-completeness of reachability or bounded reachability problems on sparse graph classes. As we shall see next, the nature of reductions changes when viewing such problems from a parameterized complexity perspective. A major difference when considering bounded reachability problems parameterized by the length ℓ of reconfiguration sequences is that we often need to give reductions from "static" to "dynamic" problems, while keeping this parameter bounded.

Lemma 3.1.10. DS-Reach *parameterized by k and* DS-Bound *parameterized by $k + \ell$ are* `W[2]`*-hard.*

Proof. We give a reduction from Dominating Set; for (G, t) an instance of Dominating Set, we form G' as the disjoint union of two graphs G'_1 and G'_2.

We form G'_1 from $t+2$ $(t+1)$-cliques C_0 (the *outer clique*) and $C_1, \ldots, C_{t+1}$ (the *inner cliques*); $V(C_0) = \{o_1, \ldots, o_{t+1}\}$ and $V(C_i) = \{w_{(i,0)}, w_{(i,1)}, \ldots w_{(i,t)}\}$ for $1 \leq i \leq t+1$. The edge set of G'_1 contains not only the edges of the cliques but also $\{\{o_j, w_{(i,j)}\} \mid 1 \leq i \leq t+1, 0 \leq j \leq t\}$; the graph to the left in Figure 3.2 illustrates G'_1 for $t = 2$. Any dominating set that does not contain all vertices in the outer clique must contain a vertex from each inner clique.

To create G'_2, we first define G^+ to be the graph formed by adding a universal vertex to G, where we assume without loss of generality that $V(G) = \{v_1, \ldots, v_{|V(G)|}\}$. We let $V(G'_2) = \cup_{0 \leq i \leq t} V(H_i)$, where $H_0, \ldots, H_t$ are $t+1$ copies of G^+; we use u_i to denote the universal vertex in H_i and $v_{(i,j)}$ to denote the copy of v_j in H_i, $1 \leq j \leq |V(G)|$, $0 \leq i \leq t$. The edge set consists of edges between each non-universal vertex $v_{(0,j)}$ in H_0 and, in each H_i, the universal vertex, its image, and the images of its neighbours in G, or more formally $E(G'_2) = \{\{v_{0,j}, u_i\} \mid 1 \leq j \leq |V(G)|, 1 \leq i \leq t\} \cup \{\{v_{0,j}, v_{i,j}\} \mid 1 \leq j \leq |V(G)|, 1 \leq i \leq t\} \cup \{\{(v_{0,j}, v_{i,k}\} \mid 1 \leq j \leq |V(G)|, 1 \leq i \leq t, (v_j, v_k) \in E(G)\}$. The graph to the right in Figure 3.2 illustrates part of G'_2, where universal vertices are shown in white and, for

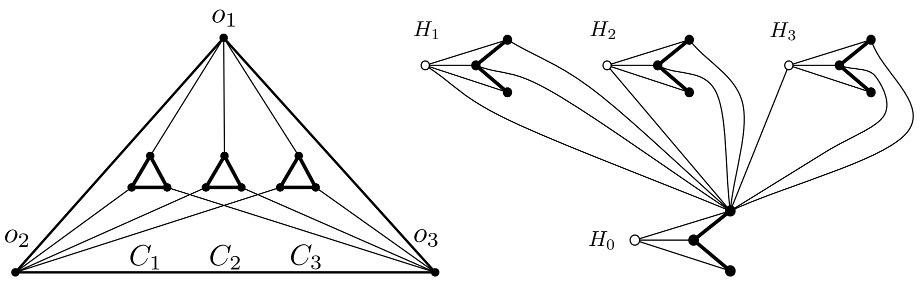

Figure 3.2: Graphs used for the dominating set reduction

the sake of readability, the only edges outside of G^+ shown are those adjacent to a single vertex in H_0.

We form an instance $(G', D_s, D_t, 3t+2, 6t+4)$ of DS-Bound, where $D_s = \{u_i \mid 0 \leq i \leq t\} \cup V(C_0)$ and $D_t = \{u_i \mid 0 \leq i \leq t\} \cup \{w_{i,i-1} \mid 1 \leq i \leq t+1\}$. Both D_s and D_t are dominating sets, as each universal vertex u_i dominates H_i as well as H_0 and $V(G'_1)$ is dominated by the outer clique in S and by one vertex from each inner clique in T. Clearly $|D_s| = |D_t| = 2t+2$.

We claim that G has a dominating set of size t if and only if there is a path of length $6t+4$ from D_s to D_s in $R_{DS}(G, 0, 3t+2)$. In G'_1, to remove any vertex from the outer clique, we must first add a vertex from each inner clique, for a total of $t+1$ additions; since $k = 3t+2$ and $|D_s| = 2t+2$, this can only take place after G'_2 has been dominated using at most t vertices. In G'_2, a universal vertex u_i cannot be deleted until H_i has been dominated. If G can be dominated with t vertices, then it is possible to add the dominating set in H_0 and remove all the universal vertices, thus making the required capacity available. If not, then none of the universal vertices, say u_i, can be removed without first adding at least $t+1$ vertices to dominate H_i, for which there is not enough capacity. Therefore, there exists a reconfiguration sequence from D_s to some D'_s such that $D'_s \cap G'_2$ has t vertices if and only if G has a dominating set of size t. Moreover, the existence of a dominating set D of size t in G implies a path of length $6t+4$ from D_s to D_t; we add D in H_0, remove all universal vertices, reconfigure G'_1, add all universal vertices, and then remove D. Consequently, there exists a reconfiguration sequence from D_s to D_t in $6t+4$ steps if and only if G has a dominating set of size t. □

In the Hitting Set (HS) problem, given a finite universe $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$, and an integer k, the goal is to find a set $H \subseteq \mathcal{U}$ such that $|H| \leq k$ and for every set $F \in \mathcal{F}$ we have $F \cap U \neq \emptyset$. H is said to be *hitting set* of $\mathcal{F}$. When every set in $\mathcal{F}$

has unbounded cardinality (not bounded by some constant), the problem is known as Unbounded Hitting Set (UHS). Combining Lemmas 3.1.1 and 3.1.10 with the fact that Red-Blue Dominating Set is equivalent to Unbounded Hitting Set, we get the following:

Corollary 3.1.11. UHS-Reach *parameterized by k and* UHS-Bound *parameterized by $k + \ell$ are* `W[2]`*-hard.*

Unbounded Hitting Set is a classical problem which generalizes many "hitting-type" problems in graphs and so the hardness result is not very surprising. However, we shall see in subsequent chapters that if we consider Bounded Hitting Set (BHS) instead, more positive results become possible.

3.2 Structural Results

In this section, we shift our attention to structural properties of the reconfiguration graph $R_{DS}(G, 0, k)$.

Proposition 3.2.1. *Given a graph G and a dominating set D of G, a vertex $v \in V(G)$ is removable if and only if v has at least one neighbour in D and v has no private neighbour.*

Proposition 3.2.2. *For dominating sets A and B of graph G, if $A \subseteq B$, then $A \; {}_0\!\leftrightarrow_k B$ and $B \; {}_0\!\leftrightarrow_k A$.*

3.2.1 On the diameter of $R_{DS}(G, 0, k)$

We show that there exists an infinite family $\mathcal{F}_{bw}$ of graphs of bounded bandwidth such that for each graph G_n in the family there exists a value of k, namely the size of a minimum dominating set of G plus one, for which the corresponding reconfiguration graph has exponential diameter. We describe G_n in terms of several component subgraphs, each playing a role in forcing the reconfiguration of dominating sets.

A *linkage gadget* (part (a), Figure 3.3) consists of five vertices, the *external vertices* (or endpoints) e_1 and e_2, and the *internal vertices* i_1, i_2, and i_3. The external vertices are adjacent to each internal vertex as well as to each other; the following results from the internal vertices having degree two:

Proposition 3.2.3. *In a linkage gadget, the minimum dominating sets of size one are* $\{e_1\}$ *and* $\{e_2\}$*. Any dominating set containing an internal vertex must contain at least two vertices. Any dominating set in a graph containing* x *vertex-disjoint linkage gadgets with all internal vertices having degree exactly two must contain at least one vertex in each linkage gadget.*

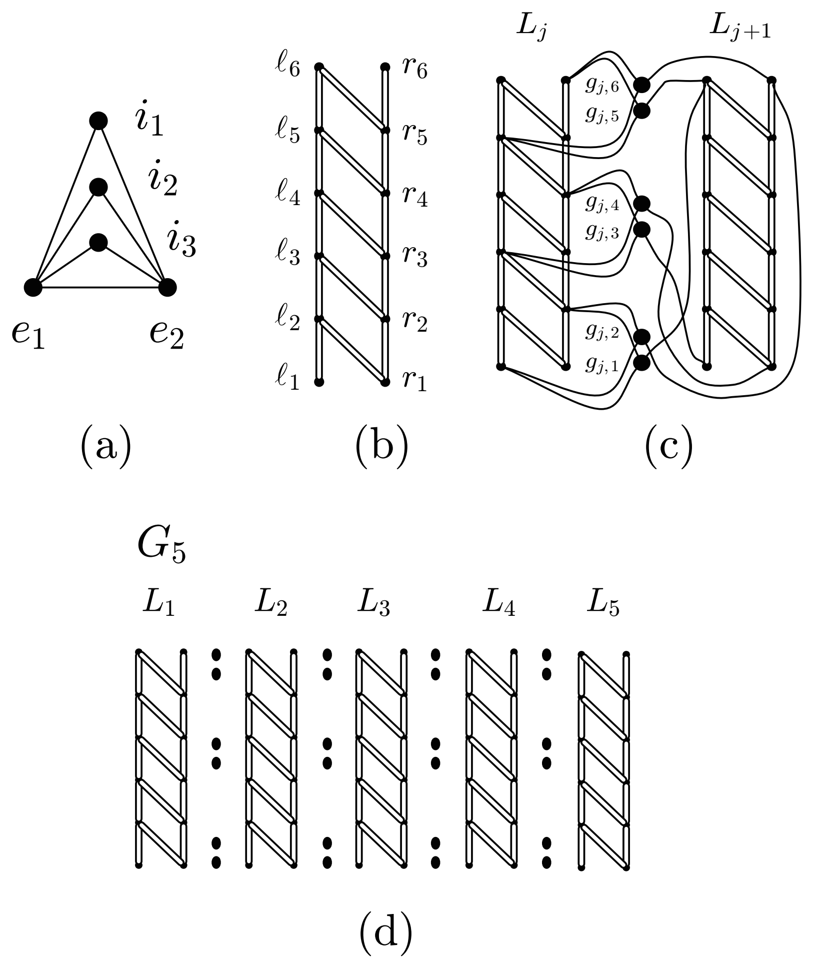

Figure 3.3: Parts of the construction.

A *ladder* (part (b) of Figure 3.3, linkage gadgets shown as double edges) is a graph consisting of twelve *ladder vertices* paired into six *rungs*, where rung i consists of the vertices ℓ_i and r_i for $1 \leq i \leq 6$, as well as the 45 internal vertices of fifteen linkage gadgets. Each linkage gadget is associated with a pair of ladder vertices, where the ladder vertices are the external vertices in the linkage gadget. The fifteen pairs are as follows: ten *vertical pairs* $\{\ell_i, \ell_{i+1}\}$ and $\{r_i, r_{i+1}\}$ for $1 \leq i \leq 5$, and five *cross pairs* $\{\ell_{i+1}, r_i\}$ for $1 \leq i \leq 5$. For convenience, we refer to vertices ℓ_i, $1 \leq i \leq 6$, and the associated linkage gadgets as the *left side of the ladder* and to vertices r_i, $1 \leq i \leq 6$, and the associated linkage gadgets as the *right side of the ladder*, or collectively as the *sides of the ladder*.

The graph G_n consists of n ladders L_1 through L_n and $n-1$ sets of *gluing vertices*, where each set consists of three *clusters* of two vertices each. For $\ell_{j,i}$ and $r_{j,i}$, $1 \leq i \leq 6$, the ladder vertices of ladder L_j, and $g_{j,1}$ through $g_{j,6}$ the gluing vertices that join ladders L_j and L_{j+1}, we have the following connections for $1 \leq j \leq n-1$:

- Edges connecting the *bottom cluster* to the bottom two rungs of ladder L_j and the top rung of ladder L_{j+1}: $\{\ell_{j,1}, g_{j,1}\}$, $\{\ell_{j,1}, g_{j,2}\}$, $\{r_{j,2}, g_{j,1}\}$, $\{r_{j,2}, g_{j,2}\}$, $\{\ell_{j+1,6}, g_{j,1}\}$, $\{r_{j+1,6}, g_{j,2}\}$.
- Edges connecting the *middle cluster* to the middle two rungs of ladder L_j and the bottom rung of ladder L_{j+1}: $\{\ell_{j,3}, g_{j,3}\}$, $\{\ell_{j,3}, g_{j,4}\}$, $\{r_{j,4}, g_{j,3}\}$, $\{r_{j,4}, g_{j,4}\}$, $\{\ell_{j+1,1}, g_{j,3}\}$, $\{r_{j+1,1}, g_{j,4}\}$.
- Edges connecting the *top cluster* to the top two rungs of ladder L_j and the top rung of ladder L_{j+1}: $\{\ell_{j,5}, g_{j,5}\}$, $\{\ell_{j,5}, g_{j,6}\}$, $\{r_{j,6}, g_{j,5}\}$, $\{r_{j,6}, g_{j,6}\}$, $\{\ell_{j+1,6}, g_{j,5}\}$, $\{r_{j+1,6}, g_{j,6}\}$.

Figure 3.3 parts (c) and (d) show details of the construction of G_n; they depict, respectively, two consecutive ladders and G_5, both with linkages represented as double edges. When clear from context, we sometimes use single subscripts instead of double subscripts to refer to the vertices of a single ladder.

We let $\mathcal{D} = \{\{\ell_{(j,2i-1)}, \ell_{(j,2i)}\}, \{r_{(j,2i-1)}, r_{(j,2i)}\} \mid 1 \leq i \leq 3, 1 \leq j \leq n\}$ denote a set of $6n$ pairs in G_n; the corresponding linkage gadgets are vertex-disjoint. Then Proposition 3.2.3 implies the following:

Proposition 3.2.4. *Any dominating set D of G_n must contain at least one vertex of each of the linkage gadgets for vertical pairs in the set $\mathcal{D}$ and hence is of size at least $6n$; if D contains an internal vertex, then $|D| > 6n$.*

Choosing an arbitrary external vertex for each vertical pair does not guarantee that all vertices on the side of a ladder are dominated; for example, the set $\{\ell_i \mid i \in \{1,4,5\}\}$ does not dominate the internal vertices in the vertical pair $\{\ell_2, \ell_3\}$. Choices that do not leave such gaps form the set $\mathcal{C} = \{\mathcal{C}_i \mid 1 \leq i \leq 4\}$ where $\mathcal{C}_1 = \{1,3,5\}$, $\mathcal{C}_2 = \{2,3,5\}$, $\mathcal{C}_3 = \{2,4,5\}$, and $\mathcal{C}_4 = \{2,4,6\}$.

Proposition 3.2.5. *In any dominating set D of size $6n$ and in any ladder L in G_n, the restriction of D to L must be of the form S_i for some $1 \leq i \leq 7$, as illustrated in Figure 3.4.*

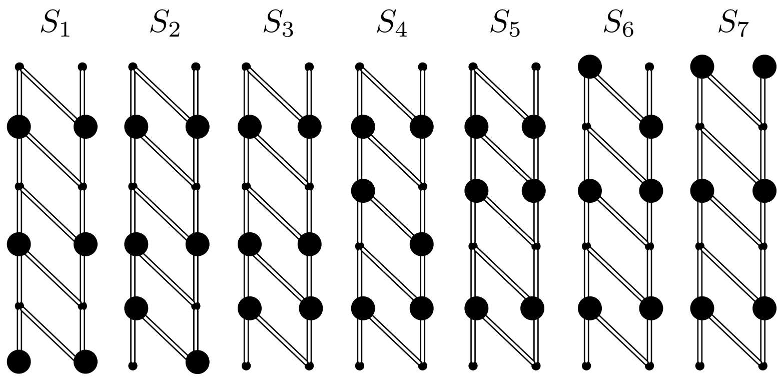

Figure 3.4: Minimum dominating sets for G_1.

Proof. Proposition 3.2.4 implies that the only choices for the left (right) vertices are $\{\ell_i \mid i \in \mathcal{C}_j\}$ ($\{r_i \mid i \in \mathcal{C}_j\}$) for $1 \leq j \leq 4$. The sets S_i, $1 \leq i \leq 7$, are the only combinations of these choices that dominate all the internal vertices in the cross pairs. □

We say that ladder L_j *is in state* D_i if the restriction of the dominating set to L_j is of the form S_i, for $1 \leq j \leq n$ and $1 \leq i \leq 7$.

The exponential lower bound on the length of reconfiguration sequences in Theorem 3.2.10 is based on counting how many times each ladder is modified from D_1 to D_7 or vice versa; we say ladder L_j undergoes a *switch* for each such modification. We first focus on a single ladder.

Proposition 3.2.6. *For D a dominating set of G_1, a vertex $v \in D$ is removable if and only if either*

- *v is the internal vertex of a linkage gadget one of whose external vertices is in D, or*
- *for every linkage gadget containing v as an external vertex, either the other external vertex is also in D or all internal vertices are in D.*

Lemma 3.2.7. *In $R_{DS}(G_1, 0, \gamma(G_1)+1)$ there is a single reconfiguration sequence between D_1 and D_7, of length* 12.

Proof. We define P to be the path in $R_{DS}(G_1, 0, \gamma(G_1) + 1)$ corresponding to the reconfiguration sequence from D_1 to D_7, i.e. $P = \langle D_1, D_1 \cup \{\ell_2\}, D_2, D_2 \cup \{r_2\}, D_3, D_3 \cup \{\ell_4\}, D_4, D_4 \cup \{r_4\}, D_5, D_5 \cup \{\ell_6\}, D_6, D_6 \cup \{r_6\}, D_7\rangle$ and demonstrate that there is no shorter path between D_1 and D_7.

By Propositions 3.2.5 and 3.2.4, G_1 has exactly seven dominating sets of size six, and any dominating set D of size seven contains two vertices from one vertical pair d in $\mathcal{D}$ and one from each of the remaining five. The neighbours of D in $R_{DS}(G_1, 0, \gamma(G_1)+1)$ are the vertices corresponding to the sets D_i, $1 \leq i \leq 7$, obtained by deleting a single vertex of D. The number of neighbours is thus at most two, depending on which, if any, vertices in d are removable.

If at least one of the vertices of D in d is an internal vertex, then at most one vertex satisfies the first condition in Proposition 3.2.6. Thus, for D to have two neighbours, there must be a ladder vertex that satisfies the second condition of Proposition 3.2.6, which by inspection of Figure 3.4 can be seen to be false.

If instead d contains two ladder vertices, in order for D to have two neighbours, the four ladder vertices on the side containing d must correspond to the union of two of the sets in $\mathcal{C}$. There are only three such unions, $\mathcal{C}_1 \cup \mathcal{C}_2$, $\mathcal{C}_2 \cup \mathcal{C}_3$, and $\mathcal{C}_3 \cup \mathcal{C}_4$, which implies that the only pairs with common neighbours are $\{D_i, D_{i+1}\}$ for $1 \leq i \leq 6$, as needed to complete the proof. □

For $n > 2$, we cannot reconfigure ladders independently from each other, as we need to ensure that all gluing vertices are dominated. For consecutive ladders L_j and L_{j+1}, any cluster that is not dominated by L_j must be dominated by L_{j+1}; the bottom, middle, and top clusters are not dominated by any vertex in S_2, S_4, and S_6, respectively.

Proposition 3.2.8. *In any dominating set D of G_n of size $6n$, for any $1 \leq j < n$,*

- *if L_j is in state D_2, then L_{j+1} is in state D_7;*
- *if L_j is in state D_4, then L_{j+1} is in state D_1; and*
- *if L_j is in state D_6, then L_{j+1} is in state D_7.*

Lemma 3.2.9. *For any reconfiguration sequence in $R_{DS}(G_n, 0, \gamma(G_n)+1)$ in which L_j and L_{j+1} are initially both in state D_1, if L_j undergoes p switches then L_{j+1} must undergo at least $2p + 1$ switches.*

Proof. We use a simple counting argument. When $p = 1$, the result follows immediately from Proposition 3.2.8 since L_j can only reach state D_7 if L_{j+1} is reconfigured from D_1 to D_7 to D_1 and finally back to D_7. After the first switch of L_j, both ladders are in state D_7.

For any subsequent switch of L_j, L_j starts in state D_7 because for L_j to reach D_1 from D_2 or to reach D_7 from D_6, by Proposition 3.2.8 L_{j+1} must have been in state D_7. Since by definition L_j starts in D_1 or D_7, to enable L_j to undergo a switch, L_{j+1} will have to undergo at least two switches, namely D_7 to D_1 and back to D_7. □

Theorem 3.2.10. *For D_s a dominating set of G_n such that every ladder of G_n is in state D_1 and D_t a dominating set of G_n such that every ladder of G_n is in state D_7, the length of any reconfiguration sequence between D_s and D_t in $R_{DS}(G_n, 0, \gamma(G_n)+1)$ is at least $12(2^{n+1} - n - 2)$.*

Proof. We first observe that Lemma 3.2.7 implies that the switch of any ladder requires at least twelve reconfiguration steps; since the vertex associated with a dominating set containing a gluing vertex will have degree at most one in $R_{DS}(G_n, 0, \gamma(G_n)+1)$, there are no shortcuts formed.

To reconfigure from D_s to D_t, ladder L_1 must undergo at least one switch. By Lemma 3.2.9, ladder L_2 will undergo at least $3 = 2^2 - 1$ switches, hence $2^j - 1$ switches for ladder L_j, $1 \leq j \leq n$. Since each switch requires twelve steps, the total number of steps is thus at least

$$12 \sum_{i=1}^{n} (2^i - 1) = 12(2^{n+1} - n - 2)$$

as claimed. □

Corollary 3.2.11. *There exists an infinite family $\mathcal{F}_{bw}$ of graphs of bounded bandwidth such that for each graph G_n in the family, the diameter of some connected component of $R_{DS}(G_n, 0, \gamma(G_n)+1)$ is in $\Omega(2^n)$.*

Proof. The bound on the diameter of $R_{DS}(G_n, 0, \gamma(G_n)+1)$ follows from Theorem 3.2.10. The fact that each graph in $\mathcal{F}_{bw}$ has bounded bandwidth follows from our construction; taking every two consecutive ladders along with their corresponding gluing vertices gives a bucket arrangement where each bucket has constant size. □

3.2.2 Graphs with a matching of size $\mu + 1$

Intuitively, and in contrast to Corollary 3.2.11, the diameter of $R_{DS}(G, 0, k)$ should decrease as the value of k increases; the diameter is in fact linear in $V(G)$ when $k = n$. But can we do any better? In other words, can we find a small constant c that guarantees either connectivity or bounds on the diameter of $R_{DS}(G, 0, k = \gamma(G)+c)$? Haas and Seyffarth [73] already tackled this question and showed that $R_{DS}(G, 0, k)$ is connected when $k = n - 1$ and G has at least two non-adjacent edges, or when $k \geq \Gamma(G_n) + 1$ and G is non-trivially bipartite or chordal. We push this boundary even further.

Theorem 3.2.12. *For any nonnegative integer μ, if G has a matching of size at least $\mu + 1$, then $R_{DS}(G, 0, n - \mu)$ is connected for $n = |V(G)|$. Moreover, the diameter of $R_{DS}(G, 0, n - \mu)$ is in $\mathcal{O}(n)$.*

Proof. For G a graph with matching $M = \{\{u_i, w_i\} \mid 0 \leq i \leq \mu\}$, we define $U = \{u_i \mid 0 \leq i \leq \mu\}$, $W = \{w_i \mid 0 \leq i \leq \mu\}$, and the set of *outsiders* $R = V(G) \setminus (U \cup W)$.

Using any dominating set D of G, we classify edges in M as follows: edge $\{u_i, w_i\}$, $0 \leq i \leq \mu$, is

- *clean* if neither u_i nor w_i is in D,
- *u-odd* if $u_i \in D$ but $w_i \notin D$,
- *w-odd* if $w_i \in D$ but $u_i \notin D$,
- *odd* if $\{u_i, w_i\}$ is u-odd or w-odd, and
- *even* if $\{u_i, w_i\} \subseteq D$.

We use clean(D) and odd(D), respectively, to denote the numbers of clean and odd edges for D. Similarly, we let u-odd(D) and w-odd(D) denote the numbers of u-odd and w-odd edges for D. In the example graph shown in Figure 3.5, $\mu + 1 = 7$ and $R = \emptyset$. There is a single clean edge, namely $\{u_1, w_1\}$, three w-odd edges $\{u_2, w_2\}$, $\{u_4, w_4\}$, and $\{u_6, w_6\}$, two u-odd edges $\{u_3, w_3\}$ and $\{u_5, w_5\}$, and a single even edge $\{u_0, w_0\}$.

It suffices to show that for D an arbitrary dominating set of G such that $|D| \leq n - \mu$, $D \leftrightarrow S$ for $S = V(G) \setminus W$; S is clearly a dominating set as each vertex $w_i \in W = V(G) \setminus S$ is dominated by u_i.

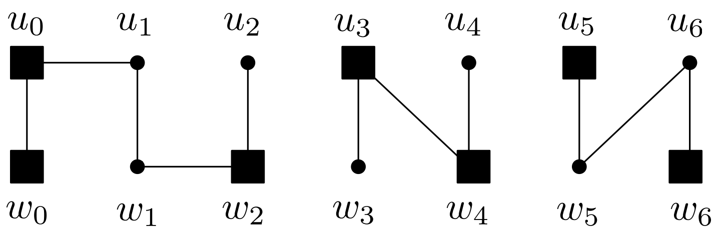

Figure 3.5: Vertices in D are marked with squares.

The reconfiguration sequence from D to S can be broken down into four stages. In the first stage, if $|D| < n - \mu$, we arbitrarily add vertices to D in order to obtain D', where $|D'| = n - \mu$. Clearly, D' is a dominating set of G since $D' \supseteq D$. By Proposition 3.2.2, since D' is a superset of D, then $D \leftrightarrow D'$. In the second stage, for a dominating set D_0 with no clean edges, we show $D' \leftrightarrow D_0$ by repeatedly decrementing the number of clean edges (u_i or w_i is added to the dominating set for some $0 \leq i \leq \mu$). In the third stage, for T_μ with μ u-odd edges and one even edge, we show $D_0 \leftrightarrow T_\mu$ by repeatedly incrementing the number of u-odd edges. Finally, we observe that deleting the single remaining element in $T_\mu \cap W$ yields $T_\mu \leftrightarrow S$. Putting all together, we obtain $D \leftrightarrow D' \leftrightarrow D_0 \leftrightarrow T_\mu \leftrightarrow S$, as needed.

In the second stage, for $x = \text{clean}(D')$, we show that $D' = D_x \leftrightarrow D_{x-1} \leftrightarrow D_{x-2} \leftrightarrow \ldots \leftrightarrow D_0$ where for each $0 \leq j \leq x$, D_j is a dominating set of G such that $|D_j| = n - \mu$ and $\text{clean}(D_j) = j$. To show that $D_a \leftrightarrow D_{a-1}$ for arbitrary $1 \leq a \leq x$, we prove that there is a removable vertex in some even edge and hence a vertex in a clean edge can be added in the next reconfiguration step. For $b = \text{odd}(D_a)$, the set E of vertices in even edges is of size $2((\mu + 1) - a - b)$.

Since each vertex in E has a neighbour in D_a, if at least one vertex in E does not have a private neighbour, then E contains a removable vertex (Proposition 3.2.1). The $n - (n - \mu) = \mu$ vertices in $V(G) \setminus D_a$ are the only possible candidates to be private neighbours. Of these, the b vertices of $V(G) \setminus D_a$ in odd edges cannot be private neighbours of vertices in E, as each is the neighbour of a vertex in $D_a \setminus E$ (the other endpoint of the edge). The number of remaining candidates, $\mu - b$, is smaller than the number of vertices in E. To see why, first we note that $\mu \geq 2a + b$ as the vertices of $V(G) \setminus D_a$ must contain both endpoints of any clean edge and one endpoint for any odd edge. Hence, we get:

$$\begin{aligned}|E| = 2((\mu+1)-a-b) &= 2\mu+2-2a-2b\\ &\geq (\mu+2a+b)+2-2a-2b\\ &= \mu-b+2\\ &> \mu-b\end{aligned}$$

Applying Proposition 3.2.1, we know that that there exists at least one removable vertex in E. When we remove such a vertex and add an arbitrary endpoint of a clean edge, the clean edge becomes an odd edge and the number of clean edges decreases. We can therefore reconfigure from D_a to the desired dominating set, and by applying the same argument a times, to D_0.

In the third stage we show that for $y = u\text{-odd}(D_0)$, $D_0 = T_y \leftrightarrow T_{y+1} \leftrightarrow T_{y+2} \leftrightarrow \ldots \leftrightarrow T_\mu$ where for each $y \leq j \leq \mu$, T_j is a dominating set of G such that $|T_j| = n - \mu$, $\text{clean}(T_j) = 0$, and $u\text{-odd}(T_j) = j$. To show that $T_c \leftrightarrow T_{c+1}$ for arbitrary $y \leq c \leq \mu - 1$, we use a counting argument to find a vertex in an even edge that is in W and removable; in one reconfiguration step the vertex is removed, increasing the number of u-odd edges, and in the next reconfiguration step an arbitrary vertex in R or in a w-odd edge is added to the dominating set. We let $d = w\text{-odd}(T_c)$ (i.e. the number of w-odd edges for T_c) and observe that since there are c u-odd edges, d w-odd edges, and no clean edges, there exist $(\mu+1)-c-d$ even edges. We define E_w to be the set of vertices in W that are in the even edges, and observe that each has a neighbour in T_c; a vertex in E_w will be removable if it does not have a private neighbour.

Of the μ vertices in $V(G)\backslash T_c$, only those in R are candidates to be private neighbours of vertices in E_w, as each vertex in an odd edge has a neighbour in T_c. As there are c u-odd edges and d w-odd edges, the total number of vertices in $R \cap V(G)\backslash T_c$ is $\mu - c - d$. Since this is smaller than the number of vertices in E_w, at least one vertex in E_w must be removable. When we remove such a vertex from T_c and in the next step add an arbitrary vertex from the outsiders or w-odd edges, the even edge becomes a u-odd edge and the number of u-odd edges increases. Note that we can always find such a vertex since there are $\mu - c - d$ outsiders, d w-odd edges, and $c \leq \mu - 1$. Hence, we can reconfigure from T_c to T_{c+1}, and by $\mu - c$ repetitions, to T_μ.

Using stages 1, 2, and 3, we can reconfigure from D to D' to T_μ. Finally, we reconfigure from T_μ to S by removing the sole vertex in $W \cap T_\mu$. Reconfiguring to D' can be achieved in at most $n - \mu$ steps and stages 1 and 2 require at most 2μ steps each, as $\mu \in \mathcal{O}(n)$ is at most the numbers of clean and u-odd edges. □

Theorem 3.2.13 shows, in some sense, that Theorem 3.2.12 is tight. The proof of Theorem 3.2.13 uses a result of Haas and Seyffarth [73, Lemma 3] which states that for any graph G with at least one edge, any dominating set of G of size $\Gamma(G)$ is an isolated node in $R_{DS}(G, 0, \Gamma(G))$ and therefore $R_{DS}(G, 0, \Gamma(G))$ is not connected.

Theorem 3.2.13. *For any nonnegative integer μ, there exists a graph G_μ with a matching of size μ such that $R_{DS}(G_\mu, 0, n - \mu)$ is not connected.*

Proof. Let G_μ be a path on $n = 2\mu$ vertices. Clearly, G_μ has μ disjoint edges, $n - \mu = 2\mu - \mu = \mu$, and $R_{DS}(G_\mu, 0, n - \mu) = R_{DS}(G_\mu, 0, \mu)$. We let D be a dominating set of G_μ such that $|D| \geq \mu + 1$. At least one vertex in D must have all its neighbors in D and is therefore removable. It follows that $\Gamma(G_\mu) = \mu$ and $R_{DS}(G_\mu, 0, \mu) = R_{DS}(G_\mu, 0, \Gamma(G_\mu))$, which is not connected by the result of Haas and Seyffarth [73, Lemma 3]. $\square$

3.2.3 $R_{DS}(G, 0, \Gamma(G) + 1)$ may not be connected

Haas and Seyffarth [73] left as an open question whether $R_{DS}(G, 0, \Gamma(G) + 1)$ is connected for any graph G, where $\Gamma(G)$ is the maximum cardinality of any minimal dominating set of G. In this section, we demonstrate that $R_{DS}(G, 0, \Gamma(G) + 1)$ is not connected for an infinite family of graphs $G_{(d,b)}$ for all positive integers $b \geq 3$ and $d \geq 2$, where graph $G_{(d,b)}$ is constructed from $d + 1$ cliques of size b. We demonstrate this fact using the graph $G_{(4,3)}$ as shown in part (a) of Figure 3.6, consisting of fifteen vertices partitioned into five cliques of size 3: the *outer clique* C_0, consisting of the top, left, and right *outer vertices* o_1, o_2, and o_3, and the four *inner cliques* C_1 through C_4, ordered from left to right. We use $v_{(i,1)}$, $v_{(i,2)}$, and $v_{(i,3)}$ to denote the top, left, and right vertices in clique C_i, $1 \leq i \leq 4$. More generally, a graph $G_{(d,b)}$ has $d + 1$ b-cliques C_i for $0 \leq i \leq d$. The clique C_0 consists of outer vertices o_j for $1 \leq j \leq b$, and for each inner clique C_i, $1 \leq i \leq d$ and each $1 \leq j \leq b$, there exists an edge $\{o_j, v_{(i,j)}\}$. The graph $G_{(d,b)}$ has some interesting properties which we already made use of in Lemma 3.1.10 to prove our `W`-hardness result.

For any $1 \leq j \leq b$, if a dominating set does not contain o_j, then the vertices $v_{(i,j)}$ of the inner cliques must be dominated by vertices in the inner cliques (hence Proposition 3.2.14). In addition, the outer vertex o_j can be dominated only by another outer vertex or some vertex $v_{(i,j)}$, $1 \leq i \leq d$ (hence Proposition 3.2.15).

Proposition 3.2.14. *Any dominating set that does not contain all of the outer vertices must contain at least one vertex from each of the inner cliques.*

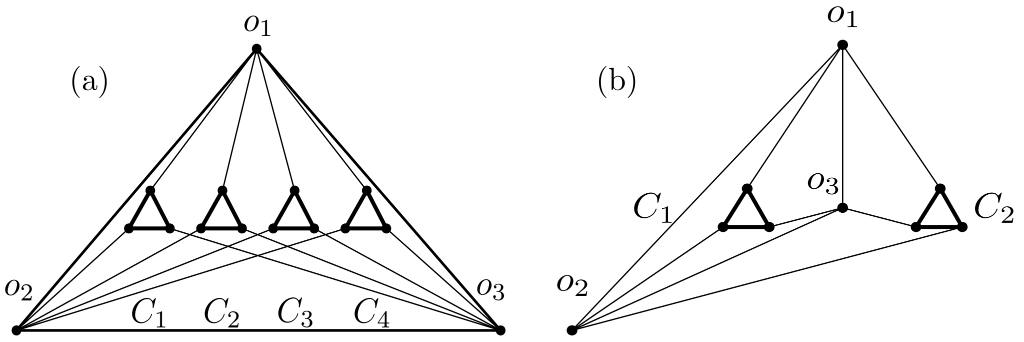

Figure 3.6: Counterexamples for (a) general and (b) planar graphs.

Proposition 3.2.15. *Any dominating set that does not contain any outer vertex must contain at least one vertex of the form $v_{(\cdot,j)}$ for each $1 \leq j \leq b$.*

Lemma 3.2.16. *For each graph $G_{(d,b)}$ as defined above, $\Gamma(G_{(d,b)}) = d + b - 2$.*

Proof. We first demonstrate that there is a minimal dominating set of size $d + b - 2$, consisting of $\{v_{(1,j)} \mid 2 \leq j \leq b\} \cup \{v_{(i,1)} \mid 2 \leq i \leq d\}$. The first set dominates $b - 1$ of the outer vertices and together the first inner clique and the second set dominate o_1 and the rest of the inner cliques. The dominating set is minimal, as the removal of any vertex $v_{(1,j)}$, $2 \leq j \leq b$, would leave vertex o_j with no neighbour in the dominating set and the removal of any $v_{(i,1)}$, $2 \leq i \leq d$, would leave $\{v_{(i,j)} \mid 1 \leq j \leq b\}$ with no neighbour in the dominating set.

By Proposition 3.2.14, any dominating set that does not contain all outer vertices must contain at least one vertex in each of the d inner cliques. Since the outer vertices form a minimal dominating set, any other minimal dominating set must contain at least one vertex from each of the inner cliques.

We now consider any dominating set D of size at least $d + b - 1$ containing one vertex for each inner clique and show that it is not minimal. If D contains at least one outer vertex, we can find a smaller dominating set by removing all but the outer vertex and one vertex for each inner clique, yielding a total of $d + 1 < d + b - 1$ vertices (since $b \geq 3$). Now suppose that D consists entirely of inner vertices. By Proposition 3.2.15, D contains at least one vertex of the form $v_{(\cdot,j)}$ for each $1 \leq j \leq b$. Moreover, for at least one value $1 \leq j' \leq b$, there exists more than one vertex of the form $v_{(\cdot,j')}$ as $d + b - 1 > b$. This allows us to choose b vertices of the form $v_{(\cdot,j)}$ for each $1 \leq j \leq b$ that dominate at least two inner cliques as well as all outer vertices. By selecting one member of D from each of the remaining $d-2$ inner cliques, we form a dominating set of size $d+b-2 < d+b-1$, proving that D is not minimal. $\square$

Theorem 3.2.17. *There exists an infinite family of graphs such that for each G in the family, $R_{DS}(G, 0, \Gamma(G)+1)$ is not connected.*

Proof. For any positive integers $b \geq 3$ and $d \geq 2$, we show that there is no path between dominating sets A to B in $R_{DS}(G_{(d,b)}, 0, d+b-1)$, where A consists of the vertices in the outer clique and B consists of $\{v_{(i,\ell)} \mid 1 \leq i \leq d, 1 \leq \ell \leq b, i \equiv \ell \pmod{b}\}$.

By Proposition 3.2.14, before we can remove any of the vertices in A, we need to add one vertex from each of the inner cliques, resulting in a dominating set of size $d+b = \Gamma(G_{(d,b)})+2$. As there is no such vertex in our graph, there is no way to connect A and B. $\square$

Each graph $G_{(d,b)}$ constructed for Theorem 3.2.17 is a b-partite graph; we can partition the vertices into b independent sets, where the jth set, $1 \leq j \leq b$, is defined as $\{v_{(i,j)} \mid 1 \leq i \leq d\} \cup \{o_i \mid 1 \leq i \leq d, i \equiv j+1 \pmod{b}\}$. Moreover, we can form a tree decomposition of width $2b-1$ of $G_{(d,b)}$, for all positive integers $b \geq 3$ and $d \geq b$, by creating bags with the vertices of the inner cliques and adding all outer vertices to each bag.

Corollary 3.2.18. *For every positive integer $b \geq 3$, there exists an infinite family of graphs of treewidth $2b-1$ such that for each G in the family, $R_{DS}(G, 0, \Gamma(G)+1)$ is not connected, and an infinite family of b-partite graphs such that for each G in the family, $R_{DS}(G, 0, \Gamma(G)+1)$ is not connected.*

Theorem 3.2.17 does not preclude the possibility that when restricted to planar graphs or any other graph class that excludes $G_{(d,b)}$, $R_{DS}(G, 0, \Gamma(G)+1)$ is connected. However, the next corollary follows directly from the fact that $G_{(2,3)}$ is planar (part (b) of Figure 3.6).

Corollary 3.2.19. *There exists a planar graph G for which $R_{DS}(G, 0, \Gamma(G)+1)$ is not connected.*

In answering Haas and Seyffarth's question, we have demonstrated infinite families of planar, bounded treewidth, and b-partite graphs for which the reconfiguration graph is not connected. Interestingly, all of our proofs break if we consider $R_{DS}(G, 0, \Gamma(G)+2)$. Hence, it remains to be seen whether $R_{DS}(G, 0, \Gamma(G)+2)$ is connected for all graphs.

3.3 Tractability

In the last section of this chapter, our goal is to design "efficient" algorithms for both DS-Reach and DS-Bound. We first tackle polynomial-time algorithms and show that DS-Reach is in fact solvable in polynomial time on paths and trees. However, and maybe surprisingly, determining whether DS-Bound is in P remains open, even for paths. We conclude the chapter with a fixed-parameter tractable algorithm for both DS-Reach and DS-Bound on graphs which exclude $K_{d,d}$ as a subgraph, a class of graphs which contains many well-known sparse graph classes such as graphs of bounded degree, bounded treewidth, and bounded degeneracy (Figure 2.2).

3.3.1 Polynomial-time algorithms

Given a path P, we will assume, without loss of generality, that $|V(P)| = x + 3n + y \geq 5$, where $x = 2$ and $y \in \{0, 1, 2\}$. Moreover, we assume that $V(P) = \{v_1, v_2, v_3, \ldots\}$ and that the vertices are ordered on the path based on their indices (from smallest to largest). We decompose P into subpaths $\{P^x, P_1, P_2, \ldots, P_n, P^y\}$, where $|V(P^x)| = 2$, $|V(P_i)| = 3$ for $1 \leq i \leq n$, and $|V(P^y)| = y$. In other words, P^x consists of the first two vertices in P, P_i consists of the i^{th} triplet (of vertices) in P, and P^y consists of the last y (possibly zero) vertices in P. A path with $2 + 3 \times 2 + 2$ vertices is shown in Figure 3.7. A decomposition of this path gives the set of subpaths $\{P^x, P_1, P_2, P^y\}$. We now construct a minimum dominating set D_* of P, which we call a *canonical* dominating set (red vertices in Figure 3.7). D_* consists of the last (highest index) vertex in P_x, the last vertex in each P_i for all $1 \leq i \leq n$, and the last vertex in P_y if $y = 2$ and no vertex in P_y otherwise.

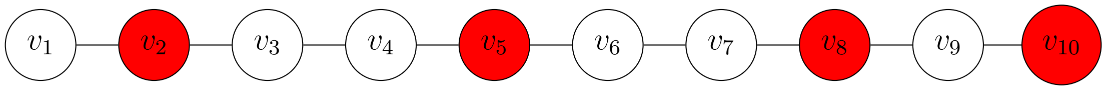

Figure 3.7: Example of a canonical dominating set (red vertices) of a path.

For an instance (P, D_s, D_t, k) of DS-Reach, we will show that either we have a trivial no-instance or $D_s \leftrightarrow D_*$ and $D_t \leftrightarrow D_*$. In other words, either we can easily conclude that there is no reconfiguration sequence from D_s to D_t in $R_{DS}(P, 0, k)$ or we can find a reconfiguration sequence from both D_s and D_t to the canonical dominating set, which implies $D_s \leftrightarrow D_t$. We will show that for any dominating set D of a path P, we can in time polynomial in $V(P)$ determine whether $D \leftrightarrow D_*$.

Proposition 3.3.1. *Given any path P such that $|V(P)| = x + 3 \times n + y \geq 5$, $x = 2$, and $y \in \{0, 1, 2\}$, the canonical dominating set D_* of P is a minimum dominating set, i.e. $\gamma(P) = |D_*|$.*

Proposition 3.3.2. *Given an instance (G, D_s, D_t, k) of* DS-Reach*, if any of the following conditions is true then we have a no-instance:*

(i) $D_s \neq D_t$ *and* $k = \gamma(G)$,

(ii) $D_s \neq D_t$, D_s *is minimal and* $k = |D_s|$, *or*

(iii) $D_s \neq D_t$, D_t *is minimal and* $k = |D_t|$.

Proof. Since $k = \gamma(G)$ in case (i), both D_s and D_t must be minimum dominating sets of G. Hence, no vertex in either dominating set is removable and both D_s and D_t are isolated nodes in $R_{DS}(G, 0, k)$. The same argument holds when D_s (D_t) is minimal and $k = |D_s|$ ($k = |D_t|$). □

Theorem 3.3.3. DS-Reach *is solvable in time polynomial in $|V(P)|$ for any path P. Moreover, the diameter of every connected component of $R_{DS}(P, 0, k)$ is polynomial in $|V(P)|$.*

Proof. Given an instance (P, D_s, D_t, k) of DS-Reach, we first construct the subpath decomposition and the canonical dominating set D_* of P. If D_s and D_t are not minimal dominating sets, we remove arbitrary vertices to obtain minimal dominating sets D'_s and D'_t. Then, we check whether any of the conditions from Proposition 3.3.2 hold, in which case we report a no-instance. Otherwise, we have a yes-instance.

To show that we have a yes-instance, we show that when none of the conditions from Proposition 3.3.2 hold, then $D'_s \leftrightarrow D_*$, $D'_t \leftrightarrow D_*$, and therefore $D_s \leftrightarrow D'_s \leftrightarrow D_* \leftrightarrow D'_t \leftrightarrow D_t$. We describe only the reconfiguration sequence from D'_s to D_*, as the same arguments hold for D'_t.

Our reconfiguration sequence will proceed in stages based on the subpath decomposition of P. We know that $V(P_x) \cap D_* = \{v_2\}$. Moreover, $|V(P_x) \cap D'_s| \geq 1$ (this statement in fact holds for any dominating set of P). We have the following cases to consider:

(1) $V(P_x) \cap D'_s = \{v_2\}$,

(2) $V(P_x) \cap D'_s = \{v_1\}$, or

(3) $V(P_x) \cap D'_s = \{v_1, v_2\}$.

In case (1), we ignore P_x and move on to the next subpath P_1. In case (3), we remove v_1, which remains dominated by v_2. In case (2), we know by Proposition 3.3.2 that $k \geq |D'_s|+1$ and we can therefore add v_2, then remove v_1. Let D''_s denote the new dominating set obtained after applying the above reconfiguration steps. We say D''_s and D_* agree on P_x.

We now consider the subpath P_1. We know that $V(P_1) \cap D_* = \{v_5\}$. Moreover, $|V(P_1) \cap D''_s| \geq 1$. The cases to consider are:

(1) $V(P_1) \cap D''_s = \{v_5\}$,

(2) $V(P_1) \cap D''_s = \{v_5, v_4\}$,

(3) $V(P_1) \cap D''_s = \{v_5, v_3\}$,

(4) $V(P_1) \cap D''_s = \{v_5, v_4, v_3\}$,

(5) $V(P_1) \cap D''_s = \{v_4\}$,

(6) $V(P_1) \cap D''_s = \{v_4, v_3\}$, or

(7) $V(P_1) \cap D''_s = \{v_3\}$.

In cases (1), (2), (3), and (4), we remove all vertices except v_5; as $v_2 \in D''_s$, all vertices remain dominated. Since $|D''_s| \leq |D'_s|$, we know (again from Proposition 3.3.2) that $k \geq |D''_s| + 1$ and we can therefore add v_5, then remove all other vertices in cases (5), (6), and (7). Repeating the same argument for every remaining subpath completes the proof. □

We can generalize the idea of canonical dominating sets to trees; we describe how to do so in what follows. Given a tree T, where $V(T) = \{v_1, \ldots v_n\}$, we pick any vertex in $V(T)$ and mark it as the root of T; we denote this vertex by v_{root}. Given a vertex $v \in V(T)$, we let parent(v) and children(v) denote the parent and the children of v, respectively, in the rooted tree. Moreover, we let depth(v) denote the length of the unique path from v to v_{root}. T_v denotes the subtree of T rooted at vertex v. We say two dominating sets D_1 and D_2 *agree* on T_v if $D_1 \cap V(T_v) = D_2 \cap V(T_v)$.

To build a canonical dominating set D_* of T (now a rooted tree), we first compute a minimum dominating set D of T (which can be accomplished in linear time using standard dynamic programming). We say a vertex $v \in D$ can be *pushed up* in T if and only if $\{D \setminus \{v\}\} \cup \text{parent}(v)$ is also a minimum dominating set of T. We then adjust this minimum dominating set using the following procedure: While D contains a vertex v that can be pushed up, we set $D = D \setminus \{v\} \cup \text{parent}(v)$.

We let D_* denote the canonical dominating set of T obtained after applying the above procedure, which can also be accomplished in time polynomial in $|V(T)|$. An example of a canonical dominating set is shown in Figure 3.8.

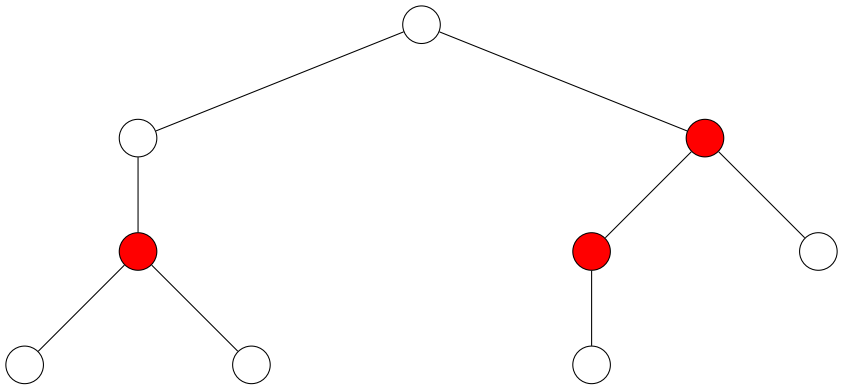

Figure 3.8: Example of a canonical dominating set (red vertices) of a tree.

Theorem 3.3.4. DS-Reach *is solvable in time polynomial in $|V(T)|$ for any tree T. Moreover, the diameter of every connected component of $R_{DS}(T, 0, k)$ is polynomial in $|V(T)|$.*

Proof. Given an instance (T, D_s, D_t, k) of DS-Reach, we first compute a minimum dominating set of D of T and then transform D into a canonical dominating set D_*, after fixing a root of T. If D_s and D_t are not minimal dominating sets, we remove arbitrary vertices to obtain minimal dominating sets D'_s and D'_t. Then, we check whether any of the conditions from Proposition 3.3.2 hold, in which case we report a no-instance. Otherwise, we show that we have a yes-instance.

Similarly to the case of paths, we will show that when none of the conditions from Proposition 3.3.2 hold, then $D'_s \leftrightarrow D_*$, $D'_t \leftrightarrow D_*$, and therefore $D_s \leftrightarrow D'_s \leftrightarrow D_* \leftrightarrow D'_t \leftrightarrow D_t$. We only describe the reconfiguration sequence from D'_s to D_*.

Intuitively, instead of dealing with subpaths, we will now apply reconfiguration steps on subtrees starting from the bottom of our rooted tree. Let v be a deepest vertex in T such that $v \in D_*$ and $v \notin D'_s$, i.e. there exists no other vertex $w \in T$ such that $w \in D_*$, $w \notin D'_s$, and $\text{depth}(w) > \text{depth}(v)$. By our choice of v, we know that $\{D_* \cap V(T_v)\} \setminus \{v\} \subseteq$

$\{D'_s \cap V(T_v)\} \setminus \{v\}$. In other words, we only have to add v to D'_s and remove some vertices (possibly none) from $D'_s \cap V(T_v)$ so that D_* and D'_s agree on T_v. We claim that the addition of v to D'_s will make some vertex in $D'_s \cap V(T_v) \setminus \{v\}$ removable, and we can therefore apply the same procedure repeatedly to reconfigure D'_s to D_*; every addition will be followed by a removal and therefore every dominating set will consist of at most $|D'_s| + 1$ vertices.

The fact that we can add v follows from Proposition 3.3.2, i.e. $k \geq |D'_s| + 1$. The fact that there exists a vertex $w \in D'_s \cap V(T_v) \setminus \{v\}$ that becomes removable after the addition of v follows from our construction of D_*; we prove the claim by contradiction. If we assume that w does not exist, it follows by our choice of v that $\{D_* \cap V(T_v)\} \setminus \{v\} = \{D'_s \cap V(T_v)\} \setminus \{v\}$, i.e. D_* and $D'_s \cup \{v\}$ agree on T_v. Moreover, as $v \notin D'_s$, every vertex in children(v) must either be in D_* or dominated by some vertex other than v (in D_*). But this implies that either v can be pushed up in D_* or D_* is not a minimum dominating set of T, a contradiction. □

3.3.2 Fixed-parameter tractable algorithms

The parameterized complexity of the Dominating Set problem (parameterized by k) on various classes of graphs has been studied extensively in the literature; the main goal has been to push the tractability frontier as far as possible. The problem was shown fixed-parameter tractable on planar graphs by Alber et al. [5], on bounded genus graphs by Ellis et al. [48], on H-minor-free graphs by Demaine et al. [39], on bounded expansion graphs by Nesetril and Ossona de Mendez [112], on nowhere-dense graphs by Dawar and Kreutzer [36], on degenerate graphs by Alon and Gutner [6], and finally on $K_{d,d}$-free graphs by Philip et al. [119] and Telle and Villanger [130]. Figure 2.2 illustrates the inclusion relationship among these classes of graphs, which all fall under the category of sparse graphs. Our fixed-parameter tractable algorithm relies on many of these earlier results. Interestingly, and since the class of $K_{d,d}$-free graphs includes all those other graph classes, our algorithm (Theorem 3.3.11) implies that the diameter of the reconfiguration graph $R_{DS}(G, 0, k)$ (or of its connected components), for G in any of the aforementioned classes, is bounded above by $f(k, c)$, where f is a computable function and c is constant which depends on the graph class at hand.

We give a single fixed-parameter tractable algorithm for both DS-Reach and DS-Bound parameterized by $k + d$ on graphs which exclude $K_{d,d}$ as a subgraph. We start with some definitions and known results.

Definition 3.3.5 ([119, 123, 130]). *Given a graph G, the* domination core *of G is a set $C \subseteq V(G)$ such that any set $D \subseteq V(G)$ is a dominating set of G if and only if D dominates*

C. In other words, D is a dominating set of G if and only if $C \subseteq N_G[D]$.

Theorem 3.3.6 ([119, 123, 130])**.** *If G is a graph which excludes $K_{d,d}$ as a subgraph and G has a dominating set of size at most k then the size of the domination core C of G is at most dk^d and C can be computed in $\mathcal{O}^*(dk^d)$ time.*

Definition 3.3.7. *A bipartite graph G with bipartition (A, B) is* B-twinless *if there are no vertices $u, v \in B$ such that $N(u) = N(v)$.*

Theorem 3.3.8 ([123])**.** *If G is a bipartite graph with bipartition (A, B) such that G is B-twinless and excludes $K_{d,d}$ as a subgraph then*

$$|B| \leq 2(d-1)(\frac{|A|e}{d})^{2d}.$$

Since Theorem 3.3.6 implies a bound on the size of the domination core and allows us to compute it efficiently, our main concern is to deal with vertices outside of the core, i.e. vertices in $V(G) \setminus C$. To that end, we introduce the notions of irrelevant and strongly irrelevant vertices for reconfiguration. Since these notions apply to problems other than Dominating Set and will be used in subsequent chapters, we give general definitions.

Definition 3.3.9. *For any vertex-subset problem $\mathcal{Q}$, n-vertex graph G, positive integers r_l and r_u, and $S_s, S_t \in V(R_{\mathcal{Q}}(G, r_l, r_u))$ such that there exists a reconfiguration sequence from S_s to S_t in $R_{\mathcal{Q}}(G, r_l, r_u)$, we say a vertex $v \in V(G)$ is* irrelevant *(with respect to S_s and S_t) if and only if $v \notin S_s \cup S_t$ and there exists a reconfiguration sequence from S_s to S_t in $R_{\mathcal{Q}}(G, r_l, r_u)$ which does not touch v. We say v is* strongly irrelevant *(with respect to S_s and S_t) if it is irrelevant and the length of a shortest reconfiguration sequence from S_s to S_t which does not touch v is no greater than the length of a shortest reconfiguration sequence which does (if the latter sequence exists).*

At a high level, it is enough to consider irrelevant vertices when solving DS-Reach, but strongly irrelevant vertices must be considered if we wish to solve DS-Bound. The next lemma shows that we can in fact find strongly irrelevant vertices outside of the domination core of a graph.

Lemma 3.3.10. *For G an n-vertex graph, C the domination core of G, and D_s and D_t two dominating sets of G, if there exist $u, v \in V(G) \setminus \{C \cup D_s \cup D_t\}$ such that $N_G(u) \cap C = N_G(v) \cap C$ then u (or v) is strongly irrelevant.*

Proof. Given a reconfiguration sequence $\sigma = \langle D_0 = D_s, D_1, \ldots, D_\ell = D_t \rangle$ from D_s to D_t which touches u, we will show how to obtain a reconfiguration sequence σ' such that $|\sigma'| \leq |\sigma|$ and σ' touches v but not u.

We construct σ' in two stages. In the first stage, we construct the sequence $\alpha = \langle D'_0, D'_1, \ldots, D'_\ell \rangle$ of dominating sets, where for all $0 \leq i \leq \ell$

$$D'_i = \begin{cases} D_i \cup \{v\} \setminus \{u\} \text{ if } u \in D_i \\ D_i \text{ if } u \notin D_i. \end{cases}$$

Note that α is not necessarily a reconfiguration sequence from D_s to D_t. In the second stage, we repeatedly delete from α any dominating set D'_i such that $D'_i = D'_{i+1}$, $0 \leq i < \ell$. We let $\sigma' = \langle D'_0, D'_1, \ldots, D'_{\ell'} \rangle$ denote the resulting sequence, in which there are no two consecutive sets that are equal, and we claim that σ' is in fact a reconfiguration sequence from D_s to D_t.

To prove the claim, we need to show that the following conditions hold:

(1) $D'_0 = D_s$ and $D'_{\ell'} = D_t$,

(2) D'_i is a dominating set of G for all $0 \leq i \leq \ell'$,

(3) $|D'_i \Delta D'_{i+1}| = 1$ for all $0 \leq i < \ell'$, and

(4) $|D'_i| \leq k$ for all $0 \leq i \leq \ell'$.

Since $u, v \notin D_s \cup D_t$, condition (1) clearly holds. Moreover, since replacing u by v in any set does not increase the size of the corresponding set, $|D'_i| \leq k$ (condition (4) holds) and $|D'_i \Delta D'_{i+1}| \leq 1$. As there are no two consecutive sets in σ' that are equal, $|D'_i \Delta D'_{i+1}| > 0$ and therefore $|D'_i \Delta D'_{i+1}| = 1$ (condition (3) holds). The fact that D'_i is a dominating set of G follows from the definition of a domination core. Since D_i is a dominating set of G, $C \subseteq N_G[D_i]$. Moreover, since $N_G(u) \cap C = N_G(v) \cap C$ and $u, v \notin C$, we know that $C \subseteq N_G[D'_i]$. By the definition of the domination core, it follows that D'_i (which still dominates C) is also a dominating set of G. Therefore, all four conditions hold, as needed. □

Theorem 3.3.11. DS-Reach *and* DS-Bound *parameterized by* $k+d$ *are fixed-parameter tractable for graphs that exclude* $K_{d,d}$ *as a subgraph.*

Proof. Given a graph G, integer k, and two dominating sets D_s and D_t of G of size at most k, we first compute the domination core C of G, which by Theorem 3.3.6 can be accomplished in $\mathcal{O}^*(dk^d)$ time. Next, and due to Lemma 3.3.10, we can delete strongly irrelevant vertices from $V(G) \setminus \{C \cup D_s \cup D_t\}$, i.e. as long as there exist $u, v \in V(G) \setminus \{C \cup D_s \cup D_t\}$ such that $N_G(u) \cap C = N_G(v) \cap C$, we delete either u or v. We denote this new graph by G'.

Now consider the bipartite graph G'' with bipartition $(A = C \setminus \{D_s \cup D_t\}, B = V(G') \setminus \{C \cup D_s \cup D_t\})$, i.e. we ignore all edges between vertices in $C \setminus \{D_s \cup D_t\}$ and all edges between vertices in $V(G') \setminus \{C \cup D_s \cup D_t\}$. This graph is B-twinless, since for every pair of vertices $u, v \in V(G) \setminus \{C \cup D_s \cup D_t\}$ such that $N_G(u) \cap C = N_G(v) \cap C$ either u or v is strongly irrelevant and is therefore not in $V(G')$ nor $V(G'')$. Moreover, since every subgraph of a $K_{d,d}$-free graph is also $K_{d,d}$-free, G'' is $K_{d,d}$-free. Hence, by Theorems 3.3.6 and 3.3.8, we have

$$\begin{aligned} |B| &\leq 2(d-1)(\frac{|A|e}{d})^{2d} \\ &\leq 2d(3|A|)^{2d} \leq 2d(3dk^d)^{2d}. \end{aligned}$$

Putting it all together, we know that after deleting strongly irrelevant vertices, the number of vertices in the resulting graph G' is at most

$$\begin{aligned} |V(G')| &= |V(C)| + |D_s \cup D_t| + |V(G') \setminus \{C \cup D_s \cup D_t\}| \\ &\leq dk^d + 2k + 2d(3dk^d)^{2d} \end{aligned}$$

Hence, we can solve DS-Reach and DS-Bound by exhaustively enumerating all $2^{|V(G')|}$ subsets of $V(G')$ and building the reconfiguration graph $R_{DS}(G', 0, k)$. $\square$

Corollary 3.3.12. *If G excludes $K_{d,d}$ as a subgraph then the diameter of each connected component of $R_{DS}(G, 0, k)$ is at most $f(k, d)$, for some computable function f.*

Theorem 3.3.11 implies the existence of an algorithm which solves both the DS-Reach and DS-Bound problems in $\mathcal{O}(f(k, d)n^{\mathcal{O}(1)})$ time on graphs excluding $K_{d,d}$ as a subgraph, for some computable function f. The reason why both problems turned out to be solvable within the same asymptotic running time is because the most computationally expensive section of the algorithm is the construction of a full reconfiguration graph of an auxiliary graph G', where the number of vertices in G' is also bounded by some function of k and d. Once this reconfiguration graph has been constructed, finding a path or a shortest path between any two nodes can be accomplished in time polynomial in the size of the reconfiguration graph. Hence, to improve the running time of the presented algorithm this construction has to be avoided. On the other hand, another problem which we have not considered here is DS-Bound with parameter $\ell + d$. In other words, can we solve the problem in $\mathcal{O}(g(\ell, d)(nk)^{\mathcal{O}(1)}$ time, for some computable function g? We consider this question, with a slightly different parameter, in Chapter 5. For now we only note that none of the techniques presented above apply in this case, as computing the domination core crucially depends on the parameter k.

Chapter 4

Vertex cover reconfiguration and beyond

In Chapter 3, we considered the reachability and bounded reachability variants of the DOMINATING SET problem, which as we have seen in Corollary 3.1.11 can be equivalently formulated as the UNBOUNDED HITTING SET problem. We now turn our attention to a special case of the BOUNDED HITTING SET problem, namely 2-HITTING SET or VERTEX COVER. Formally, given an instance (G, k) of VERTEX COVER, we can construct the corresponding 2-HITTING SET instance $(\mathcal{U}, \mathcal{F}, k)$, where $\mathcal{U} = \{e_u \mid u \in V(G)\}$ and $\mathcal{F} = \{\{e_u, e_v\} \mid uv \in E(G)\}$. Every set in $\mathcal{F}$ has size exactly two. We say an element $e_u \in \mathcal{U}$ *hits* a set $F \in \mathcal{F}$ if $e_u \in F$. Hence, the goal is to *hit* every set in $\mathcal{F}$ using at most k elements from $\mathcal{U}$.

Under classical complexity assumptions, VC-REACH and VC-BOUND are, in some sense, very similar to DS-REACH and DS-BOUND when viewed from the hard side of the spectrum. That is, all four problems are PSPACE-complete on planar graphs of bounded degree and graphs of bounded bandwidth; it is an easy exercise to show the latter result using a reduction from the $\mathcal{H}$-WORD REACH problem, as we did in Lemma 3.1.6. However, the tractable side seems more generous for VC-REACH and VC-BOUND and the fixed-parameter tractable side even more so.

We start by introducing the notions of edit sequences and nice reconfiguration sequences in Section 4.1, and show that any reconfiguration sequence between two vertex covers of a graph can be converted into a nice one. Intuitively, a nice reconfiguration sequence can be split into smaller "pieces" where the added/removed vertices in the first and last piece induce independent sets in the input graph G and the added/removed vertices in all other pieces induce bicliques in G. For graphs of degree at most d, each biclique has at most d vertices on each side. The structure of nice reconfiguration sequences will help us prove some of the results in the remainder of the chapter.

We present lower bounds in Sections 4.2, 4.3, and 4.4. The W[2]-hardness proof of DS-Reach and DS-Bound in Lemma 3.1.10 implied that solving either problem is as hard as solving the DS problem itself. We show that a similar relationship holds for a large number of graph problems defined using hereditary graph properties. That is, our first lower bound, which we present in Section 4.2, is a meta-theorem which relates the parameterized complexity of Π-Sub to the parameterized complexity of Π-Del-Bound, Π-Sub-Reach, and Π-Sub-Bound. In particular, we show that for hereditary graph properties satisfying some structural conditions, Π-Del-Bound parameterized by ℓ, Π-Sub-Reach parameterized by k, and Π-Sub-Bound parameterized by $k + \ell$ are at least as hard as Π-Del parameterized by k. If we consider Π as the collection of all edgeless graphs, then our meta-theorem implies that VC-Bound parameterized by ℓ, IS-Reach parameterized by k, and IS-Bound parameterized by $k + \ell$ are at least as hard as IS parameterized by k, which is W[1]-hard. An interesting but not surprising aspect of this meta-theorem is that it shows, more or less, that reconfiguration problems parameterized by ℓ, or equivalently finding reconfiguration sequences of length at most ℓ in $\mathcal{O}(f(\ell)n^{\mathcal{O}(1)})$ time, is "almost always" hard.

In Section 4.3 we prove another general theorem, this time showing that both Π-Del-Reach and Π-Del-Bound are at least as hard as Π-Comp (Section 2.1). In the context of Vertex Cover, an equivalent statement of the theorem states that given a graph G and a vertex cover S of G of size k, it is possible to construct an instance of VC-Reach which is a yes-instance if and only if G has a vertex cover of size $k - 1$. That is, we can do "compression via reconfiguration". We illustrate the usefulness of this result by providing a general framework to prove NP-hardness of VC-Reach and VC-Bound on restricted graph classes. As an example, we show that both problems remain NP-hard on 4-regular graphs. To complete the hardness results, we show in Section 4.4 that VC-Bound parameterized by ℓ remains W[1]-hard on bipartite graphs. There are several reasons for considering the parameterized complexity of the problem on bipartite graphs. On the one hand, we know from Proposition 2.4.8 that the problem becomes fixed-parameter tractable whenever $\ell = |S_{s\setminus t} \cup S_{t\setminus s}|$. When $n = |S_{s\setminus t} \cup S_{t\setminus s}|$, we know from Proposition 2.4.3 that $\ell \geq n$, since every vertex in $S_{s\setminus t} \cup S_{t\setminus s}$ must be touched at least once. Therefore, even though the problem is fixed-parameter tractable, any algorithm solving an instance of the VC-Bound problem where $|S_{s\setminus t} \cup S_{t\setminus s}| = \ell = n$ would run in time exponential in n. Moreover, Proposition 2.4.7 implies that whenever $|S_{s\setminus t} \cup S_{t\setminus s}| = n$ the input graph must be bipartite. It is thus natural to ask about the complexity of the problem when

$|S_{s \setminus t} \cup S_{t \setminus s}| = \ell < n$ and the input graph is restricted to be bipartite. On the other hand, since the Vertex Cover problem is known to be solvable in time polynomial in n on bipartite graphs, this result is, to the best of our knowledge, the first example of a problem solvable in polynomial time whose bounded reachability variant is `W[1]`-hard.

Section 4.5 is dedicated to our positive results. As the reader might have noticed, none of the hardness results mentioned above consider VC-Reach or VC-Bound parameterized by k. In fact, both problems are fixed-parameter tractable and even admit a simple $\mathcal{O}(k^2)$ kernel in that case. Having established the existence of an algorithm solving both problems in $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ time on general graphs, for some computable function f, we then focus on the VC-Bound problem parameterized by ℓ in an attempt to find some graph classes on which the problem becomes fixed-parameter tractable. We show that this is indeed possible for graphs of bounded degree, graphs of bounded treewidth, and planar graphs. Finally, on the polynomial side, we show that VC-Reach and VC-Bound are both in `P` for even-hole-free and cactus graphs. All of our positive results exhibit interesting relationships with either the size or the structure of the graph induced by the symmetric difference of two vertex covers of a graph. In particular, for our polynomial-time algorithms, it turns out that the structure is rather simple and the number of vertices we need to touch outside of the symmetric difference is bounded by a very small constant c; $c \leq 2$ for even-hole-free graphs and $c \leq 4$ for cactus graphs. For fixed-parameter tractable algorithms on graph class $\mathcal{C}$, with ℓ as the parameter, we already have a bound on the size of the symmetric difference; it is at most ℓ. Hence, the main technical challenge in this case is to show that (i) the number of vertices to consider outside of the symmetric difference is bounded by some function of ℓ and (ii) we can find those vertices "efficiently".

4.1 Nice reconfiguration sequences

There are multiple ways of representing a reconfiguration sequence σ between two vertex covers S_s and S_t of a graph G. So far, we have been representing σ as a sequence of sets or vertex covers, i.e. $\sigma = \langle S_0, S_1, \ldots, S_\ell \rangle$. Alternatively, we can view σ as a sequence of added/removed vertices, which brings us to the notion of edit sequences. For any sequence σ, we let $\sigma[p]$, $1 \leq p \leq |\sigma|$, denote the element at position p in σ. Moreover, we use the notation $\sigma[p_1, p_2]$, $1 \leq p_1, p_2 \leq |\sigma|$, to denote the subsequence (with no gaps) starting at position p_1 and ending at position p_2.

Given a graph G, we assume all vertices of G are labeled from 1 to n, i.e. $V(G) = \{v_1, v_2, \ldots, v_n\}$. We let $\{v_1, \ldots, v_n\}$, $\{\varnothing\}$, $\{+\}$, $\{-\}$, and $\{+, -\}$ (or equivalently $\{+1, -1\}$) denote the set of *vertex markers*, the *blank marker*, the *addition marker*, the *removal*

+	−	−	+	−	−	+	+	−	−	−	+	−	+	+
1	15	3	∅	5	10	9	8	7	6	11	∅	13	14	2

Figure 4.1: Example of an edit sequence.

marker, and the set of *sign markers*, respectively. An *edit sequence* σ is a pair of ordered sequences (sign-σ, vertex-σ) of the same length, where sign-$\sigma \subseteq \{+, -\}^\ell$ and vertex-$\sigma \subseteq \{v_1, \ldots, v_n, \varnothing\}^\ell$. Alternatively, an edit sequence can be viewed as a $2 \times \ell$ matrix, where the first row corresponds to the *sign-sequence* sign-σ and the second row corresponds to the *vertex-sequence* vertex-σ. An example is given in Figure 4.1.

For any edit sequence σ, we say σ is *unlabeled* if vertex-σ only contains the element $\varnothing$, σ is *partial* if vertex-σ contains at least one element from each of the sets $\{v_1, \ldots, v_n\}$ and $\{\varnothing\}$, and σ is *labeled* if vertex-σ does not contain the element $\varnothing$. The *length* of edit sequence σ is denoted by $|\sigma|$, where $|\sigma| = |\text{sign-}\sigma| = |\text{vertex-}\sigma|$. Proposition 4.1.1 will be particularly useful when designing fixed-parameter tractable algorithms with the length of a reconfiguration sequence as the parameter.

Proposition 4.1.1. *The total number of possible unlabeled edit sequences of length at most ℓ is $\sum_{i=1}^{\ell} 2^i < 2^{\ell+1}$.*

By a slight abuse of notation, we let $\sigma[p] = \text{vertex-}\sigma[p]^{\text{sign-}\sigma[p]} \in \{v_i^+ \mid 1 \leq i \leq n\} \cup \{v_i^- \mid 1 \leq i \leq n\} \cup \{\varnothing^+, \varnothing^-\}$, $1 \leq p \leq |\sigma|$, denote the *marker* at position p in σ. That is, when vertex-$\sigma[p] \in \{v_1, \ldots, v_n\}$, $\sigma[p]$ either corresponds to a *vertex addition marker* or a *vertex removal marker* depending on whether sign-$\sigma[p] \in \{+\}$ or sign-$\sigma[p] \in \{-\}$, respectively. When vertex-$\sigma[p] \in \{\varnothing\}$, $\sigma[p] \in \{\varnothing^+\}$ ($\sigma[p] \in \{\varnothing^-\}$) is a *blank addition (removal) marker.*

Given an edit sequence σ = (sign-σ, vertex-σ) and two positive integers $1 \leq p_1 < p_2 \leq |\sigma|$, we say β = (sign-$\sigma[p_1, p_2]$, vertex-$\sigma[p_1, p_2]$) is a *segment* of σ. Two segments β and β' of σ are *consecutive* whenever β' occurs later than β in σ and there are no gaps between β and β'. A *piece* is a group of zero or more consecutive segments. For any pair of consecutive segments β and β' in σ, we say β' (β) is the *successor* (*predecessor*) of β (β'). Given an edit sequence σ, a segment β of σ is an *add-remove segment* if sign-β contains addition markers followed by removal markers and no removal marker is followed by an addition marker. We say β is a *d-add-remove segment*, $d > 0$, if β is an add-remove segment and sign-β contains 1 to d addition markers followed by 1 to d removal markers.

Definition 4.1.2. *Given a positive integer $d > 0$, an edit sequence σ is d-*well-formed *if it is subdivided into three consecutive pieces σ_s, σ_c, and σ_e, such that:*

$-$	$-$	$+$	$+$	$-$	$-$	$+$	$+$	$-$	$-$	$+$	$+$	$-$	$+$	$+$
1	15	3	$\varnothing$	5	10	9	8	7	6	11	$\varnothing$	13	14	2

Figure 4.2: Example of a 2-well-formed edit sequence.

(1) In the starting piece, *σ_s, sign-σ_s consists of zero or more removal markers,*

(2) the central piece, *σ_c, consists of zero or more d-add-remove segments, and*

(3) in the ending piece, *σ_e, sign-σ_e consists of zero or more addition markers.*

An example of a 2-well-formed edit sequence σ is given in Figure 4.2, where the starting piece of σ consists of two vertex removal markers, the central piece consists of three 2-add-remove segments, and the ending piece consists of two vertex addition markers.

We introduce some useful operations on edit sequences. The *concatenation* of two edit sequences σ_1 and σ_2, denoted by $concat(\sigma_1, \sigma_2)$, results in an edit sequence σ_3. The length of σ_3 is $|\sigma_1| + |\sigma_2|$, $\sigma_3[p] = \sigma_1[p]$ for all $1 \leq p \leq |\sigma_1|$, and $\sigma_3[p] = \sigma_2[p - |\sigma_1|]$ for all $|\sigma_1| + 1 \leq p \leq |\sigma_1| + |\sigma_2|$. Given an edit sequence σ and two positive integers p_1 and p_2, $1 \leq p_1, p_2 \leq |\sigma|$, the *cut* operation, denoted by $cut(\sigma, p_1, p_2)$, results in a new edit sequence $\sigma' = concat(\sigma[1, p_1 - 1], \sigma[p_2 + 1, |\sigma|])$. We write $cut(\sigma, p_1)$ whenever $p_1 = p_2$. The *clean* operation, denoted by $clean(\sigma)$, results in a new edit sequence σ' which is obtained by repeatedly applying $cut(\sigma, p)$ as long as $\sigma[p] \in \{\varnothing^+, \varnothing^-\}$. Given a partial edit sequence σ_p and a labeled edit sequence σ_f, the *merging* operation consists of replacing the pth blank marker in vertex-σ_p with the pth vertex marker in vertex-σ_f. We say σ_f is a *filling edit sequence* of σ_p if $merge(\sigma_p, \sigma_f)$ produces σ, where σ_p is partial and σ and σ_f are not. Hence, $|clean(\sigma_p)| + |\sigma_f|$ must be equal to $|\sigma|$. Finally, given $t \geq 2$ labeled edit sequences σ_1, ..., σ_t, the *mixing* of those sequences, $mix(\sigma_1, \ldots, \sigma_t)$, produces the set of all labeled edit sequences of length $|\sigma_1| + \ldots + |\sigma_t|$. Each $\sigma \in mix(\sigma_1, \ldots, \sigma_t)$ consists of all markers in each σ_i, $1 \leq i \leq t$. However, the respective orderings of markers from each σ_i is maintained, i.e. if we cut from σ the markers of all sequences except σ_1, we get σ_1.

We now discuss how edit sequences relate to reconfiguration sequences. Given a graph G and an edit sequence σ, we use $V(\sigma)$ to denote the set of vertices touched in σ, i.e. $V(\sigma) = \{v_i \mid v_i \in \text{vertex-}\sigma\}$. Similarly, we let $V^+(\sigma)$ and $V^-(\sigma)$ denote the (not necessarily disjoint) sets of added and removed vertices in σ, respectively. Formally, $V^+(\sigma) = \{v_i \mid v_i^+ \in \sigma\}$ and $V^-(\sigma) = \{v_i \mid v_i^- \in \sigma\}$.

When σ is labeled, we let $V(S, \sigma)$ denote the set of vertices obtained after executing all reconfiguration steps in σ on G starting from some vertex cover S of G. We say σ is *valid* whenever every set $V(S, \sigma[1, p])$, $1 \leq p \leq |\sigma|$, is a vertex cover of G, $S \neq V(S, \sigma[1])$, and $V(S, \sigma[1, q-1]) \neq V(S, \sigma[1, q])$, $1 < q \leq |\sigma|$; we say σ is *invalid* otherwise. Note that even if $|S| \leq k$, σ is not necessarily a walk in $R_{VC}(G, 0, k)$, as σ might violate the maximum allowed capacity constraint k. Hence, we let $cap(\sigma) = max_{1 \leq p \leq |\sigma|}(|V(S, \sigma[1, p])|)$ and we say σ is *tight* whenever it is valid and $cap(\sigma) \leq k$. For σ a partial or unlabeled edit sequence, we can still compute $cap(\sigma)$ using only sign-σ, i.e. $cap(\sigma) = max_{1 \leq p \leq |\sigma|}(|S| + \sum_{i=1}^{|p|} \text{sign-}\sigma[i])$. A partial edit sequence σ is valid or tight if $clean(\sigma)$ is valid or tight.

Proposition 4.1.3. *Given a graph G and two vertex covers S_s and S_t of G, an edit sequence σ is a reconfiguration sequence from S_s to S_t if and only if σ is a tight edit sequence from S_s to S_t.*

Given a graph G, a vertex cover S of G, an unlabeled edit sequence σ, and an ordered sequence $L = \langle l_1, \ldots, l_{|\sigma|} \rangle$ of (not necessarily distinct) vertex labels in $\{v_1, \ldots, v_n\}$, the *label* operation, denoted by $label(\sigma, L)$, returns a labeled edit sequence $\sigma' = (\text{sign-}\sigma, L)$. We say an unlabeled edit sequence σ can be *applied* to G and S if there exists an L such that $label(\sigma, L)$ is valid starting from S.

Definition 4.1.4. *Given a graph G, a vertex cover S of G, and $t \geq 2$ valid labeled edit sequences σ_1, …, σ_t (starting from S), we say σ_1, …, σ_t are* compatible *if every $\sigma \in mix(\sigma_1, \ldots, \sigma_t)$ is a valid edit sequence starting from S. They are* incompatible *otherwise.*

Recall that vertices s and t (vertex sets S and T) are said to be separated if $s \neq t$ ($S \cap T = \emptyset$) and there is no edge $st \in E(G)$ (no edge $st \in E(G)$ for $s \in S$ and $t \in T$).

Proposition 4.1.5. *Given a graph G, a vertex cover S of G, and $t \geq 2$ valid labeled edit sequences σ_1, …, σ_t (starting from S), if $V(\sigma_i)$ and $V(\sigma_j)$ are separated, for all $1 \leq i, j \leq t$ and $i \neq j$, then σ_1, …, σ_t must be compatible.*

Proof. If σ_1, …, σ_t are not compatible, then there exists a $\sigma \in mix(\sigma_1, \ldots, \sigma_t)$ which is not valid starting from S. That is, for some p, $1 \leq p \leq |\sigma|$, $V(S, \sigma[1, p])$ is not a vertex cover of G. But since $V(\sigma_i)$ and $V(\sigma_j)$ are separated, for all $1 \leq i, j \leq t$ and $i \neq j$, $V(\sigma_i)$ and $V(\sigma_j)$ are vertex disjoint and there exists no edge $uv \in E(G)$ such that $u \in V(\sigma_i)$ and $v \in V(\sigma_j)$. Therefore, if $V(S, \sigma[1, p])$ is not a vertex cover of G then one of σ_1, …, σ_t is not valid, a contradiction. □

Edit sequences have certain special properties, which we capture using the notion of nice edit sequences. Moreover, we present the ConvertToNice algorithm (Algorithm 1) which transforms any valid edit sequence σ from S_s to S_t into a nice edit sequence σ' from S_s to S_t, such that $|V(S_s, \sigma'[1,p])| \leq |V(S_s, \sigma[1,p])|$ for all $1 \leq p \leq |\sigma|$. Hence, if σ is tight then so is σ'.

Definition 4.1.6. *Given a graph G, a vertex cover S of G, and a valid edit sequence σ starting from S, we say σ is a* nice edit sequence *if it is valid, $\Delta(G)$-well-formed, and satisfies the following invariants:*

(i) **Connectivity invariant:** *For $\mathcal{S}(\sigma) = \{\sigma_1, \ldots, \sigma_t\}$ the ordered set of $\Delta(G)$-add-remove segments in σ, $G[V(\sigma_i)]$ is connected for all i between 1 and t. Moreover, $G[V(\sigma_i)]$ is a biclique where the added vertices in σ_i belong to one partition and the removed vertices to the other.*

(ii) **Early removal invariant:** *For $1 \leq p_1 < p_2 < p_3 \leq |\sigma|$, if $\sigma[p_1] \in \{v_1^+, \ldots, v_n^+\}$, $\sigma[p_2] \in \{v_1^+, \ldots, v_n^+\}$, and $\sigma[p_3] \in \{v_1^-, \ldots, v_n^-\}$, then $V(\sigma[p_3])$ and $V(\sigma[p_1+1, p_3-1])$ are not separated.*

Intuitively, the early removal invariant states that every vertex removal marker in a nice edit sequence must occur "as early as possible". In other words, a vertex is removed right after its neighbors (and possibly itself) are added. The next definition and propositions will be useful in proving the correctness of the ConvertToNice algorithm.

Definition 4.1.7. *Given a graph G, a vertex cover S of G, and a valid edit sequence σ starting from S, we say a vertex removal marker at position $p > 1$ in σ satisfies the* non-separation condition *if there exists a position q between 1 and $p - 1$ such that $\sigma[q] \in \{v_1^+, \ldots, v_n^+\}$ and $\sigma[p]$ cannot occur before $\sigma[q]$.*

Proposition 4.1.8. *Given a graph G, a vertex cover S of G, and a valid edit sequence σ starting from S, if some vertex removal marker at position $p > 1$ in σ violates the non-separation condition, then either all markers in $\sigma[1,p]$ are vertex removal markers or σ is not nice. Moreover, if $\sigma[p]$ violates the non-separation condition then $concat(\sigma[p], cut(\sigma, p))$ remains a valid edit sequence starting from S.*

Proof. We let $\sigma[p] = v_i^-$, $1 \leq i \leq n$, denote a vertex removal marker violating the non-separation condition. Hence, either $\sigma[q] \in \{v_1^-, \ldots, v_n^-\}$, for all positions q between 1 and $p-1$, or for all positions $q' < p$ such that $\sigma[q'] = v_j^+$, $1 \leq j \leq n$, $\sigma[p]$ can occur before $\sigma[q']$ (Definition 4.1.7). In the former case, all markers in $\sigma[1,p]$ are vertex removal markers. In the latter case, we show that the connectivity invariant must be violated and therefore σ cannot be nice (Definition 4.1.6).

If $\sigma[1, p-1]$ contains at least one vertex addition marker, then for one such marker, say $\sigma[p']$, $\sigma[p']$ and $\sigma[p]$ must belong to the same add-remove segment in the central piece of σ. We let $\sigma[p'] = v_j^+$, $1 \leq j \leq n$. Since $\sigma[p]$ can occur before $\sigma[p']$, $v_j \neq v_i$, $v_j v_i \notin E(G)$, and the connectivity invariant is violated.

Finally, we note that if any segment β of an edit sequence σ consists of two or more consecutive vertex removal markers, then any reordering of those markers within β results in a valid edit sequence. Combining this observation with the fact that $\sigma[p]$ can occur before all vertex addition markers in $\sigma[1, p]$ (if any), it follows that $concat(\sigma[p], cut(\sigma, p))$ remains a valid edit sequence starting from S, as needed. □

Proposition 4.1.9. *Given a graph G, a vertex cover S of G, and a valid edit sequence σ starting from S, we let $\sigma[p] = v_i^-$, where $p > 1$ and $1 \leq i \leq n$. Then, $\{v_i\}$ and $V(\sigma[1, p-1])$ are not separated if and only if $\sigma[p]$ satisfies the non-separation condition.*

Proof. If $\sigma[p]$ satisfies the non-separation condition, then by Definition 4.1.7, there exists a position q between 1 and $p-1$ such that $\sigma[q] = v_j^+$, $1 \leq j \leq n$, and $\sigma[p]$ cannot occur before $\sigma[q]$. Hence, either $v_j = v_i$ or $v_j \in N_G(v_i)$. In both cases, $\{v_i\}$ and $V(\sigma[1, p-1])$ are not separated.

Conversely, if $\{v_i\}$ and $V(\sigma[1, p-1])$ are not separated, one of the following conditions must hold:

(1) $v_i^+ \in \sigma[1, p-1]$,

(2) $v_i^- \in \sigma[1, p-1]$,

(3) $\{v_j^+ \mid v_j \in N_G(v_i)\} \cap \sigma[1, p-1] \neq \emptyset$, or

(4) $\{v_j^- \mid v_j \in N_G(v_i)\} \cap \sigma[1, p-1] \neq \emptyset$.

We let $q < p$ denote the closest position to p such that $\sigma[q]$ satisfies one of the conditions above. If $\sigma[q]$ satisfies condition (2), then vertex v_i is removed at positions q and p. By our choice of q, this is not possible since v_i is not added in $\sigma[q+1, p-1]$ and σ is assumed to be valid. Similarly, if $\sigma[q]$ satisfies condition (4), then some vertex $w \in N_G(v_i)$ is removed at position q and v_i is removed at position p. By our choice of q, neither w nor v_i is added in $\sigma[q+1, p-1]$, leaving the edge wv_i uncovered. Consequently, if $\{v_i\}$ and $V(\sigma[1, p-1])$ are not separated, then either condition (1) or (3) must hold and therefore $\sigma[q]$ satisfies the non-separation condition. □

Given a graph G, vertex covers S_s and S_t of G, and a valid edit sequence σ from S_s to S_t, the ConvertToNice algorithm (Algorithm 1) constructs a new edit sequence σ' from S_s to S_t that is nice. Algorithm 1 can be divided into three stages. In the first stage (Lines 1–5), the starting piece of σ' is formed by copying all possible vertex removal markers in σ that violate the non-separation condition. By Proposition 4.1.8, we know that for σ' to be nice, all such markers must belong to its starting piece. Each copied marker is replaced with a $\varnothing^-$ in σ, which is then cut by applying the clean operation. If after the first stage $clean(\sigma)$ consists only of vertex addition markers, then we copy all remaining markers to the ending piece of σ', which will be a nice edit sequence with an empty central piece. This is handled in the third stage of the algorithm (Lines 25–27). Otherwise, we know that each remaining vertex removal marker in σ must satisfy the non-separation condition and is therefore preceded by at least one vertex addition.

The second stage of the algorithm (Lines 6–24) handles the central piece of σ'. The while loop is repeated as long as $clean(\sigma)$ contains at least one vertex removal marker. Each iteration handles the first occurrence of a vertex removal marker in σ and copies a $\Delta(G)$-add-remove segment from σ to σ' as follows. We find the first remaining vertex removal marker in σ, say $\sigma[p] = v_i^-$, $1 \leq p \leq |\sigma|$ and $1 \leq i \leq n$. Hence, all markers in $\sigma[1, p-1]$ are vertex addition markers. We then compute the set of pairs M_+ consisting of all markers in $\sigma[1, p-1]$ that have to occur before $\sigma[p]$ along with their respective positions in σ (Lines 7–9). Formally, $M_+ = \{(v_j^+, q) \mid (v_j \in N_G(v_i) \vee i = j) \wedge v_j^+ \in \sigma[1, p-1] \wedge \sigma[q] = v_j^+\}$. Each marker in M_+ corresponds to the addition of some vertex in $N_G[v_i]$. Although all markers in M_+ have to occur before $\sigma[p]$, executing only a subset of them might be enough for some vertex removal markers in $\sigma[p+1, |\sigma|]$ to become "executable". Hence, for every subset N_+ of M_+ in increasing order of size (Lines 10–12), we try executing all possible vertex addition markers in N_+ and form a $\Delta(G)$-add-remove segment in σ' as soon as some vertex removal markers become executable (possibly excluding $\sigma[p]$). We "simulate" the execution of vertex addition markers in N_+ by replacing their corresponding positions in σ with blank addition markers. If after these replacements we can find one or more vertex removal markers which no longer satisfy the non-separation condition in $clean(\sigma)$ (Lines 13–15), then such markers can be executed right after the addition markers in N_+ (Propositions 4.1.8 and 4.1.9). Hence, the markers in N_+ are copied to σ' followed by the (newly found) executable removal markers. Lines 16–18 guarantee that each remaining vertex removal in σ must again satisfy the non-separation condition and the process is repeated. Otherwise, the changes done to σ during the simulation are "reversed" (Lines 21–22) and we proceed to the next subset of M_+. As an example, consider a graph G where $\{v_1, \ldots, v_6\} \subseteq V(G)$ and $\{v_5v_1, v_5v_2, v_5v_3, v_5v_4, v_6v_1, v_6v_3\} \subseteq E(G)$ (Figure 4.3). In addition, assume $\sigma = \langle v_1^+, v_2^+, v_3^+, v_4^+, v_5^-, v_6^- \rangle$. The ConvertToNice algorithm will

Algorithm 1 CONVERTTONICE

Input: A graph G, vertex covers S_s and S_t, and a valid edit sequence σ from S_s to S_t.

Output: A nice edit sequence σ' from S_s to S_t.

```
1: $p = 1$; $\beta = \sigma$;
2: for ($1 \le i \le |\sigma|$)
3:     if ($\sigma[i] = v_x^-$, for $1 \le x \le n$, and $\{v_x\}$ is separated from $V(\beta[1, i-1])$)
4:         $\sigma'[p] = \sigma[i]$; $\sigma[i] = \varnothing^-$; $p = p + 1$;
5: $\sigma = clean(\sigma)$;
6: while ($\sigma$ contains at least one element from $\{v_1^-, \ldots, v_n^-\}$)
7:     for ($1 \le i \le |\sigma|$)
8:         if ($\sigma[i] = v_w^-$, for $1 \le w \le n$)
9:             $M_+ = \{(v_x^+, q) \mid (v_x \in N_G(v_w) \vee w = x) \wedge v_x^+ \in \sigma[1, i-1] \wedge \sigma[q] = v_x^+\}$;
10:    for each $N_+ \subseteq M_+$ (in increasing size order)
11:        for each $(v_x^+, q) \in N_+$, $1 \le x \le n$ and $1 \le q \le i - 1$
12:            $\sigma[q] = \varnothing^+$;
13:        if ($\exists j$ s.t. $\sigma[j] = v_y^-$, $1 \le y \le n$, and $\{v_y\}$ is separated from $V(\sigma[1, j-1])$)
14:            for each $(v_x^+, q) \in N_+$
15:                $\sigma'[p] = v_x^+$; $p = p + 1$;
16:            for ($1 \le l \le |\sigma|$)
17:                if ($\sigma[l] = v_z^-$, $1 \le z \le n$, and $\{v_z\}$ is separated from $V(\sigma[1, l-1])$)
18:                    $\sigma'[p] = \sigma[l]$; $\sigma[l] = \varnothing^-$; $p = p + 1$;
19:            break;
20:        else
21:            for each $(v_x^+, q) \in N_+$
22:                $\sigma[q] = v_x^+$;
23:    $\sigma = clean(\sigma)$;
24: end while
25: for ($1 \le i \le |\sigma|$)
26:     $\sigma'[p] = \sigma[i]$; $p = p + 1$;
27: return $\sigma'$;
```

produce $\sigma' = \langle v_1^+, v_3^+, v_6^-, v_2^+, v_4^+, v_5^- \rangle$. That is, v_6^- can occur right after v_1^+ and v_3^+ have been executed.

Lemma 4.1.10. *Given a graph G and two vertex covers S_s and S_t of G, the* CONVERT-TONICE *algorithm transforms any valid edit sequence σ from S_s to S_t into a nice edit sequence σ' from S_s to S_t. Moreover, $|V(S_s, \sigma'[1, p])| \le |V(S_s, \sigma[1, p])|$ for all $1 \le p \le |\alpha|$. That is, if σ is tight then so is σ'.*

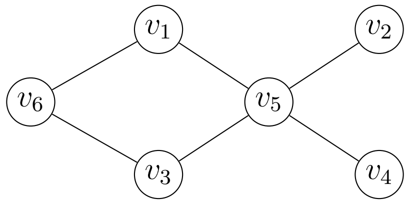

Figure 4.3: A graph G illustrating the ConvertToNice algorithm.

Proof. We divide the proof into three parts. Since σ' will contain exactly one copy of each marker in σ, we know that if σ' is valid starting from S_s then it is a valid edit sequence from S_s to S_t. Hence, we first show that σ' is a valid edit sequence (starting from S_s). We then show that σ' is a nice edit sequence. Finally, we prove that if σ is tight then so is σ'.

Claim 4.1.11. *The* ConvertToNice *algorithm produces a valid edit sequence σ'.*

Proof. We show that every time we copy a vertex deletion marker from σ to σ' and replace it with $\varnothing^-$ in σ, $concat(\sigma', clean(\sigma))$ remains a valid edit sequence. Since the third stage of the algorithm copies any remaining vertex addition markers to σ' in the same order they appear in $clean(\sigma)$, the claim follows.

A vertex deletion marker at position p, $1 \leq p \leq |\sigma|$, is copied from σ to σ' only when $V(\sigma[p])$ and $V(\sigma[1, p-1])$ are separated (Lines 3–4 and 17–18). Consequently, Proposition 4.1.9 implies that $\sigma[p]$ violates the non-separation condition. Therefore, by Proposition 4.1.8, $concat(\sigma[p], cut(\sigma, p))$, and hence $concat(concat(\sigma', \sigma[p]), cut(\sigma, p))$, remains a valid edit sequence. □

Claim 4.1.12. *The* ConvertToNice *algorithm produces a nice edit sequence σ'.*

Proof. We start by showing that σ' is in fact $\Delta(G)$-well-formed. If the central piece of σ' is empty, the first and third stages of the algorithm guarantee that σ' will be $\Delta(G)$-well-formed. When σ' has a non-empty central piece, we show that every iteration of the while loop will copy exactly one $\Delta(G)$-add-remove segment from σ to σ'.

First we note that Lines 2–4 and 16–18 guarantee that every vertex removal marker in σ must satisfy the non-separation condition prior to every iteration of the while loop. Hence, the first marker in σ must be a vertex addition marker at this point in the algorithm. We let v_w^-, $1 \leq w \leq n$, denote the first vertex removal marker in σ (Line 8). Combining Proposition 4.1.9 with the fact that G has degree at most $\Delta(G)$, we know that $|M_+| \leq \Delta(G)$. Note that M_+ cannot consist of all the vertices in the closed neighborhood of v_w since this would leave some edge in G uncovered. If we replace all vertex addition markers in M_+

by $\varnothing^+$ in σ, then v_w^- will satisfy all the conditions on line 13. Thus, in every iteration of the while loop, lines 14–19 will execute exactly once. We let $N_+ \subseteq M_+$ denote the set which results in the conditions on line 13 being true. Since we iterate over the subsets of M_+ in increasing order of size, we know that for the first vertex removal marker v_z^- satisfying the conditions on line 17, v_z is not separated from any of the vertices in $\{v_x \mid \{v_x^+, q\} \in N_+\}$. Thus, for every $v_x \in \{v_x \mid \{v_x^+, q\} \in N_+\}$, either $x = z$ or $v_x v_z \in E(G)$. Moreover, if v_z^- is at position q_1, $1 \leq q_1 \leq |\sigma|$, any other vertex removal marker v_y^- at position $q_2 > q_1$, $q_2 \leq |\sigma|$, which now satisfies the conditions on line 17 is also not separated from any of the vertices in $\{v_x \mid \{v_x^+, q\} \in N_+\}$; otherwise $|N_G[v_y] \cap \{v_x \mid \{v_x^+, q\} \in M_+\}| < |N_+|$, contradicting the fact that N_+ is the smallest subset of M_+, which results in the conditions on line 13 being true. Finally, we note that since we iterate over the subsets of M_+ in increasing order of size, at most $\Delta(G)$ vertex removal markers will satisfy the conditions on line 17. Putting it all together, Lines 14–19 will execute exactly once and copy at most $\Delta(G)$ vertex addition markers followed by at most $\Delta(G)$ vertex removal markers.

We now show that every $\Delta(G)$-add-remove segment in σ' satisfies the connectivity invariant. We let β denote such a segment. The fact that $G[V(\beta)]$ is connected and induces a biclique follows from the observation that every removed vertex in β is not separated from any of the added vertices in β. Moreover, as σ' is valid, there can be no edges between two added vertices or two removed vertices in β.

Finally, the early removal invariant is satisfied because vertex removal markers are copied from σ to σ' as soon as they no longer satisfy the non-separation condition (lines 2–4 and 16–18). We conclude that σ' is a nice edit sequence from S_s to S_t. □

Claim 4.1.13. *If σ is tight then so is σ'.*

Proof. To prove $|V(S_s, \sigma'[1,p])| \leq |V(S_s, \sigma[1,p])|$, for all $1 \leq p \leq |\sigma|$, we show that the number of vertex removal markers in $\sigma'[1,p]$ is greater than or equal to the number of vertex removal markers in $\sigma[1,p]$. We assume without loss of generality that $\sigma[1]$ is a vertex addition marker, as all vertex removal markers which are not preceded by a vertex addition marker in σ will be copied to σ' in the first stage of the algorithm. In lines 13–19 of the algorithm, the number of vertex addition markers copied from σ to σ' is equal to $|N_+| \leq |M_+|$. Moreover, whenever any marker is shifted from position q_1 in σ to position $q_2 > q_1$ in σ', it must be the case that some vertex removal marker was shifted from position $q_3 > q_1$ in σ to some position $q_4 < q_2$ in σ'. In other words, if any marker is "shifted to the right" when copied from σ to σ' it must be the case that a removal marker was "shifted" to the left. Hence, there exists no p such that $\sigma[1,p]$ contains more vertex removal markers than $\sigma'[1,p]$. □

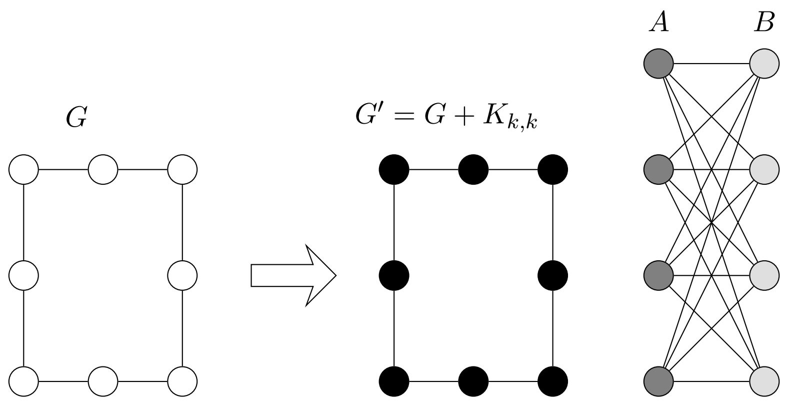

Figure 4.4: The graph G' consisting of the disjoint union of a graph G and a biclique $K_{k,k}$.

The statement of the lemma follows by combining Claims 4.1.11, 4.1.12, and 4.1.13. □

4.2 A meta-theorem

Before we define the collection of hereditary properties for which our meta-theorem applies, we consider Π to be the set of all edgeless graphs and prove the following lemma, which is a building block of the meta-theorem.

Lemma 4.2.1. VC-Bound *parameterized by* ℓ*,* IS-Reach *parameterized by* k*, and* IS-Bound *parameterized by* $k + \ell$ *are at least as hard as* Independent Set *parameterized by* k*.*

Proof. Given an instance (G, k) of Independent Set, we construct a graph G' by taking the disjoint union of G and a $K_{k,k}$ (Figure 4.4). Formally, we let $V(G') = A \cup B \cup V_G$, where $A = \{a_i \mid 1 \leq i \leq k\}$, $B = \{b_i \mid 1 \leq i \leq k\}$, and $V_G = \{g_i \mid v_i \in V(G)\}$. The edge set of G' is therefore defined as $E(G') = \{g_i g_j \mid v_i v_j \in E(G)\} \cup \{a_i b_j \mid 1 \leq i, j \leq k\}$. We let $(G', S_s = V_G \cup A, S_t = V_G \cup B, |V_G| + k, \ell = 4k)$ be an instance of VC-Bound. In Figure 4.4, S_s consists of the union of black and dark gray vertices and S_t consists of the union of black and light gray vertices.

If there is a reconfiguration sequence from S_s to S_t, such a sequence cannot remove a vertex from A before adding all vertices in B, as otherwise some edge would be left uncovered. In other words, whether we start from S_s and attempt to reach S_t or vice-versa, one node, say S_f, along any reconfiguration sequence between the two must consist of a vertex cover containing all vertices in $A \cup B$. However, our maximum allowed capacity is $|V_G| + k$ and $|A \cup B| = 2k$. Therefore, $|S_f \cap V_G| \leq |V_G| - k$. Moreover, since S_f must be a vertex cover of G', we know that $S_f \cap V_G$ must be a vertex cover of $G'[V_G]$ (or equivalently G) of size at most $|V_G| - k$. Consequently, $I_f = V_G \setminus \{S_f \cap V_G\}$ is an independent set of size at least k.

For the converse, we let I be an independent set in G (or equivalently in $G'[V_G]$) of size at least k and show how to obtain a reconfiguration sequence of length at most $4k$ from S_s to S_t. The following sequence of $4k$ steps transforms S_s to S_t: Remove k vertices from I, add all vertices in B, remove all vertices in A, and finally add back the k vertices in I.

We have shown that G has an independent set of size at least k if and only if there is a path of length at most $4k$ from S_s to S_t in $R_{VC}(G', 0, |V(G)| + k)$. Since $|V(G')| - (|V(G)| + k) = k$, this implies that G has an independent set of size at least k if and only if there is a path of length at most $4k$ from $V(G') \setminus S_s$ to $V(G') \setminus S_t$ in $R_{IS}(G', k, |V(G')|)$. Therefore, IS-Reach parameterized by k and IS-Bound parameterized by $k + \ell$ are also at least as hard as Independent Set parameterized by k. □

To generalize Lemma 4.2.1, we make use of the forbidden set characterization of heredity properties (Section 2.1.1). A Π-*critical graph* H is a (minimal) graph in the forbidden set $\mathcal{F}_\Pi$ that has at least two vertices; we use the fact that $H \notin \Pi$, but the deletion of any vertex from H results in a graph in Π. For convenience, we will refer to two of the vertices in a Π-critical graph as *terminals* and the rest as *internal vertices*. We construct graphs from multiple copies of H. For a positive integer c, we let H_c^* be the star graph obtained from each of c copies H_i of H by identifying an arbitrary terminal v_i, $1 \leq i \leq c$, from each H_i; in H_c^* vertices v_1 through v_c are replaced with a vertex w, the *gluing vertex of v_1 to v_c*, to form a graph with vertex set $\cup_{1 \leq i \leq c}(V(H_i) \setminus \{v_i\}) \cup \{w\}$ and edge set $\cup_{1 \leq i \leq c}\{\{u, v\} \in E(H_i) \mid v_i \notin \{u, v\}\} \cup \cup_{1 \leq i \leq c}\{\{u, w\} \mid \{u, v_i\} \in E(H_i)\}$. A terminal is *non-identified* if it is not used in forming a gluing vertex.

In Figure 4.5, H is a K_3 with terminals marked black and gray; H_4^* is formed by identifying all the gray terminals to form w.

Theorem 4.2.2. *Let Π be any hereditary property satisfying the following:*

- *For any two graphs G_1 and G_2 in Π, the graph obtained by their disjoint union is in Π.*

- *There exists an $H \in \mathcal{F}_\Pi$ such that if H_c^* is the graph obtained from identifying a terminal from each of c copies of H, then the graph $R = H_c^*[V(H_c^*) \setminus \{u_1, u_2, \dots u_c\}]$ is in Π, where $u_1, u_2, \dots u_c$ are the non-identified terminals in the c copies of H.*

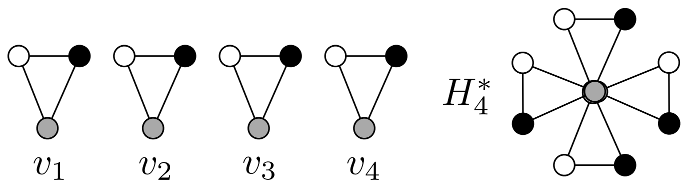

Figure 4.5: An example H_c^* and its components.

Then each of the following is at least as hard as Π-Sub(G, k) *parameterized by* k*:*

1. Π-Del-Bound(G, S_s, S_t, k, ℓ) *parameterized by* ℓ*,*
2. Π-Sub-Reach(G, S_s, S_t, k) *parameterized by* k*, and*
3. Π-Sub-Bound(G, S_s, S_t, k, ℓ) *parameterized by* $k + \ell$*.*

Proof. Given an instance (G, k) of Π-Sub and a Π-critical graph H satisfying the hypothesis of the theorem, we form an instance $(G', S_s, S_t, |V(G)| + k, 4k)$ of Π-Del-Bound, with G', S_s, and S_t defined below. The graph G' is the disjoint union of G and a graph W formed from k^2 copies of H, where $H_{i,j}$ has terminals $\ell_{i,j}$ and $r_{i,j}$. We let a_i, $1 \leq i \leq k$, be the gluing vertex of $\ell_{i,1}$ through $\ell_{i,k}$, and let b_j, $1 \leq j \leq k$, be the gluing vertex of $r_{1,j}$ through $r_{k,j}$, so that there is a copy of H joining each a_i and b_j. An example W is shown in Figure 4.6, where copies of H are shown schematically as gray ovals. We let $A = \{a_i \mid 1 \leq i \leq k\}$, $B = \{b_i \mid 1 \leq i \leq k\}$, $C = \{c \mid c \in V(W) \setminus \{A \cup B\}\}$, and $V_G = \{g_i \mid v_i \in V(G)\}$. The vertex and edge sets of G' are therefore defined as $V(G') = A \cup B \cup C \cup V_G$ and $E(G') = \{g_i g_j \mid v_i v_j \in E(G)\} \cup E(W)$. We fix $S_s = A \cup V_G$ and $S_t = B \cup V_G$. Clearly, $|V(G')| = |V(G)| + 2k + k^2(|V(H)| - 2)$ and $|S_s| = |S_t| = |V(G)| + k$. Moreover, each of $V(G') \setminus S_s$ and $V(G') \setminus S_t$ induce a graph in Π, as each consists of k disjoint copies of H_k^* with one of the terminals removed from each H in H_k^*.

Suppose the instance $(G', S_s, S_t, |V(G)| + k, 4k)$ of Π-Del-Bound is a yes-instance. As there is a copy of H joining each vertex of A to each vertex of B, before removing $a \in A$ from S_s, the reconfiguration sequence must add all of B to ensure that the complement of each intermediate set induces a graph in Π. Otherwise, the complement will contain at least one copy of H as a subgraph and is therefore not in Π. The capacity bound of $|V(G)| + k$ implies that the reconfiguration sequence must have removed from S_s a subset $S' \subseteq V_G$ of size at least k such that $V(G') \setminus (S_s \setminus S') = S' \cup B$ induces a subgraph in Π. Thus, $G[S'] \in \Pi$, and hence Π-Sub(G, k) is a yes-instance.

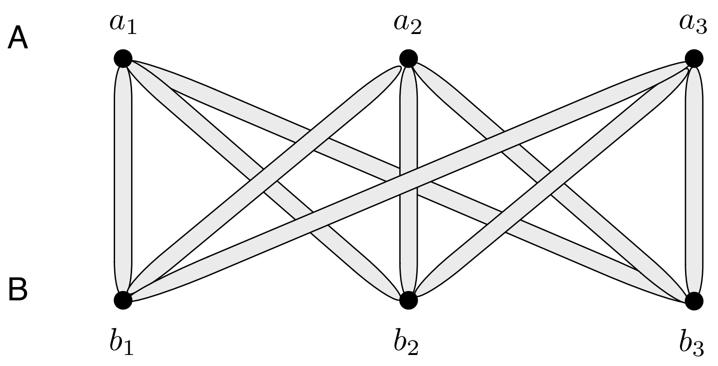

Figure 4.6: An example W

Conversely, if the instance (G, k) of Π-Sub is a yes-instance, then there exists $S' \subseteq V(G)$ such that $|S'| = k$ and $G[S'] \in \Pi$. We form a reconfiguration sequence between S_s and S_t by first deleting all vertices in S' from S_s to yield a set of size $|V(G)|$. $G'[V(G') \setminus (S_s \setminus S')]$ consists of the union of $G'[V'(G) \setminus S_s]$ and $G'[S'] = G[S']$, both of which are in Π. Next we add one by one all vertices of B, then delete one by one all vertices of A and then add back one by one each vertex in the set S' resulting in a reconfiguration sequence of length $k + k + k + k = 4k$. It is clear that in every step, the complement of the set induces a graph in Π.

Thus we have showed that Π-Sub(G, k) is a yes-instance if and only if there is a path of length at most $4k$ between S_s and S_t in $R_{\overline{\Pi}}(G', 0, |V(G)| + k)$. Since $|V(G')| - (|V(G)| + k) = k + k^2(|V(H)| - 2))$, this implies that Π-Sub(G, k) is a yes-instance if and only if there is a path of length at most $4k$ between $V(G') \setminus S_s$ and $V(G') \setminus S_t$ in $R_\Pi(G', k + k^2(|V(H)| - 2), |V(G')|)$. Therefore, Π-Sub-Reach parameterized by k and Π-Sub-Bound parameterized by $k + \ell$ are at least as hard as Π-Sub parameterized by k □

Corollary 4.2.3. FVS-Bound *and* OCT-Bound *are* `W[1]`*-hard parameterized by ℓ.* IF-Reach *and* IBS-Reach *are* `W[1]`*-hard parameterized by k.* IF-Bound *and* IBS-Bound *are* `W[1]`*-hard parameterized by $k + \ell$.*

Proof. It is known that for any hereditary property Π that consists of all edgeless graphs but not all cliques [93], Π-Sub parameterized by k is `W[1]`-hard. It is clear that the collections of all bipartite graphs and of all forests satisfy this condition for hardness, as well as the hypothesis of Theorem 4.2.2; we let $H \in \mathcal{F}_\Pi$ be a triangle. When we identify multiple triangles at a vertex, and remove another vertex of each of the triangles, we obtain a tree, which is in Π. $\square$

We obtain further results for interesting properties not covered by Theorem 4.2.2. Lemma 4.2.4 handles the collection of all cliques, which does not satisfy the first condition of the theorem, and the collection of all *cluster graphs* (disjoint unions of cliques), which satisfies the first condition but not the second. Moreover, as Π-SUB parameterized by k is fixed-parameter tractable for Π the collection of all cluster graphs [93], Theorem 4.2.2 provides no lower bounds. It remains open whether Theorem 4.2.2 can be generalized even further to cover such properties.

Lemma 4.2.4. *CLIQUE-REACH and CLUSTER SUBGRAPH-REACH parameterized by k are* `W[1]`*-hard. CLIQUE-BOUND and CLUSTER SUBGRAPH-BOUND parameterized by $k+\ell$ are* `W[1]`*-hard.*

Proof. We first give an `FPT` reduction from CLIQUE, known to be `W[1]`-hard, to CLUSTER SUBGRAPH-BOUND. For (G, k) an instance of CLIQUE, $V(G) = \{v_1, \ldots, v_n\}$, we form a graph consisting of four K_k's (with vertex sets A, B, C, and D) and a subgraph mimicking G (with vertex set X), where there is an edge from each vertex in X to each vertex in each K_k, and each of the subgraphs on the following vertex sets induce a K_{2k}: $A \cup B$, $A \cup C$, $B \cup D$, $C \cup D$. Formally, $V(G') = X \cup A \cup B \cup C \cup D$ and $E(G') = E_X \cup E_T \cup E_C$, where $X = \{x_1, \ldots, x_n\}$, $|A| = |B| = |C| = |D| = k$, $E_X = \{\{x_i, x_j\} \mid \{v_i, v_j\} \in E(G)\}$ corresponds to the edges in G, $E_T = \{\{a, a'\} \mid a, a' \in A, a \neq a'\} \cup \{\{b, b'\} \mid b, b' \in B, b \neq b'\} \cup \{\{c, c'\} \mid c, c' \in C, c \neq c'\} \cup \{\{d, d'\} \mid d, d' \in D, d \neq d'\}$ forms the K_k cliques, and $E_C = \{\{x, a\}, \{x, b\}, \{x, c\}, \{x, d\}, \{a, b\}, \{a, c\}, \{b, d\}, \{c, d\} \mid a \in A, b \in B, c \in C, d \in D, x \in X\}$ forms the connections among the vertex sets.

We let $(G', S_s, S_t, 2k, 6k)$ be an instance of CLUSTER SUBGRAPH-BOUND, where $S_s = A \cup B$ and $S_t = C \cup D$. Clearly $|S_s| = |S_t| = 2k$ and both S_s and S_t induce cluster graphs (in fact cliques). We claim that G has a clique of size k if and only if there is a reconfiguration sequence of length $6k$ from S_s to S_t.

If G has a clique of size k, then there exists a subset $Y \subseteq X$ forming a clique of size k. We form a reconfiguration sequence of length $6k$ as follows: Add the vertices in Y, remove the vertices in A, add the vertices in D, remove the vertices in B, add the vertices in C, and remove the vertices in Y, one by one. It is not hard to see that at every step in this sequence we maintain an induced clique in G' of size greater than or equal to $2k$ (and hence a cluster subgraph).

If there exists a reconfiguration sequence of length $6k$ from S_s to S_t, we make use of the fact that no cluster subgraph contains an induced path of length three to show that G

has a clique of size k. Observe that before adding any vertex of C, we first need to remove (at least) all of B since otherwise we obtain an induced path of length three containing vertices in C, A, and B, respectively. Similarly, we cannot add any vertex of D until we have removed all of A. Therefore, before adding any vertex from S_t, we first need to delete at least k vertices from S_s. To do so without violating our minimum capacity of $2k$, at least k vertices must be added from X. Since every vertex in X is connected to all vertices in S_s and S_t, if any pair of those k vertices do not share an edge, we obtain an induced path on three vertices. Thus X, and hence G, must have a clique of size k.

Since in our reduction S_s and S_t are cliques and every reconfiguration step maintains an induced clique in G' of size greater than or equal to $2k$, it follows that Clique-Reach and Cluster Subgraph-Reach parameterized by k and Clique-Bound and Cluster Subgraph-Bound parameterized by $k + \ell$ are all `W[1]`-hard. □

4.3 Compression via reconfiguration

In this section, we prove the following theorem and give an example of how it can be applied to prove `NP`-hardness of reachability and bounded reachability problems when restricted to certain graph classes.

Theorem 4.3.1. *Let Π be any hereditary property satisfying the following:*

- *For any two graphs G_1 and G_2 in Π, the graph obtained by their disjoint union is in Π.*
- *There exists an $H \in \mathcal{F}_\Pi$ such that if H_c^* is the graph obtained from identifying a terminal from each of c copies of H, then the graph $R = H_c^*[V(H_c^*) \setminus \{u_1, u_2, \ldots u_c\}]$ is in Π, where $u_1, u_2, \ldots u_c$ are the non-identified terminals in the c copies of H.*

Then Π-Del-Reach and Π-Del-Bound are at least as hard as Π-Comp.

Proof. We give a reduction from Π-Comp to Π-Del-Bound. Given an instance (G, S, k) of Π-Comp and a Π-critical graph H satisfying the hypothesis of the theorem, we construct, as we did in the proof of Theorem 4.2.2, a graph G' as follows.

The graph G' is the disjoint union of G and a graph W formed from k^2 copies of H, where $H_{i,j}$ has terminals $\ell_{i,j}$ and $r_{i,j}$. We let a_i, $1 \leq i \leq k$, be the gluing vertex of $\ell_{i,1}$ through $\ell_{i,k}$, and let b_j, $1 \leq j \leq k$, be the gluing vertex of $r_{1,j}$ through $r_{k,j}$, so that there is a copy of H joining each a_i and b_j. We let $A = \{a_i \mid 1 \leq i \leq k\}$, $B = \{b_i \mid 1 \leq i \leq k\}$, $C = \{c \mid c \in V(W) \setminus \{A \cup B\}$, and $V_G = \{g_i \mid v_i \in V(G)\}$. The vertex and edge sets of G' are therefore defined as $V(G') = A \cup B \cup C \cup V_G$ and $E(G') = \{g_i g_j \mid v_i v_j \in E(G)\} \cup E(W)$.

We let $(G', S_s, S_t, 3k-1, 6k-2)$ be an instance of Π-Del-Bound, where $S_s = A \cup \{s_i \mid v_i \in S\}$ and $S_t = B \cup \{s_i \mid v_i \in S\}$. Clearly, $|V(G')| = |V(G)| + 2k + k^2(|V(H)| - 2)$ and $|S_s| = |S_t| = k + |S| = 2k$. Moreover, each of $V(G') \setminus S_s$ and $V(G') \setminus S_t$ induce a graph in Π, as each consists of k disjoint copies of H_k^* with one of the terminals removed from each H in H_k^*. We claim that there exists a set $S' \subseteq V(G)$ such that $|S'| < S$ and $G[V(G) \setminus S'] \in \Pi$ if and only if there is a reconfiguration sequence of length $6k-2$ or less from S_s to S_t.

As there is a copy of H joining each vertex of A to each vertex of B, before removing $a \in A$ from S_s the reconfiguration sequence must add all of B to ensure that the complement of each intermediate set induces a graph in Π. But $2k + k = 3k > 3k-1$, which violates the maximum allowed capacity. Therefore, if there is a reconfiguration sequence from S_s to S_t, then one of the sets in the sequence must contain at most $2k-1$ vertices. Of those $2k-1$ vertices, k vertices correspond to the vertices in A and cover only the forbidden subgraphs in W. Thus, the remaining $k-1$ vertices must be in V_G and should cover all the forbidden subgraphs in $G[V_G]$. By our construction of G', these $k-1$ vertices correspond to a set $S' \subseteq V(G)$ such that $|S'| < S$ and $G[V(G) \setminus S'] \in \Pi$, as needed.

Similarly, if there exists a set $S' \subseteq V(G)$ such that $|S'| \leq k-1$ and $G[V(G) \setminus S'] \in \Pi$, then the following reconfiguration sequence transforms S_s to S_t: add all vertices of S', remove all vertices of S, add all vertices of B, remove all vertices from A, and finally add back all vertices of S and remove those of S'. The length of this sequence is equal to $6k-2$ whenever $S \cap S' = \emptyset$ and is shorter otherwise. □

4.3.1 An application: NP-hardness on 4-regular graphs

An immediate consequence of Theorem 4.3.1 is that both VC-Reach and VC-Bound are at least as hard as VC-Comp. Moreover, the graph G' resulting from our reduction consists of the disjoint union of the input graph G and a biclique $K_{k,k}$. Since any algorithm which solves VC-Comp can be used to solve the VC problem itself (by running the algorithm at most n times), we know that VC-Comp is at least as hard as VC. Therefore, if we know that VC is NP-hard on a graph class $\mathcal{C}$ then VC-Comp is also NP-hard on $\mathcal{C}$. Hence, to show that VC-Reach and VC-Bound are also NP-hard on $\mathcal{C}$, we only need to replace the biclique $K_{k,k}$ by a graph which is also in $\mathcal{C}$. Needless to say, constructing such a graph is not always an easy task. We give one example which shows that VC-Reach and VC-Bound are NP-hard on 4-regular graphs. That is, we construct a 4-regular gadget W_k (which will replace $K_{k,k}$) on $6k^2$ vertices with the following property: There exist two minimum vertex-disjoint vertex covers S_s and S_t of W_k, each of size $3k^2$, such that there exists a path of length $6k^2$ between the nodes corresponding to S_s and S_t in $R_{VC}(W_k, 0, 3k^2 + g(k))$ but no such path exists in $R_{VC}(W_k, 0, 3k^2 + g(k) - 1)$, for some computable function g and $k - 2 \leq g(k) \leq k + 3$. The existence of graphs with properties similar to W_k has played an important role in determining the complexity of other reconfiguration problems [25]. For instance, and since VC is NP-hard on both 3-regular and 4-regular graphs [64], the existence of a 3-regular version of W_k, combined with Theorem 4.3.1, would immediately imply that VC-Reach and VC-Bound are **NP**-hard on 3-regular graphs.

We describe W_k (Figure 4.7) in terms of several component subgraphs, each playing a role in forcing the reconfiguration of vertex covers. A *k-necklace*, $k \geq 4$, is a graph obtained by replacing each edge in a cycle on k vertices by two vertices and four edges. For convenience, we refer to every vertex on the original cycle as a *bead* and every new vertex in the resulting graph as a *sequin*. The resulting graph has k beads each of degree four and $2k$ sequins each of degree two. Every two sequins that share the same neighborhood in a k-necklace are called a *sequin pair*. We say two beads are *adjacent* whenever they share exactly two common neighbors. Similarly, we say two sequin pairs are *adjacent* whenever they share exactly one common neighbor. Every two adjacent beads (sequin pairs) are *linked* by a sequin pair (bead).

The graph W_k consists of $2k$ copies of a k-necklace. We let $U = \{U_1, \ldots U_k\}$ and $L = \{L_1, \ldots L_k\}$ denote the first and second k copies respectively; for convenience, we use the terms "upper" and "lower" to mean "in U" and "in L", respectively. We let $b^u_{i,j}$ and $b^l_{i,j}$ denote the jth beads of necklace U_i and L_i respectively, where $1 \leq i \leq k$ and $1 \leq j \leq k$. Beads on each necklace in W_k are numbered consecutively in clockwise order from 1 to k. For every two adjacent beads $b^x_{i,j}$ and $b^x_{i,j+1}$, where $x \in \{u, l\}$, we let $p^x_{i,j}$ denote the sequin pair which links both beads.

For each sequin pair $p^l_{i,j}$, we add four edges to form a $K_{2,2}$ (a *joining biclique*) with the pair $p^u_{j,i}$, for all $1 \leq i, j \leq k$ (Figure 4.7); we say that sequin pairs $p^l_{i,j}$ and $p^u_{j,i}$ are *joined*. All k^2 joining bicliques in W_k are vertex disjoint. The total number of vertices in W_k is $6k^2$. Every vertex has degree exactly four; every bead is connected to four sequins from the same necklace and every sequin is connected to two beads from the same necklace and two other sequins from a different necklace. We let S_s be the set containing all upper beads and lower sequins, whereas S_t contains all lower beads and upper sequins. Formally, $S_s = \{b^u_{i,j} \mid 1 \leq i, j \leq k\} \cup \{v \in p^l_{i,j} \mid 1 \leq i, j \leq k\}$ and $S_t = \{b^l_{i,j} \mid 1 \leq i, j \leq k\} \cup \{v \in p^u_{i,j} \mid 1 \leq i, j \leq k\}$. Each set contains $3k^2$ vertices, that is, half the vertices in W_k.

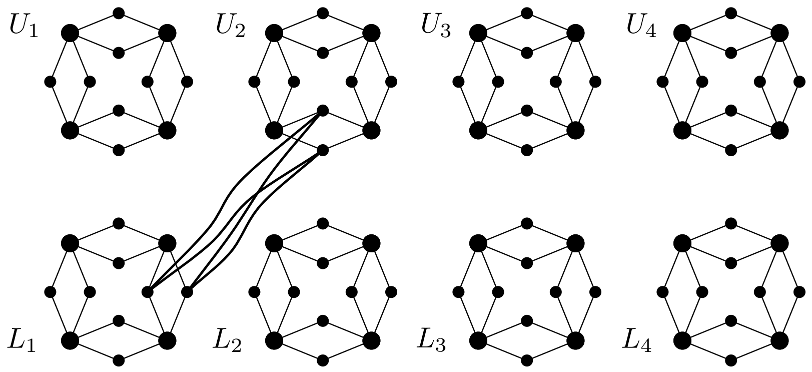

Figure 4.7: The graph W_4 (the edges of one out of the k^2 joining bicliques are shown).

Proposition 4.3.2. *S_t and S_t are minimum vertex covers of W_k.*

Proof. We need at least $2k^2$ vertices to cover the edges in the k^2 vertex disjoint joining bicliques contained in W_k. Moreover, any minimal vertex cover S of W_k which includes a vertex v from a sequin pair $p^x_{i,j} = \{v, w\}$, where $x \in \{u, l\}$, must also include w. Otherwise, the two beads linking $p^x_{i,j}$ to its adjacent sequin pairs must be in S to cover the edges incident on w, making v removable. Hence, any minimal vertex cover S of W_k must include either one or both sequin pairs in a joining biclique.

We let x denote the number of joining bicliques from which two sequin pairs are included in S. Similarly, we let y denote the number of joining bicliques from which only one sequin pair is included in S. Hence, $x + y = k^2$ and $|S| \geq 4x + 2y$. When $y = 0$, $|S| \geq 4k^2$ and S cannot be a minimum vertex cover, as S_s (and S_t) are both vertex covers of W_k of size $3k^2$. When $y \geq 1$, we are left with at least y uncovered edges incident to the sequin pairs not in S. Those edges must be covered using at least y beads and hence $|S| \geq 4x + 3y$. If we assume $4x + 3y < 3k^2$, we get a contradiction since $4x + 4y = 4k^2 < 3k^2 + y$ and $k^2 < y$. Therefore, S_s and S_t must be minimum vertex covers of W_k. □

To prove the next two lemmas, we consider the representation of reconfiguration sequences as nice edit sequences. We know from Lemma 4.1.10 that any reconfiguration sequence of length $6k^2$ from S_s to S_t can be converted into a nice edit sequence σ of the same length. Since S_s is a minimal vertex cover of W_k, σ cannot start with a vertex removal and hence the starting piece of σ must be empty. Since $V(S_s, \sigma[1, |\sigma|-1])$ is a vertex cover of W_k, $|S_s| = |S_t|$, and S_s and S_t are minimum vertex covers of W_k, σ cannot end with a vertex addition and hence the ending piece of σ must also be empty. Moreover, $|\sigma| = 6k^2 = |V(W_k)|$ implies that σ must touch every vertex in W_k exactly once. Since

W_k is a 4-regular graph, each d-add-remove segment in the central piece of σ consists of at most four additions followed by at most four removals. Hence, we know $\sigma = \sigma_1\sigma_2 \ldots \sigma_j$, where each σ_i is a 4-add-remove segment in σ.

Lemma 4.3.3. *Any nice edit sequence σ of length $6k^2$ from S_s to S_t either adds or removes both vertices u and v in a sequin pair in the same 4-add-remove segment β.*

Proof. The two vertices in a sequin pair share the same neighborhood. Hence, when u is removed, all of its neighbors must have been added, also making v removable. Moreover, since every vertex is touched exactly once in σ, none of the neighbors of both u and v will be touched in σ after the removal of u. Therefore, if v is not removed in the same segment β as u, the early removal invariant of Definition 4.1.6 will be violated since v is separated from any vertex that gets added after β (all neighbors of v are added either in β or prior to β).

For the case of additions, if only u is added in β then none of its neighbors can be removed. It follows that u is not connected to any vertex in $V(\beta)$, hence violating the connectivity invariant of Definition 4.1.6. □

Lemma 4.3.4. *There exists a function of k, $f(k)$, such that $(W_k, S_s, S_t, 3k^2 + f(k), \ell)$ is a yes-instance and $(W_k, S_s, S_t, 3k^2 + f(k) - 1, \ell)$ is a no-instance of* VC-Bound*, for $\ell = 6k^2$. Moreover, $k - 2 \leq f(k) \leq k + 3$.*

Proof. To show that such an $f(k)$ exists, we first prove the $k-2$ lower bound by showing that any nice edit sequence $\sigma = \sigma_1\sigma_2 \ldots \sigma_j$ of length $6k^2$ from S_s to S_t must have some prefix with exactly $5k$ vertex removals and at least $6k - 2$ additions. We let position x, $1 \leq x \leq |\sigma|$, be the smallest position such that $\sigma[1, x]$ contains exactly $5k$ vertex removals. Those $5k$ vertices correspond to a set $S \subset S_s$, as σ touches every vertex exactly once. The claim is that $\sigma[1, x]$ must contain at least $6k - 2$ vertex additions. We let $T \subset S_t$ denote the set of added vertices in $\sigma[1, x]$. Since $N_{W_k}(S) \subseteq T$, we complete the proof of the lower bound by showing that

$$|T| \geq |N_{W_k}(S)| \geq \frac{6}{5}|S| - 2 \geq 6k - 2.$$

To do so, we show that for any $S \subset S_s$ of size $5k$, $N_{W_k}(S) \subseteq T$ contains at least

$$\frac{6}{5}|S| - 2 = 6k - 2$$

vertices.

In what follows, we restrict our attention to the bipartite graph $Z = W_k[S \cup T]$ and we let S and T denote the two partitions of Z. We subdivide S into two sets: S_u contains upper beads and S_l contains lower sequins. Since every vertex in S_u has four neighbors in T and adjacent beads share exactly two neighbors, we have $|N_Z(S_u)| \geq 2|S_u|$ and equality occurs whenever S_u contains $2k$ beads from the same two upper necklaces. Whenever S_u contains fewer than $2k$ beads and $Z[S_u \cup N_Z(S_u)]$ consists of $t_u \geq 1$ connected components, at least one bead from each component (except possibly the first) will be adjacent to at most one other bead in the same component. Therefore, $|N_Z(S_u)| \geq 2|S_u| + 2(t_u - 1)$.

Lemma 4.3.3 implies that T will always contain both vertices of any sequin pair. Since we are only considering vertices in $V(\sigma[1, x])$, some sequins in S_l might be missing the other sequin in the corresponding pair. However, all the neighbors of the sequin pair have to be in T so we assume without loss of generality that vertices in S_l can be grouped into sequin pairs. Every sequin pair in S_l has four neighbors in T. Adjacent sequin pairs share exactly one neighbor. Hence, $|N_Z(S_l)| \geq \frac{3}{2}|S_l|$ and equality occurs whenever S_l contains k sequin pairs of a single lower necklace. Whenever S_l contains fewer than k sequin pairs and $Z[S_l \cup N_Z(S_l)]$ consists of $t_l \geq 1$ connected components, at least one sequin pair from each component will be adjacent to at most one other sequin pair in the same component. Therefore, $|N_Z(S_l)| \geq \frac{3}{2}|S_l| + t_l$.

Combining the previous observations, we know that when either S_u or S_l is empty, $|N_Z(S)| \geq \frac{6}{5}|S|$, as needed. When both are not empty, we let $I = N_Z(S_u) \cap N_Z(S_l)$. Hence, $|N_Z(S_u)| + |N_Z(S_l)| - |I| \geq 2|S_u| + 2(t_u - 1) + \frac{3}{2}|S_l| + t_l - |I|$ and we rewrite it as:

$$|N_Z(S)| + 2 \geq \frac{100}{50}|S_u| + \frac{75}{50}|S_l| + 2(t_u - 1) + t_l - (|I| - 2) \tag{4.1}$$

We now bound the size of I. Note that I can only contain upper sequin pairs joined with sequin pairs in S_l. As every sequin pair in S_l has either zero or two neighbors in I, $|S_l| \geq |I|$. Moreover, for every two sequin pairs in S_l having two neighbors in I, there must exist at least one vertex in S_u, which implies $|S_u| \geq \frac{|I|}{4}$. Finally, whenever a sequin pair $p \in S_l$ has two neighbors in I, then $t_u, t_l \geq 1$ as at least one bead neighboring the sequin pair joined with p must be in S_u. Every other sequin pair $p' \in S_l$, $p' \neq p$, with two neighbors in I will force at least one additional connected component in either $Z[S_u \cup N_Z(S_u)]$ or $Z[S_l \cup N_Z(S_l)]$ since W_k contains a single joining biclique between any two necklaces. Therefore, the total number of connected components is $t_u + t_l \geq \frac{|I|}{2}$. Putting it all together, we obtain:

$$\begin{aligned}\frac{40}{50}|S_u| + \frac{15}{50}|S_l| + 2(t_u - 1) + t_l &\geq \frac{2}{10}|I| + \frac{3}{10}|I| + \frac{5}{10}|I| + t_u - 2 \\ &\geq |I| - 2 \end{aligned} \tag{4.2}$$

Combining Equations 4.1 and 4.2, we get:

$$\begin{aligned}|N_Z(S)| + 2 &\geq \frac{6}{5}|S| + \frac{40}{50}|S_u| + \frac{15}{50}|S_l| + 2(t_u - 1) + t_l - (|I| - 2) \\ &\geq \frac{6}{5}|S| \end{aligned} \tag{4.3}$$

Therefore, $V(S, \sigma[1, x])$ is a vertex cover of W_k of size at least $3k^2 + k - 2$, as needed.

To show the $f(k) \leq k + 3$ upper bound, we show that $(W_k, S_s, S_t, 3k^2 + k + 3, 6k^2)$ is a yes-instance by providing an actual reconfiguration sequence (that is not nice):

(1) Add all k beads in L_1. Since S is a vertex cover of W_k, we know that the additional k beads will result in a vertex cover of size $3k^2 + k$.

(2) Add both vertices in $p^u_{1,1}$ and remove both vertices in $p^l_{1,1}$. The removal of both vertices in $p^l_{1,1}$ is possible since we added all their neighbors in L_1 (step (1)) and U_1. The size of a vertex cover reaches $3k^2 + k + 2$ after the additions and then reduces to $3k^2 + k$.

(3) Repeat step (2) for all sequin pairs $p^u_{i,1}$ and $p^l_{1,i}$ for $2 \leq i \leq k$. The size of a vertex cover is again $3k^2 + k$ once step (3) is completed. Step (2) is repeated a total of k times. After every repetition, we have a vertex cover of W_k since all beads in L_1 were added in step (1) and the remaining neighbors of each sequin pair in U_i are added prior to the removals.

(4) Add both vertices in $p^u_{1,2}$ and remove vertex $b^u_{1,2}$.

(5) Add $b^l_{2,1}$ and $b^l_{2,2}$. At this point, the size of a vertex cover is $3k^2 + k + 3$.

(6) Remove both vertices in $p^l_{2,1}$.

(7) Repeat steps (4), (5), and (6) until all beads in L_2 have been added and all sequin pairs removed. When we reach the last sequin pair in L_2, $b^l_{2,1}$ was already added and hence we gain a surplus of one which brings the vertex cover size back to $3k^2 + k$.

(8) Repeat steps (4) to (7) for every remaining necklace in L.

Since every vertex in W_k is touched exactly once, we know that $\ell = 6k^2$. In the course of the described reconfiguration sequence, the maximum size of any vertex cover is $3k^2+k+3$. Hence, $f(k) \leq k+3$. This completes the proof. □

It would be interesting to close the gap on $f(k)$, but the existence of such a value is enough to prove the main theorem of this section.

Theorem 4.3.5. VC-Reach *and* VC-Bound *are* NP*-hard on* 4*-regular graphs.*

Proof. We give a reduction from VC-Comp to VC-Reach where the input graph is restricted to be 4-regular in both cases. For (G, S, k) an instance of VC-Comp, we construct a graph G' by taking the disjoint union of G and W_k. We let $(G', S_s, S_t, 3k^2+k+f(k)-1)$ be an instance of VC-Reach, where $S_s = \{e^u_{i,j} \mid 1 \leq i, j \leq k\} \cup \{p^l_{i,j} \mid 1 \leq i, j \leq k\} \cup S$ and $S_t = \{e^l_{i,j} \mid 1 \leq i, j \leq k\} \cup \{p^u_{i,j} \mid 1 \leq i, j \leq k\} \cup S$ and $f(k)$ is the value whose existence was shown in Lemma 4.3.4. Clearly $|S_s| = |S_t| = 3k^2+k$ and both S_s and S_t are vertex covers of G'. We claim that G has a vertex cover of size $k-1$ if and only if there is a reconfiguration sequence from S_s to S_t.

We know from Lemma 4.3.4 that the reconfiguration of W_k requires at least $f(k)$ available capacity. But $3k^2 + k + f(k) > 3k^2 + k + f(k) - 1$, which violates the maximum allowed capacity. Therefore, if there is a reconfiguration sequence from S_s to S_t, then one of the vertex covers in the sequence must contain at most $3k^2 + k - 1$ vertices. By Proposition 4.3.2, we know that $3k^2$ of those $3k^2 + k - 1$ vertices are needed to cover the edges in $E(W_k)$. Thus, the remaining $k-1$ vertices must be in $V(G)$ and should cover all the edges in $E(G)$. By our construction of G', these $k-1$ vertices correspond to a vertex cover of G.

Similarly, if G has a vertex cover S' such that $|S'| = k-1$, then the following reconfiguration sequence transforms S_s to S_t: add all vertices of S', remove all vertices of S, apply the reconfiguration sequence whose existence was shown in Lemma 4.3.4 to $G'[V(W_k)]$, and finally add back all vertices of S and remove those of S'. The length of this sequence is equal to $6k^2 + 4k - 2$ whenever $S \cap S' = \emptyset$ and is shorter otherwise. □

4.4 Hardness on bipartite graphs

For a graph G, a *crown* [1, 28] is a pair (W, H) satisfying the following properties:

(i) $W \neq \emptyset$ is an independent set of G,

(ii) $N_G(W) = H$, and

(iii) there exists a matching in $G[W \cup H]$ which saturates H.

H is called the *head* of the crown and the *width* of the crown is $|H|$. Crown structures have played a central role in the development of efficient kernelization algorithms for the Vertex Cover problem [1, 28]. We define the closely related notion of (k, d)-constrained crowns and show in the remainder of this section, via a problem of independent interest, that the complexity of finding such structures in a bipartite graph plays an important role in determining the complexity of the bounded reachability problem.

For a graph G, we define a (k, d)*-constrained crown* as a pair (W, H) satisfying all properties of a regular crown with the additional constraints that $|H| \leq k$ and $|W| - |H| \geq d \geq 0$. We are now ready to introduce the (k, d)-Bipartite Constrained Crown problem, or (k, d)-BCC, which is formally defined as follows:

(k, d)-Bipartite Constrained Crown
Input: A bipartite graph G with bipartition (A, B) and two positive integers k and d
Question: Does G have a (k, d)-constrained crown (W, H) such that $W \subseteq A$ and $H \subseteq B$?

Lemma 4.4.1. (k, d)-Bipartite Constrained Crown *parameterized by* $k + d$ *is* W[1]-*hard even when the input graph* G *is* C_4*-free and all vertices in one partition of* G *have degree at most two.*

Proof. We give an FPT reduction from Clique to $(k, \binom{k}{2})$-Bipartite Constrained Crown. For (G, k) an instance of Clique, we let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$.

We first form a bipartite graph G' with bipartition $(X \cup Z, Y)$ and edge set $E_1 \cup E_2$, where vertex sets X and Y contain one vertex for each vertex in $V(G)$ and Z contains one vertex for each edge in $E(G)$. Formally, we set $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, and $Z = \{z_1, \ldots, z_m\}$. The edges in E_1 join each pair of vertices x_i and y_i for $1 \leq i \leq n$ and the edges in E_2 join each vertex z in Z to the two vertices y_i and y_j corresponding to the endpoints of the edge in $E(G)$ to which z corresponds. Since each edge either joins vertices in X and Y or vertices in Y and Z, it is not difficult to see that the vertex sets $X \cup Z$ and Y form a bipartition.

By our construction, G' is C_4-free; vertices in X have degree 1, and since there are no double edges in G (i.e. two edges between the same pair of vertices), no pair of vertices in Y can have more than one common neighbor in Z.

For $(G', k, \binom{k}{2})$ an instance of $(k, \binom{k}{2})$-BCC, $A = X \cup Z$, and $B = Y$, we claim that G has a clique of size k if and only if G' has a $(k, \binom{k}{2})$-constrained crown (W, H) such that $W \subseteq A$ and $H \subseteq B$.

If G has a clique K of size k, we set $H = \{y_i \mid v_i \in V(K)\}$, namely the vertices in Y corresponding to the vertices in the clique. To form W, we choose $\{x_i \mid v_i \in V(K)\} \cup \{z_i \mid e_i \in E(K)\}$, that is, the vertices in X corresponding to the vertices in the clique and the vertices in Z corresponding to the edges in the clique. Clearly H is a subset of size k of B and W is a subset of size $k + \binom{k}{2}$ of A; this implies that $|W| - |H| \geq d = \binom{k}{2}$, as required. To see why $N_{G'}(W) = H$, it suffices to note that every vertex $x_i \in W$ is connected to exactly one vertex $y_i \in H$ and every degree-two vertex $z_i \in W$ corresponds to an edge in K whose endpoints $\{v_i, v_j\}$ must have corresponding vertices in H. Moreover, due to E_1 there is a matching between the vertices of H and the vertices of W in X, and hence a matching in $G'[W \cup H]$ that saturates H.

We now assume that G' has a $(k, \binom{k}{2})$-constrained crown (W, H) such that $W \subseteq X \cup Z$ and $H \subseteq Y$. It suffices to show that $|H|$ must be equal to k, $|W \cap Z|$ must be equal to $\binom{k}{2}$, and hence $|W \cap X|$ must be equal to k; from this we can conclude the vertices in $\{v_i \mid y_i \in H\}$ form a clique of size k in G as $|W \cap Z| = \binom{k}{2}$, requiring that edges exist between each pair of vertices in the set $\{v_i \mid y_i \in H\}$. Moreover, since $|W \cap X| = k$ and $N_{G'}(W) = H$, a matching that saturates H can be easily found by simply picking all edges $\{x_i, y_i\}$ for $y_i \in H$.

To prove the sizes of H and W, we first observe that since $|H| \leq k$, $N_{G'}(W) = H$, and each vertex in Y has exactly one neighbour in X, we know that $|W \cap X| \leq |H| \leq k$. Moreover, since $|W| = |W \cap X| + |W \cap Z|$ and $|W| - |H| \geq \binom{k}{2}$, we know that $|W \cap Z| = |W| - |W \cap X| \geq \binom{k}{2} + |H| - |W \cap X| \geq \binom{k}{2}$. If $|W \cap Z| = \binom{k}{2}$ our proof is complete since, by our construction of G', H is a set of at most k vertices in the original graph G and the subgraph induced by those vertices in G has $\binom{k}{2}$ edges. Hence, $|H|$ must be equal to k. Suppose instead that $|W \cap Z| > \binom{k}{2}$. In this case, since each vertex of Z has degree two, the number of neighbours of $W \cap Z$ in Y is greater than k, violating the assumptions that $N_{G'}(W) = H$ and $|H| \leq k$. □

We can now show the main result of this section:

Theorem 4.4.2. VC-Bound *parameterized by ℓ and restricted to bipartite graphs is* `W[1]`*-hard.*

Proof. We give an FPT reduction from (t,d)-Bipartite Constrained Crown to VC-Bound in bipartite graphs. For (G,t,d) an instance of (t,d)-Bipartite Constrained Crown and $A = \{a_1, \ldots, a_{|A|}\}$ and $B = \{b_1, \ldots, b_{|B|}\}$ the bipartition of G, we form G' with vertex set $X \cup Y \cup U \cup V$ and edge set $E_1 \cup E_2$ such that X and Y correspond to the vertex sets A and B, E_1 connects vertices in X and Y corresponding to vertices in A and B joined by edges in G, and U, V, and E_2 form a bipartite clique $K_{d+t,d+t}$. Formally, $X = \{x_1, \ldots, x_{|A|}\}$, $Y = \{y_1, \ldots, y_{|B|}\}$, $U = \{u_1, \ldots, u_{d+t}\}$, $V = \{v_1, \ldots, v_{d+t}\}$, $E_1 = \{\{x_i, y_j\} \mid \{a_i, b_j\} \in E(G)\}$ and $E_2 = \{\{u_i, v_j\} \mid 1 \leq i \leq d+t, 1 \leq j \leq d+t\}$.

We let $(G', S_s, S_t, k = |A| + d + 2t, \ell = 4d + 6t)$ be an instance of VC-Bound, where $S_s = X \cup U$ and $S_t = X \cup V$. Clearly $|S_s| = |S_t| = |A| + d + t$. We claim that G has a (k,d)-constrained crown (W,H) such that $W \subseteq A$ and $H \subseteq B$ if and only if there is a path of length less than or equal to $4d + 6t$ from S_s to S_t.

If G has such a pair (W,H), we form a reconfiguration sequence of length at most $4d + 6t$ as follows:

1. Add each vertex y_i such that $b_i \in H$. The resulting vertex cover size is $|A|+d+t+|H|$.
2. Remove $d + |H|$ vertices x_i such that $a_i \in W$. The resulting vertex cover size is $|A| + t$.
3. Add each vertex from V. The resulting vertex cover size is $|A| + d + 2t$.
4. Remove each vertex from U. The resulting vertex cover size is $|A| + t$.
5. Add each vertex removed in phase 2. The resulting vertex cover size is $|A|+d+t+|H|$.
6. Remove each vertex added in phase 1. The resulting vertex cover size is $|A| + d + t$.

The length of the sequence follows from the fact that $|H| \leq t$: phases 1 and 6 consist of at most t steps each and phases 2, 3, 4, and 5 of at most $d + t$ steps each. The fact that each set forms a vertex cover is a consequence of the fact that $N_G(W) = H$.

For the converse, we observe that before removing any vertex u_i, $1 \leq i \leq d + t$, from U, we first need to add all $d + t$ vertices from V. Therefore, if there is a path of length at most $4d + 6t$ from S_s to S_t, then we can assume without loss of generality that there exists a node Q (i.e. a vertex cover) along this path such that:

(1) $|Q| \leq |A| + t$ and,

(2) all vertices that were touched in order to reach node Q belong to $X \cup Y$.

In other words, at node Q, the available capacity is greater than or equal to $d + t$ and all edges in $G[U \cup V]$ are still covered by U. We let $Q_{\text{IN}} = Q \setminus S_s$ and $Q_{\text{OUT}} = S_s \setminus Q$. Since $S_s = X \cup U$, $Q_{\text{IN}} \subseteq Y$ and $Q_{\text{OUT}} \subseteq X$. Moreover, since $|Q| = |S_s| + |Q_{\text{IN}}| - |Q_{\text{OUT}}| = |A| + d + t + |Q_{\text{IN}}| - |Q_{\text{OUT}}| \leq |A| + t$, we know that $|Q_{\text{OUT}}| - |Q_{\text{IN}}|$ must be greater than or equal to d. Given that $\ell \leq 4d + 6t$ and we need exactly $2d + 2t$ steps to add all vertices in V and remove all vertices in U, we have $2d + 4t$ remaining steps to allocate elsewhere. Therefore, $|Q_{\text{OUT}}| + |Q_{\text{IN}}| \leq d + 2t$ as $Q_{\text{IN}} \subseteq Y$, $Q_{\text{OUT}} \subseteq X$, and every vertex in $Q_{\text{IN}} \cup Q_{\text{OUT}}$ must be touched at least twice (i.e. added and then removed). Combining those observations, we get:

$$|Q_{\text{OUT}}| + |Q_{\text{IN}}| \leq d + 2t$$
$$|Q_{\text{IN}}| - |Q_{\text{OUT}}| \leq -d$$
$$|Q_{\text{IN}}| \leq t$$

We have just shown that G has a pair $(Q_{\text{OUT}}, Q_{\text{IN}})$ such that $Q_{\text{OUT}} \subseteq X$, $Q_{\text{IN}} \subseteq Y$, $|Q_{\text{IN}}| \leq t$, $|Q_{\text{OUT}}| - |Q_{\text{IN}}| \geq d \geq 0$, and $N_G(Q_{\text{OUT}}) = Q_{\text{IN}}$ as otherwise some edge is not covered. The remaining condition for $(Q_{\text{OUT}}, Q_{\text{IN}})$ to satisfy is for $G[Q_{\text{OUT}} \cup Q_{\text{IN}}]$ to have a matching which saturates Q_{IN}. Hall's Marriage Theorem [74] states that such a saturating matching exists if and only if for every subset P of Q_{IN}, $|P| \leq |N_{G[Q_{\text{OUT}} \cup Q_{\text{IN}}]}(P)|$. By a simple application of Hall's theorem, if no such matching exists then there exists a subgraph Z of $G[Q_{\text{OUT}} \cup Q_{\text{IN}}]$ such that $|V(Z) \cap Q_{\text{OUT}}| < |V(Z) \cap Q_{\text{IN}}|$. By deleting this subgraph from $Q_{\text{OUT}} \cup Q_{\text{IN}}$ we can get a new pair $(Q'_{\text{OUT}}, Q'_{\text{IN}})$ which must satisfy $Q'_{\text{OUT}} \subseteq X$, $Q'_{\text{IN}} \subseteq Y$, $|Q'_{\text{IN}}| \leq t$, $|Q'_{\text{OUT}}| - |Q'_{\text{IN}}| \geq d \geq 0$, and $N_G(Q'_{\text{OUT}}) = Q'_{\text{IN}}$ since we delete more vertices from Q_{IN} than we do from Q_{OUT} and $N_{G[Q_{\text{OUT}} \cup Q_{\text{IN}}]}(V(Z) \cap Q_{\text{IN}}) = V(Z) \cap Q_{\text{OUT}}$. Finally, if $(Q'_{\text{OUT}}, Q'_{\text{IN}})$ does not have a matching which saturates Q'_{IN}, we can repeatedly apply the same rule until we reach a pair which satisfies all the required properties. Since $|Q_{\text{OUT}}| \geq |Q_{\text{IN}}|$, such a pair is guaranteed to exist as otherwise every subset P of Q_{IN} would satisfy $|P| > |N_{G[Q_{\text{OUT}} \cup Q_{\text{IN}}]}(P)|$ and hence $|Q_{\text{OUT}}| < |Q_{\text{IN}}|$, a contradiction. □

4.5 Tractability

There are two different but closely related approaches one can take to show that VC-Reach and VC-Bound are fixed-parameter tractable when parameterized by k. We present one approach in this section and defer the second to the next chapter (Section 5.1), where we show that it applies to other types of problems. For now, we show in Theorem 4.5.2 that both VC-Reach and VC-Bound admit an $\mathcal{O}(k^2)$ kernel and can therefore be solved in $\mathcal{O}^*(k^k)$ time by exhaustive enumeration. Stated differently, Theorem 4.5.2 implies that all but $\mathcal{O}(k^2)$ vertices of the input graph are (strongly) irrelevant (Definition 3.3.9).

There are two classical rules [46] used in kernelization algorithms for the Vertex Cover problem. Given a graph G, the first rule, called the *high-degree rule*, states that any vertex of degree $k+1$ or more must be in any vertex cover of size k or less. From a reconfiguration point of view, such a vertex is frozen and every vertex cover corresponding to a node in $R_{VC}(G, 0, k)$ contains this vertex. A simplified version of the *low-degree rule* states that any vertex of degree zero need not be part of any vertex cover of G. That is, vertices of degree zero are clearly strongly irrelevant for reconfiguration. Combining these two rules with the following proposition is enough to prove Theorem 4.5.2.

Recall that for a graph G and two vertex covers S_s and S_t of G, we partition $V(G)$ into the sets $C_{st} = S_s \cap S_t$ (vertices common to S_s and S_t), $S_{s \setminus t} = S_s \setminus S_t$ (vertices to be removed from S_s in the course of a reconfiguration sequence), $S_{t \setminus s} = S_t \setminus S_s$ (vertices to be added to form S_t), and $O_{st} = V(G) \setminus (S_s \cup S_t)$ (all other vertices).

Proposition 4.5.1. *For a graph G and two vertex covers S_s and S_t of G, $G[S_{s \setminus t} \cup S_{t \setminus s}]$ is bipartite, and there is no edge uv for $u \in S_{s \setminus t} \cup S_{t \setminus s}$ and $v \in O_{st}$.*

Proof. Since $S_{s \setminus t} \cap S_t = \emptyset$ and S_t is a vertex cover of G, each edge of G must have an endpoint in S_t, and hence $G[S_{s \setminus t}]$ must be an independent set. Similar arguments apply to $G[S_{t \setminus s}]$ and to show that there is no edge $uv \in E(G)$ for $u \in S_{s \setminus t} \cup S_{t \setminus s}$ and $v \in O_{st}$. □

Theorem 4.5.2. VC-Reach *and* VC-Bound *parameterized by k admit an $\mathcal{O}(k^2)$ kernel on general graphs.*

Proof. We let (G, S_s, S_t, k, ℓ) be an instance of VC-Bound. Due to Proposition 4.5.1, we know that $G[S_{s \setminus t} \cup S_{t \setminus s}]$ is a bipartite graph and there are no edges between vertices in O_{st} and vertices in $S_{s \setminus t} \cup S_{t \setminus s}$. Figure 4.8 illustrates a hypothetical graph G with vertices in $S_{s \setminus t}$ shown in black, vertices in $S_{t \setminus s}$ shown in dark gray, vertices in C_{st} shown in light gray, and vertices in O_{st} shown in white.

As $G[O_{st}]$ is an independent set, all the neighbors of a vertex $v \in O_{st}$ must be in C_{st}. If there is a vertex $v \in O_{st}$ with no neighbors in C_{st} then such a vertex has degree zero in G and we can therefore delete it, as it is clearly strongly irrelevant. Hence, we assume that every vertex in O_{st} has at least one neighbor in C_{st}. Because vertices of degree $k+1$

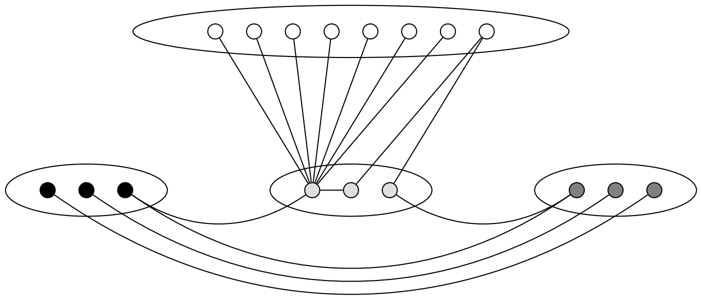

Figure 4.8: An instance of VC-Bound.

or more in G are frozen, all such vertices must be in C_{st}. For a vertex $v \in O_{st}$, if every neighbor of v in C_{st} is frozen then v is also strongly irrelevant; any reconfiguration sequence which adds v cannot remove any neighbor of v. Again, we assume that all such vertices are deleted.

Putting it all together, we know that the number of vertices in the graph (after deleting all strongly irrelevant vertices) is at most $2k + k^2$, as $|S_s| + |S_t| \leq 2k$ and the number of vertices in O_{st} with at least one neighbor in C_{st} of degree k or less is at most k^2. $\square$

4.5.1 Even-hole-free and cactus graphs

In this section, we present a characterization of instances of VC-Reach and VC-Bound solvable in time polynomial in the size of the input graph, and apply this characterization to even-hole-free graphs and cactus graphs (Section 2.1). Interestingly, as shown in Theorem 4.4.2, excluding odd cycles does not seem to make the VC-Bound problem any easier, whereas excluding (induced or non-induced) even cycles puts both VC-Reach and VC-Bound in P. We show that we can in fact obtain polynomial-time algorithms whenever the graph induced by the symmetric difference of two vertex covers has some "nice" properties. We show that for even-hole-free graphs the symmetric difference of any two vertex covers of the graph induces a forest. The structure is slightly more complex for cactus graphs. Moreover, in both cases, the number of vertices we need to consider outside of the symmetric difference is bounded by a constant. We note that a similar polynomial-time algorithm for even-hole-free graphs was also recently, and independently, obtained by Kamiński et al. for solving several variants of the IS-Bound problem [91]. Unless stated otherwise, reconfiguration sequences are represented as ordered sequences of vertex covers or nodes in the reconfiguration graph.

Definition 4.5.3. *Given two vertex covers of G, A and B, a reconfiguration sequence β from A to some vertex cover A' is a c-*bounded prefix *of a reconfiguration sequence α from A to B, if and only if all of the following conditions hold:*

(1) $|A'| \leq |A|$.

(2) For every node A'' in β, $|A''| \leq |A| + c$.

(3) For every node A'' in β, A'' is obtained from its predecessor by either the removal or the addition of a single vertex in the symmetric difference of the predecessor and B, which implies that no vertex is touched more than once in the course of β.

We write $A \xleftrightarrow{c, B} A'$ when such a c-bounded prefix exists.

Proposition 4.5.4. *Given two vertex covers A and B of G, if G has a vertex cover S such that $A \xleftrightarrow{c, B} S$, then $A \xleftrightarrow{d, B} S$ for all $d > c$.*

Lemma 4.5.5. *Given two vertex covers S_s and S_t of G and two positive integers k and c such that $|S_s|, |S_t| \leq k$, a reconfiguration sequence α of length $|S_s \Delta S_t|$ from S_s to S_t exists if:*

(1) $|S_s| \leq k - c$,

(2) $|S_t| \leq k - c$, *and*

(3) For any two vertex covers A and B of G, $A \neq B$ and $|A|, |B| \leq k - c$, either there exists A' such that $A \xleftrightarrow{c, B} A'$ or there exists B' such that $B \xleftrightarrow{c, A} B'$, where A' and B' are vertex covers of G.

Moreover, if c-bounded prefixes (i.e. if A' or B') can be found in time polynomial in n, then α can be found in time polynomial in n.

Proof. We prove the lemma by induction on $|S_s \Delta S_t|$. When $|S_s \Delta S_t| = 0$, S_s is equal to S_t, and the claim holds trivially since $|\alpha| = 0$.

When $|S_s \Delta S_t| > 0$, and since $S_s \neq S_t$ and $|S_s|, |S_t| \leq k - c$, we know that either there exists S'_s such that $S_s \xleftrightarrow{c, S_t} S'_s$ or there exists S'_t such that $S_t \xleftrightarrow{c, S_s} S'_t$, where S'_s and S'_t are vertex covers of G. Without loss of generality, we assume $S_s \xleftrightarrow{c, S_t} S'_s$ and let β denote the c-bounded prefix from S_s to S'_s. From Definition 4.5.3, we know that the size of every node in β is no greater than $|S_s| + c \leq k$. Therefore, the maximum allowed capacity constraint is never violated.

Since $|S'_s| \leq |S_s|$ and $|S'_s \Delta S_t| \leq |S_s \Delta S_t|$, as we only touch vertices in $S_s \Delta S_t$ to reach S'_s (Definition 4.5.3), by the induction hypothesis there exists a reconfiguration sequence from S'_s to S_t whose length is $|S'_s \Delta S_t|$. By appending the reconfiguration sequence from S'_s to S_t to the reconfiguration sequence from S_s to S'_s, we obtain a reconfiguration sequence α from S_s to S_t.

To show that $|\alpha| = |S_s \Delta S_t|$, it suffices to show that $|\beta| + |S'_s \Delta S_t| = |S_s \Delta S_t|$. We know that no vertex is touched more than once in β and every touched vertex belongs to $S_s \Delta S_t$ (Definition 4.5.3). We let $H \subseteq S_s \Delta S_t$ denote the set of touched vertices in β and we subdivide H into two sets $H_s = H \cap S_s = H \cap S_{s \setminus t}$ and $H_t = H \cap S_t = H \cap S_{t \setminus a}$. It follows that $|\beta| = |H_s| + |H_t|$ and $|S'_s \Delta S_t| = |S_{s \setminus t} \setminus H_s| + |S_{t \setminus s} \setminus H_t|$. Therefore, $|\beta| + |S'_s \Delta S_t| = |H_s| + |H_t| + |S_{s \setminus t} \setminus H_s| + |S_{t \setminus s} \setminus H_t| = |S_{s \setminus t}| + |S_{t \setminus s}| = |S_s \Delta S_t|$, as needed.

When c-bounded prefixes can be found in time polynomial in n, the proof gives an algorithm for constructing the full reconfiguration sequence from S_s to S_t in time polynomial in n. □

Theorem 4.5.6. VC-Reach *and* VC-Bound *restricted to trees can be solved in time polynomial in n.*

Proof. We let (G, S_s, S_t, k, ℓ) be an instance of VC-Bound. The proof proceeds in two stages. We start by showing that when G is a tree and S_s and S_t are of size at most $k - 1$, we can always find 1-bounded prefixes $S_s \xleftrightarrow{1, S_t} S'_s$ or $S_t \xleftrightarrow{1, S_s} S'_t$ in time polynomial in n. Therefore, we can apply Lemma 4.5.5 with $c = 1$ to find a reconfiguration sequence of length $|S_s \Delta S_t|$ from S_s to S_t in time polynomial in n. In the second part of the proof, we show how to handle the remaining cases where S_s, S_t, or both S_s and S_t are of size greater than $k - 1$.

First, we note that every forest either has a degree-zero or a degree-one vertex. Hence, trees and forests are 1-degenerate graphs. Since G is a tree, $G[S_{s \setminus t} \cup S_{t \setminus s}]$ is a forest and is therefore 1-degenerate. To find 1-bounded prefixes in $G[S_{s \setminus t} \cup S_{t \setminus s}]$, it is enough to find a vertex v of degree at most one, which can clearly be done in time polynomial in n. The existence of v guarantees the existence of a 1-bounded prefix from either S_s to some vertex cover S'_s or from S_t to some vertex cover S'_t. When $v \in S_{s \setminus t}$ and $|N_{G[S_{s \setminus t} \cup S_{t \setminus s}]}(v)| = 0$, we have $S_s \xleftrightarrow{0, S_t} S'_s$ since S'_s is obtained from S_s by simply removing v. When $v \in S_{s \setminus t}$ and $|N_{G[S_{s \setminus t} \cup S_{t \setminus s}]}(v)| = 1$, we have $S_s \xleftrightarrow{1, S_t} S'_s$ since S'_s is obtained from S_s by first adding the unique neighbor of v and then removing v. Similar arguments hold when $v \in S_{t \setminus s}$.

Therefore, combining Lemma 4.5.5 and the fact that $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is 1-degenerate, we know that if $|S_s| \leq k-1$ and $|S_t| \leq k-1$, a reconfiguration sequence of length $|S_{s\setminus t} \cup S_{t\setminus s}|$ from S_s to S_t can be found in time polynomial in n. Furthermore, since the length of a reconfiguration sequence can never be less than $|S_{s\setminus t} \cup S_{t\setminus s}|$, the reconfiguration sequence given by Lemma 4.5.5 is a shortest path from S_s to S_t in the reconfiguration graph.

When S_s (respectively, S_t) has size k and is minimal, then we have a no-instance since neither removing nor adding a vertex results in a vertex cover of size k, and hence S_s (respectively, S_t) will be an isolated node in the reconfiguration graph, with no path to S_t (respectively, S_s).

Otherwise, there always exists a reconfiguration sequence from S_s to S_t, since S_s and S_t can be reconfigured to solutions S'_s and S'_t, respectively, of size less than k, to which Lemma 4.5.5 can be applied. The only reconfiguration steps from S_s (or S_t) of size k are to subsets of S_s of size $k-1$ (or to subsets of S_t of size $k-1$); the reconfiguration sequence obtained from Lemma 4.5.5 is thus a shortest path. Therefore, we can obtain a shortest path from S_s to S_t through a careful selection of S'_s and S'_t. There are two cases to consider:

Case (1): $|S_s| = k$, $|S_t| = k$, S_s is non-minimal, and S_t is non-minimal. When both S_s and S_t are of size k and are non-minimal, then each must contain at least one removable vertex. Hence, by removing such vertices, we can transform S_s and S_t into vertex covers S'_s and S'_t, respectively, of size $k-1$. We let u and v be removable vertices in S_s and S_t respectively, and we set $S'_s = S_s \setminus \{u\}$ and $S'_t = S_t \setminus \{v\}$. Note that there are at most k^2 pairs of removable vertices in S_s and S_t and we can exhaustively try all pairs to choose one that minimizes the length of a reconfiguration sequence.

1. If $u \in S_{s\setminus t}$ and $v \in S_{t\setminus s}$, then the length of a shortest reconfiguration sequence from S'_s to S'_t will be $|S'_s \Delta S'_t| = |S_s \Delta S_t| - 2$. Therefore, accounting for the two additional removals, the length of a shortest path from S_s to S_t will be equal to $|S_s \Delta S_t|$.

2. If $u \in S_{s\setminus t}$ and $v \in C_{st}$, then the length of a shortest reconfiguration sequence from S'_s to S'_t will be $|S'_s \Delta S'_t| = |S_s \Delta S_t| - 1$. Since v is in C_{st}, it must be removed and added back. Therefore, the length of a shortest path from S_s to S_t will be equal to $|S_s \Delta S_t| + 2$. The same is true when $u \in C_{st}$ and $v \in S_{t\setminus s}$ or when $u = v$ and $u \in C_{st}$.

3. Otherwise, when $u \in C_{st}$, $v \in C_{st}$, and $u \neq v$, the length of a shortest path from S_s to S_t will be $|S_s \Delta S_t| + 4$ since we have to touch two vertices in C_{st} (i.e. two extra additions and two extra removals).

Case (2): $|S_s| = k$, $|S_t| = k-1$, and S_s is non-minimal. Since $|S_t| = k-1$, we only need to reduce the size of S_s to $k-1$ in order to apply Lemma 4.5.5. Since S_s is non-minimal, it must contain at least one removable vertex. We let u be a removable vertex in S_s and we set $S'_s = S_s \setminus \{u\}$.

1. If $u \in S_{s\setminus t}$, then the length of a shortest reconfiguration sequence from S'_s to S_t will be $|S'_s \Delta S_t| = |S_s \Delta S_t| - 1$. Therefore, accounting for the additional removal, the length of a shortest path from S_s to S_t will be equal to $|S_s \Delta S_t|$.

2. If $u \in C_{st}$, then the length of a shortest reconfiguration sequence from S'_s to S_t will be $|S'_s \Delta S_t| = |S_s \Delta S_t|$. Since v is in C_{st}, it must be removed and added back. Therefore, the length of a shortest path from S_s to S_t will be equal to $|S_s \Delta S_t| + 2$.

Similar arguments hold for the symmetric case where $|S_s| = k-1$, $|S_t| = k$, and S_t is non-minimal. As there are at most k^2 pairs of removable vertices in S_s and S_t to check for Case (1), we can exhaustively try all pairs and choose one that minimizes the length of a reconfiguration sequence. Similarly, there are at most k removable vertices to check in Case (2). Consequently, VC-Reach and VC-Bound restricted to trees can be solved in time polynomial in n. □

Recall that a cactus graph G is a graph in which every edge belongs to at most one cycle. We let $\mathcal{C}(G)$ denote the set of all cycles in G. The following proposition is a consequence of the fact that for any cactus graph G, we can construct a maximal matching of G containing at least one edge from each cycle in $\mathcal{C}(G)$.

Proposition 4.5.7. *For a cactus graph G, the number of cycles in G is bounded above by the size of any maximal matching $\mathcal{M}_G$, i.e. $|\mathcal{C}(G)| \leq |\mathcal{M}_G|$.*

The next proposition is a consequence of the fact that for any cactus graph G, we can obtain a spanning tree of G by removing a single edge from every cycle in G.

Proposition 4.5.8. *For a cactus graph G and T_G a spanning tree of G, the total number of edges in G is equal to the number of edges in T_G plus the total number of cycles in G, i.e. $|E(G)| = |E(T_G)| + |\mathcal{C}(G)| = |V(T_G)| - 1 + |\mathcal{C}(G)|$.*

Any graph with no even cycles is a cactus graph (but the converse is not always true). For a graph G with no even cycles and any two vertex covers, S_s and S_t, of G, we know that $G[S_{s\setminus t} \cup S_{t\setminus s}]$ must be a forest, i.e. a bipartite graph with no even cycles (Proposition 4.5.1).

Corollary 4.5.9 follows from the fact that in the proof of Theorem 4.5.6, the fact that G is a tree is used only to determine that $G[S_{s\setminus t} \cup S_{t\setminus s}]$ must be a forest. Therefore, using the same proof as in Theorem 4.5.6, we get:

Corollary 4.5.9. VC-Reach *and* VC-Bound *restricted to even-hole-free graphs can be solved in time polynomial in n.*

The goal now is to generalize Corollary 4.5.9 to all cactus graphs. To do so, we first show, in Lemmas 4.5.10 and 4.5.11, that the third condition of Lemma 4.5.5 is satisfied for cactus graphs with $c = 2$. In Lemma 4.5.12, we show how 2-bounded prefixes can be found in time polynomial in n, which leads to Theorem 4.5.13.

Lemma 4.5.10. *Given two vertex covers S_s and S_t of G, there exists a vertex cover S'_s (or S'_t) of G such that $S_s \xleftrightarrow{2, S_t} S'_s$ (or $S_t \xleftrightarrow{2, S_s} S'_t$) if one of the following conditions holds:*

(1) $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has a vertex $v \in S_{s\setminus t}$ ($v \in S_{t\setminus s}$) such that $|N_{G[S_{s\setminus t} \cup S_{t\setminus s}]}(v)| \leq 1$, or

(2) there exists a cycle Y in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ such that all vertices in $Y \cap S_{s\setminus t}$ ($Y \cap S_{t\setminus s}$) have degree exactly two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$.

Moreover, both conditions can be checked in time polynomial in n and when one of them is true the corresponding 2-bounded prefix can be found in time polynomial in n.

Proof. First, we note that checking for condition (1) can be accomplished in time polynomial in n by inspecting the degree of every vertex in $G[S_{s\setminus t} \cup S_{t\setminus s}]$. The total number of cycles satisfying condition (2) is linear in the number of degree-two vertices in $G[S_{s\setminus t} \cup S_{t\setminus s}]$. Therefore, we can check for condition (2) in time polynomial in n by a simple breadth-first search starting from every degree-two vertex in $G[S_{s\setminus t} \cup S_{t\setminus s}]$.

If $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has a vertex $v \in S_{s\setminus t}$ of degree zero, we let S'_s denote the vertex cover obtained by removing v from S_s. It is easy to see that the reconfiguration sequence from S_s to S'_s is a 0-bounded prefix and can be found in time polynomial in n.

Similarly, if $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has a vertex $v \in S_{s\setminus t}$ of degree one, we let S'_s denote the node obtained by the addition of the single vertex in $N_{G[S_{s\setminus t} \cup S_{t\setminus s}]}(v)$ followed by the removal of v. The reconfiguration sequence from S_s to S'_s is a 1-bounded prefix and can be found in time polynomial in n.

For the second case, we let Y be a cycle in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ and we partition the vertices of the cycle into two sets; $Y_S = Y \cap S_{s\setminus t}$ and $Y_T = Y \cap S_{t\setminus s}$. Since $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is bipartite, we know that $|Y_S| = |Y_T|$. Since all vertices in Y_S have degree exactly two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$, it follows that $N_{G[S_{s\setminus t} \cup S_{t\setminus s}]}(Y_S) \subseteq Y_T$. Therefore, a reconfiguration sequence from S_s to some vertex cover S'_s that adds all vertices in Y_T (one by one), and then removes all vertices in Y_S (one by one) will satisfy conditions (1), (3), and (4) from Definition 4.5.3 for any value of c. For $c = 2$, such a sequence will not satisfy condition (2) if the cycle has at least six vertices (i.e. $|Y_T| \geq 3$). However, using the fact that every vertex in Y_S has degree exactly two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$, we can find a reconfiguration sequence from S_s to S'_s in which no vertex cover has size greater than $|S_s|+2$. To do so, we restrict our attention to $G[Y_S \cup Y_T]$. Since Y is an even cycle, we can label all the vertices of Y in clockwise order from 0 to $|Y| - 1$ such that all vertices in Y_S receive even labels. The reconfiguration sequence from S_s to S'_s starts by adding the two vertices labeled 1 and $|Y| - 1$. After doing so, the vertex labeled 0 is removed. Next, to remove the vertex labeled 2, we only need to add the vertex labeled 3. The same process is repeated for all vertices with even labels up to $|Y| - 4$. Finally, when we reach the vertex labeled $|Y| - 2$, both of its neighbors will have already been added and we can simply remove it. Hence, we have a 2-bounded prefix from S_s to S'_s and it is not hard to see that finding this reconfiguration sequence can be accomplished in time polynomial in n.

When the appropriate assumptions hold, we can show the symmetric case $S_t \xleftrightarrow{2,\, S_s} S'_t$ using similar arguments. $\square$

Lemma 4.5.11. *If G is a cactus graph and S_s and S_t are two vertex covers of G, then there exists a vertex cover S'_s (or S'_t) of G such that $S_s \xleftrightarrow{2,\, S_t} S'_s$ (or $S_t \xleftrightarrow{2,\, S_s} S'_t$).*

Proof. We assume that $|S_{s\setminus t}| \geq |S_{t\setminus s}|$, as we can swap the roles of S_s and S_t whenever $|S_{s\setminus t}| < |S_{t\setminus s}|$. We observe that every connected component of $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is a cactus graph since every induced subgraph of a cactus graph is also a cactus graph. Since we assume $|S_{s\setminus t}| \geq |S_{t\setminus s}|$, at least one connected component X of $G[S_{s\setminus t} \cup S_{t\setminus s}]$ must satisfy $|V(X) \cap S_{s\setminus t}| \geq |V(X) \cap S_{t\setminus s}|$.

To prove the lemma, we show that if neither condition of Lemma 4.5.10 applies to X, it must be the case that $|V(X) \cap S_{s\setminus t}| < |V(X) \cap S_{t\setminus s}|$, contradicting our assumption. To simplify notation, we assume without loss of generality that $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is connected, as we can otherwise set $G[S_{s\setminus t} \cup S_{t\setminus s}] = X$. The proof proceeds in two steps. First, we show that if condition (1) of Lemma 4.5.10 is not satisfied, then $G[S_{s\setminus t} \cup S_{t\setminus s}]$ must have at least one vertex $u \in S_{s\setminus t}$ of degree at most two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$. In the second step, we show that if both conditions (1) and (2) of Lemma 4.5.10 are not satisfied, then $|S_{s\setminus t}| < |S_{t\setminus s}|$, which completes the proof by contradiction.

Since $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is a cactus graph, we can apply Propositions 4.5.7 and 4.5.8 to get:

$$\begin{aligned}|E(G[S_{s\setminus t} \cup S_{t\setminus s}])| &= |S_{s\setminus t}| + |S_{t\setminus s}| - 1 + |\mathcal{C}(G[S_{s\setminus t} \cup S_{t\setminus s}])| \\ &\leq |S_{s\setminus t}| + |S_{t\setminus s}| - 1 + |\mathcal{M}_{G[S_{s\setminus t} \cup S_{t\setminus s}]}| \end{aligned} \tag{4.4}$$

Moreover, since $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is bipartite (Proposition 4.5.1), the size of a maximum matching in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is less than or equal to $min(|S_{s\setminus t}|, |S_{t\setminus s}|)$. Therefore:

$$|\mathcal{C}(G[S_{s\setminus t} \cup S_{t\setminus s}])| \leq |\mathcal{M}_{G[S_{s\setminus t} \cup S_{t\setminus s}]}| \leq |S_{s\setminus t}|. \tag{4.5}$$

Combining (4.4) and (4.5), we get:

$$\begin{aligned}|E(G[S_{s\setminus t} \cup S_{t\setminus s}])| &= |S_{s\setminus t}| + |S_{t\setminus s}| - 1 + \mathcal{C}(G[S_{s\setminus t} \cup S_{t\setminus s}]) \\ &\leq 2|S_{s\setminus t}| + |S_{t\setminus s}| - 1. \end{aligned} \tag{4.6}$$

If the minimum degree in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ of any vertex in $S_{s\setminus t}$ is three or more, then $3|S_{s\setminus t}| \leq |E(G[S_{s\setminus t} \cup S_{t\setminus s}])| \leq 2|S_{s\setminus t}| + |S_{t\setminus s}| - 1$ and thus $|S_{s\setminus t}| \leq |S_{t\setminus s}| - 1$, contradicting our assumption that $|S_{s\setminus t}| \geq |S_{t\setminus s}|$. Hence, $G[S_{s\setminus t} \cup S_{t\setminus s}]$ must have at least one vertex of degree two in $S_{s\setminus t}$.

Next, we show that if $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has no vertex $v \in S_{s\setminus t}$ such that $|N_{G[S_{s\setminus t} \cup S_{t\setminus s}]}(v)| \leq 1$ and no cycle Y such that all vertices in $Y \cap S_{s\setminus t}$ have degree exactly two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$, then $|S_{s\setminus t}| < |S_{t\setminus s}|$. We let S^x denote the set of vertices in $S_{s\setminus t}$ having degree x in $G[S_{s\setminus t} \cup S_{t\setminus s}]$. Since $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has no vertex $v \in S_{s\setminus t}$ such that $|N_{G[S_{s\setminus t} \cup S_{t\setminus s}]}(v)| \leq 1$, we know that S^2 cannot be empty. In addition, since there is no cycle Y in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ such that all vertices in $Y \cap S_{s\setminus t}$ have degree exactly two in $G[S_{s\setminus t} \cup S_{t\setminus s}]$, any cycle involving a vertex in S^2 must also include a vertex from $\bigcup_{i\geq 3} S^i$. It follows that $\bigcup_{i\geq 3} S^i$ is a feedback vertex set of $G[S_{s\setminus t} \cup S_{t\setminus s}]$ and $G[S^2 \cup S_{t\setminus s}]$ is a forest.

We let m_s denote the maximum degree in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ of any vertex in $S_{s\setminus t}$. Since each edge in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has one endpoint in $S_{s\setminus t}$,

$$\sum_{i=2}^{m_s} i|S^i| \leq |E(G[S_{s\setminus t} \cup S_{t\setminus s}])| \tag{4.7}$$

and since each vertex in $S_{s\setminus t}$ is in some S^i, and using (4.4), we can rewrite (4.7) as

$$\sum_{i=2}^{m_s} i|S^i| \leq \left(\sum_{i=2}^{m_s} |S^i|\right) + |S_{t\setminus s}| - 1 + |\mathcal{C}(G[S_{s\setminus t} \cup S_{t\setminus s}])|. \tag{4.8}$$

To bound $|\mathcal{C}(G[S_{s\setminus t} \cup S_{t\setminus s}])|$, we note that since no edge can belong to more than one cycle in a cactus graph, any vertex $v \in S^x$ can be involved in at most $\lfloor \frac{x}{2} \rfloor$ cycles. Combining this observation with the fact that any cycle involving a vertex in S^2 must also include a vertex from $\bigcup_{i\geq 3} S^i$, we have:

$$\begin{aligned}\sum_{i=2}^{m_s} i|S^i| &\leq \left(\sum_{i=2}^{m_s} |S^i|\right) + |S_{t\setminus s}| - 1 + \left(\sum_{i=3}^{m_s} \lfloor \frac{i}{2} \rfloor |S^i|\right) \\ &\leq |S^2| + \left(\sum_{i=3}^{m_s} (1 + \lfloor \frac{i}{2} \rfloor)|S^i|\right) + |S_{t\setminus s}| - 1 \end{aligned} \tag{4.9}$$

Finally, by rewriting $\sum_{i=2}^{m_s} i|S^i|$ as $2|S^2| + \sum_{i=3}^{m_s} i|S^i|$ and given that $i - (1 + \lfloor \frac{i}{2} \rfloor) \geq 1$ for $i \geq 3$, we obtain the desired bound:

$$\begin{aligned} 2|S^2| + \sum_{i=3}^{m_s} i|S^i| &\leq |S^2| + \left(\sum_{i=3}^{m_s} (1 + \lfloor \frac{i}{2} \rfloor)|S^i|\right) + |S_{t\setminus s}| - 1 \\ |S^2| + \sum_{i=3}^{m_s} i|S^i| &\leq \left(\sum_{i=3}^{m_s} (1 + \lfloor \frac{i}{2} \rfloor)|S^i|\right) + |S_{t\setminus s}| - 1 \\ |S^2| + \sum_{i=3}^{m_s} (i - (1 + \lfloor \frac{i}{2} \rfloor))|S^i| &\leq |S_{t\setminus s}| - 1 \\ |S_{s\setminus t}| = \sum_{i=2}^{m_s} |S^i| &\leq |S_{t\setminus s}| - 1 \end{aligned} \tag{4.10}$$

This completes the proof. □

Lemma 4.5.12. *If G is a cactus graph and S_s and S_t are vertex covers of G, then finding a 2-bounded prefix from S_s to a vertex cover S'_s (or from S_t to a vertex cover S'_t) of G can be accomplished in time polynomial in n.*

Proof. To find a 2-bounded prefix from S_s to a vertex cover S'_s (or from S_t to a vertex cover S'_t) we simply need to satisfy one of the conditions of Lemma 4.5.10, which can both be checked in time polynomial in n. Since $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is a cactus graph, we know from Lemma 4.5.11 that one of them must be true. □

Theorem 4.5.13. VC-Reach *and* VC-Bound *restricted to cactus graphs can be solved in time polynomial in n.*

Proof. From Lemma 4.5.11, we know that for any cactus graph G and two vertex covers S_s and S_t of G, then either $S_s \xleftrightarrow{2, S_t} S'_s$ or $S_t \xleftrightarrow{2, S_s} S'_t$, where S'_s and S'_t are some vertex covers of G. In addition, Lemma 4.5.12 shows that such 2-bounded prefixes can be found in time polynomial in n. By combining these facts, we can now apply Lemma 4.5.5. That is, if $|S_s| \leq k-2$ and $|S_t| \leq k-2$, a reconfiguration sequence of length $|S_{s\setminus t}| + |S_{t\setminus s}|$ from S_s to S_t can be found in time polynomial in n.

When S_s (or S_t) has size k and is minimal, then we have a no-instance since neither removing nor adding a vertex results in a vertex cover of size k, and hence S_s (or S_t) will be an isolated node in the reconfiguration graph, with no path to S_t (or S_s).

The remaining cases to consider are listed in Table 4.1. Some of these cases are symmetric since the roles of S_s and S_t can be interchanged.

Table 4.1: Case Analysis. A ✓denotes a yes-instance and a X denotes a no-instance.

		S_t non-minimal			S_t minimal		
		$k-2$	$k-1$	k	$k-2$	$k-1$	k
S_s non-minimal	$k-2$	✓	(1)	(5)	✓	(3)	X
	$k-1$	(1)	(1)	(5)	(3)	(3)	X
	k	(5)	(5)	(5)	(4)	(4)	X
S_s minimal	$k-2$	✓	(3)	(4)	✓	(2)	X
	$k-1$	(3)	(3)	(4)	(2)	(2)	X
	k	X	X	X	X	X	X

Case (1): $|S_s| = k-1$, $|S_t| = k-1$, S_s is non-minimal, and S_t is non-minimal. When both S_s and S_t are of size $k-1$ and are non-minimal, then each must contain at least one removable vertex. Hence, by removing such vertices, we can transform S_s and S_t into vertex covers S'_s and S'_t, respectively, of size $k-2$. Finding a reconfiguration sequence of shortest length from S'_s to S'_t can be accomplished in time polynomial in n by Lemma 4.5.5. To guarantee a shortest path from S_s to S_t, we have to carefully select S'_s and S'_t. We let u and v be removable vertices in S_s and S_t respectively, and we set $S'_s = S_s \setminus \{u\}$ and $S'_t = S_t \setminus \{v\}$.

1. If $u \in S_{s\setminus t}$ and $v \in S_{t\setminus s}$, then the length of a shortest reconfiguration sequence from S'_s to S'_t will be $|S'_{s\setminus t}| + |S'_{t\setminus s}| = |S_{s\setminus t}| + |S_{t\setminus s}| - 2$. Therefore, accounting for the two additional removals, the length of a shortest path from S_s to S_t will be equal to $|S_{s\setminus t}| + |S_{t\setminus s}|$.

2. If $u \in S_{s\setminus t}$ and $v \in C_{st}$, then the length of a shortest reconfiguration sequence from S'_s to S'_t will be $|S'_{s\setminus t}|+|S'_{t\setminus s}| = |S_{s\setminus t}|+|S_{t\setminus s}|-1$. Since v is in C_{st}, it must be removed and added back. Therefore, the length of a shortest path from S_s to S_t will be equal to $|S_{s\setminus t}| + |S_{t\setminus s}| + 2$. The same is true when $u \in C_{st}$ and $v \in S_{t\setminus s}$ or when $u = v$ and $u \in C_{st}$.

3. Otherwise, when $u \in C_{st}$, $v \in C_{st}$, and $u \neq v$, the length of a shortest path from S_s to S_t will be $|S_{s\setminus t}| + |S_{t\setminus s}| + 4$ since we have to touch two vertices in C_{st} (i.e. two extra additions and two extra removals).

As there are at most $(k-1)^2$ pairs of removable vertices in S_s and S_t to check for Case (1), we can exhaustively try all pairs and choose one that minimizes the length of a reconfiguration sequence. Similar arguments can be applied when one of S_s or S_t is of size $k-2$, in which case we only need to check at most $k-1$ removable vertices.

Case (2): $|S_s| = k-1$, $|S_t| = k-1$, S_s is minimal, and S_t is minimal. If both S_s and S_t are minimal, of size $k-1$, and the minimum degree in $G[S_{s\setminus t} \cup S_{t\setminus s}]$ is two, then we have a no-instance; any removal will require at least two additions, therefore violating the maximum allowed capacity constraint. If $G[S_{s\setminus t} \cup S_{t\setminus s}]$ has a vertex $u \in S_{s\setminus t}$ such that $N_{G[S_{s\setminus t}\cup S_{t\setminus s}]}(u) = \{v\}$, we let S'_s be the vertex cover obtained by adding v and removing u as well as all vertices in $N_{G[S_{s\setminus t}\cup S_{t\setminus s}]}(v)$ that become removable after the addition of v. Since S_t is also minimal, we can apply the same transformation starting from S_t to get some vertex cover S'_t. These transformations correspond to 1-bounded prefixes. If both S'_s and S'_t are still minimal and of size $k-1$, we can exhaustively repeat this process until we either find a reconfiguration from S_s to S_t of length $|S_{s\setminus t}| + |S_{t\setminus s}|$, reach a state similar to Case (1), or determine a no-instance (i.e. minimum degree two in $G[S'_s \Delta S'_t]$). To see why Case (2) can be handled in time polynomial in n, we note that the described process is repeated at most $|S_{s\setminus t}| + |S_{t\setminus s}|$ times. At every iteration, we simply inspect the neighborhood of every vertex in the graph induced by the symmetric difference of the corresponding vertex covers. Finally, when one of S_s or S_t is of size $k-1$ and the other of size $k-2$, we only need to apply the transformations described above starting from the vertex cover of size $k-1$.

Case (3): $|S_s| = |S_t| = k-1$, S_s (S_t) is minimal, and S_t (S_s) is non-minimal. Case (3) can be handled by combining the arguments from Cases (1) and (2); we apply 1-bounded prefixes to the minimal vertex cover (Case (2)) and select a removable vertex from the non-minimal vertex cover which minimizes the total number of reconfiguration steps (Case (1)). Since $|S_s| = |S_t| = k-1$, we need to remove at most one vertex from S_s and one from S_t to obtain vertex covers of size $k-2$ to which we can apply Lemma 4.5.5 to obtain a reconfiguration sequence of shortest possible length. Hence, the length of a reconfiguration sequence from S_s to S_t will be at most $|S_{s\setminus t}| + |S_{t\setminus s}| + 4$, which occurs whenever we have to touch two vertices in C_{st}. Whenever one of S_s or S_t is of size $k-2$, we only need to apply the described arguments to the vertex cover of size $k-1$.

Case (4): $|S_s| = k$, $|S_t| = k-1$, S_s is non-minimal, and S_t is minimal. Case (4) can be broken down into at most k instances which can be solved as in Cases (2) and (3). In each instance, we let $S'_s = S_s \setminus \{v\}$ where v is a removable vertex in S_s. If S'_s is minimal, we apply Case (2). Otherwise, we apply Case (3). Since $|S_s| = k$ and $|S_t| = k-1$, we need to remove at most 2 vertices from S_s and 1 from S_t to obtain vertex covers of size $k-2$ to which we can apply Lemma 4.5.5 to obtain a reconfiguration sequence of shortest possible length. Hence, the length of a reconfiguration sequence from S_s to S_t will be at most $|S_{s\setminus t}| + |S_{t\setminus s}| + 6$, which occurs whenever we have to touch three vertices in C_{st}. Similarly to previous cases, whenever one of S_s or S_t is of size $k-2$, we only need to apply the described arguments to the vertex cover of size k.

Case (5): $|S_s| = k$, $|S_t| = k$, S_s is non-minimal, and S_t is non-minimal. Similarly, Case (5) can be broken down into at most k instances which can be solved as in Case (4). In each instance, we let $S'_t = S_t \setminus \{v\}$, where v is a removable vertex in S_t. The length of a reconfiguration sequence will be at most $|S_{s\setminus t}|+|S_{t\setminus s}|+8$, which occurs whenever we have to touch four vertices in C_{st}. When one of S_s or S_t is of size $k-2$ or $k-1$ and the other is of size k, Case (5) can be broken down into at most k instances which can be solved as in Case (3).

It is not hard to see that in all cases, VC-Reach and VC-Bound restricted to cactus graphs can be solved in time polynomial in n. This completes the proof. □

We conclude this section with a few remarks. As we have seen, VC-Reach and VC-Bound are solvable in polynomial time when we can guarantee the existence as well as efficiently find c-bounded prefixes, for $c \leq 2$. Is it possible to identify the class of graphs where the problems become solvable in polynomial-time using 3-bounded prefixes? Note that this cannot be the class of graphs of degree at most three since both problems are already PSPACE-complete on this class. Could this class be that of 3-regular graphs? Alternatively, can we show NP-hardness (or PSPACE-hardness) for 3-regular graphs? We suspect the answer to the last question to be yes but a proof eludes us thus far.

4.5.2 Graphs of bounded degree

In this section, we present a fixed-parameter tractable algorithm for VC-Bound parameterized by $\ell + \Delta(G)$. The algorithm is rather technical, as it involves reductions to three intermediary problems, uses a structural decomposition of the input graph, and exploits the properties of nice reconfiguration sequences.

As previously noted, there is a close relation between the fixed-parameter tractability of the VC-Bound problem parameterized by ℓ and the size of the symmetric difference of the two vertex covers in question. In particular, when the size of the symmetric difference is greater than ℓ, we have a trivial no-instance. When the size is equal to ℓ, the problem is solvable by a simple $\mathcal{O}(\ell!)$ time enumeration algorithm. Even when the size of the symmetric difference is $\ell - c$, for any constant c, we can solve the problem in $\mathcal{O}(\ell^{\ell}\binom{n}{c})$ time. These observations imply that the problem becomes "harder" as the number of "choices" we have to make "outside" of the symmetric difference increases. At a very high level, our result on graphs of bounded degree relates to the symmetric difference as follows: In any yes-instance of the VC-Bound problem on graphs of degree at most d, one can easily bound the size of the symmetric difference and the set of vertices "not too far away" from it, i.e. the distance is some function of ℓ and d. However, to guarantee that the capacity constraint k, i.e. the maximum size of a vertex cover, is never violated, it may be necessary to add/remove vertices which are "far" from (maybe not even connected to) the symmetric difference. Hence, the main technical challenge is to show that finding such vertices can be accomplished "efficiently".

Before we discuss the technical details, we give a few additional definitions, propositions, and lemmas. We also introduce the intermediary problems, which we will use to develop our FPT algorithm. We extensively use the different kinds of edit sequences defined in Section 4.1. Given a (labeled) graph G, we say two subgraphs of G are *vertex-differing* whenever they differ in at least one vertex. Recall that for $r \geq 0$, the r-neighborhood of a vertex $v \in V(G)$ is defined as $N^r_G[v] = \{u \mid dist_G(u,v) \leq r\}$, $B(v,r) = N^r_G[v]$ is a ball of radius r around v and for $A \subseteq V(G)$, $B(A,r) = \bigcup_{v \in A} N^r_G[v]$. The following propositions will be useful for determining an upper bound on the running time of our algorithm.

Proposition 4.5.14. *For any graph G of degree at most d, $v \in V(G)$, and $A \subseteq V(G)$, $|B(v,r)| \leq d^{r+1}$ and $|B(A,r)| \leq |A|d^{r+1}$.*

Proposition 4.5.15. *Given a graph G of degree at most d, every vertex $v \in V(G)$ can appear in at most $d^{s^2(r+1)}$ vertex-differing connected subgraphs of G having at most s vertices and diameter at most r.*

Proof. Since G has degree at most d and the diameter of every connected subgraph must be at most r, the number of vertices which can belong to the same connected subgraph as v is at most d^{r+1} (Proposition 4.5.14). Of those d^{r+1} vertices, there are at most $\sum_{i=1}^{s} \binom{d^{r+1}}{i} < d^{s^2(r+1)}$ possible connected subgraphs on at most s vertices which include v and have diameter at most r. $\square$

Proposition 4.5.16. *Given a set S of vertices, $|S| \geq r$, the number of ways we can order at most r vertices from S into a sequence of length exactly r is less than $|S|^{2r}$.*

Proof. There are $\binom{|S|}{r}$ possible subsets of size r in S. For each subset, there are at most r^r ways we can order the vertices into a sequence of length exactly r. Since $|S| \geq r$, we establish the desired bound. $\square$

Proposition 4.5.17. *Given a set S of vertices, the total number of possible labeled edit sequences of length at most ℓ touching only vertices in S is at most $2^{\ell+1}|S|^{2\ell+2}$. The total number of possible partial edit sequences of length at most ℓ touching only vertices in S is at most $3^{\ell+1}|S|^{2\ell+2}$.*

Proof. There are at most 2^r possible unlabeled edit sequences of length exactly r. In addition, Proposition 4.5.16 states that there are at most $|S|^{2r}$ ways we can order at most r vertices from S into a sequence of size exactly r. Therefore, combining the two observations, we obtain the desired bound of at most $\sum_{i=1}^{\ell} 2^i |S|^{2i} < 2^{\ell+1}|S|^{2\ell+2}$ possible labeled edit sequences of length at most ℓ. Using similar arguments, we get the $\sum_{i=1}^{\ell} 3^i|S|^{2i} < 3^{\ell+1}|S|^{2\ell+2}$ bound on the total number of partial edit sequences of length at most ℓ. $\square$

Intermediary problems

Recall that an unlabeled edit sequence σ can be *applied* to graph G and vertex cover S of G if there exists a sequence of vertex labels L such that $label(\sigma, L)$ is valid starting from S. The VERTEX COVER WALK problem is formally defined as follows:

Vertex Cover Walk (VC-Walk)

Input:	A graph G, a vertex cover S of G, and an unlabeled edit sequence σ of length $\ell \geq 1$
Question:	Can we apply σ to G and S?

As an aside, we note that using an algorithm for the VC-Walk problem one can easily solve the Independent Set problem, i.e., finding an independent set of size ℓ in a graph G, by simply setting $S = V(G)$ and $\sigma = (\varnothing^-)^\ell$, where $(\varnothing^-)^\ell$ is an unlabeled edit sequence consisting of ℓ blank removal markers. In the parameterized setting, VC-Walk is at least as hard as Vertex Cover Local Search [57], defined as:

Vertex Cover Local Search (VC-Local-Search)

Input:	A graph G, a vertex cover S of G, and integer $\ell \geq 1$
Question:	Does G have a vertex cover S' such that $|S'| < |S|$ and $|S' \Delta S| \leq \ell$?

Lemma 4.5.18. VC-Walk *parameterized by* ℓ *is at least as hard as* VC-Local-Search *parameterized by* ℓ.

Proof. By Proposition 4.1.1, any algorithm that can solve the VC-Walk problem in $\mathcal{O}^*(f(\ell))$ time, for a computable function f, can solve the VC-Local-Search problem in $\mathcal{O}^*(f(\ell)2^{\ell+1})$ time by enumerating all possible unlabeled edit sequences of size at most ℓ, where the number of blank removal markers is at least one greater than the number of blank addition markers. □

VC-Local-Search is known to be W[1]-hard on graphs of bounded degeneracy and fixed-parameter tractable on graphs of bounded degree [57]. We also make use of the Multicolored Independent Set problem:

Multicolored Independent Set (MIS)

Input:	A graph G, a positive integer ℓ, and a vertex-coloring $c : V(G) \to \{c_1, \ldots, c_\ell\}$ for G
Question:	Does G have an independent set of size ℓ including exactly one vertex of each color?

Note that edges in $E(G)$ need not have endpoints assigned different colors. The MIS problem is W[1]-hard in general graphs as we can reduce the W[1]-hard Multicolored Clique problem to it by complementing all edges in the input graph [56]. For a vertex $v \in V(G)$, we denote by $c(v)$ the color assigned to v. We let $V_i(G)$ denote the set of vertices assigned colored c_i in G, i.e. $V_i(G) = \{v \in V(G) \mid c(v) = c_i\}$. We say vertex v belongs to *color class* c_i if $c(v) = c_i$.

Lemma 4.5.19. *The* Multicolored Independent Set *problem parameterized by ℓ is fixed-parameter tractable if for every vertex $v \in V(G)$ such that $c(v) = c_i$, $|N_G(v) \cap V_j(G)| \leq d$, for some fixed integer d, $i \neq j$, and $1 \leq i, j \leq \ell$. Moreover, there is an algorithm which solves the problem in $\mathcal{O}^*((d\ell)^{2\ell})$ time.*

Proof. Since every vertex $v \in V(G)$ assigned color c_i has at most d neighbors assigned color c_j for all $j \neq i$, v has at most $d\ell$ neighbors not in $V_i(G)$.

If $|V_i(G)| > d\ell$ for all i between one and ℓ, we can pick $d\ell + 1$ vertices from each $V_i(G)$ and delete all remaining vertices from G to obtain G'. By exhaustively enumerating all possible subsets of ℓ vertices assigned different colors in G', we are guaranteed to find an independent set of size ℓ. For v_1 a vertex in $V_1(G)$, since v_1 has at most d neighbors in $V_2(G)$, there must exist a vertex $v_2 \in V_2(G)$ such that $\{v_1, v_2\} \notin E(G')$. Similarly, v_1 and v_2 have at most d neighbors in $V_3(G)$, for a total of at most $2d$ neighbors. Since $|V_3(G)| = d\ell + 1 > 2d$, there must exist a vertex $v_3 \in V_3(G)$ such that $\{v_1, v_3\}, \{v_2, v_3\} \notin E(G')$. By repeating the same argument, we know that once we reach $V_\ell(G)$, all the previously selected vertices can have at most $(\ell - 1)d < d\ell + 1$ distinct neighbors in $V_\ell(G)$. Therefore, we know that there must exist an independent set of size ℓ in G' that includes exactly one vertex of each color. Moreover, we can exhaust all possibilities in $\mathcal{O}^*(\binom{d\ell^2+\ell}{\ell})$ time.

Otherwise, we know that the size of some $V_i(G)$ is at most $d\ell$. In this case, we can generate $d\ell$ new instances of the problem, where in each instance we delete all but one of the vertices in $V_i(G)$ to obtain G'. If G' has some $V_i(G)$ with at most $d\ell$ vertices, we repeat the process. Otherwise, we apply the exhaustive search described above. It is not hard to see that this algorithm runs in $\mathcal{O}^*((d\ell)^{2\ell})$ time. □

High level description

Our algorithm for VC-Bound relies on a combination of exhaustive enumeration and reductions to the problems VC-Walk and MIS. We make use of instances of AVC-Bound, i.e. the bounded reachability variant of Annotated Vertex Cover, in which the vertex set of the input graph is partitioned into sets X, W, and R such that X and R are separated, $S_{s \setminus t} \cup S_{t \setminus s} \subseteq X$, and no vertex in W is touched during reconfiguration.

AVC-Bound

Input: A graph G, positive integers k and ℓ,
two vertex covers S_s and S_t of G of size at most k,
and a partition (X, W, R) of $V(G)$ such that X and R are separated.

Question: Is there a reconfiguration sequence from S_s to S_t
of length at most ℓ touching no vertex in W?

The algorithm implicit in the proof of Lemma 4.5.20 generates 2ℓ instances $\{\mathcal{I}_1, \ldots, \mathcal{I}_{2\ell}\}$ of AVC-Bound such that the original instance is a yes-instance of VC-Bound if and only if some $\mathcal{I}_x$ is a yes-instance of AVC-Bound.

In each instance $\mathcal{I}$, solved by the Solve-AVC-Bound algorithm (Algorithm 2), there is a subset W of C_{st} separating a superset X of $S_{s\setminus t} \cup S_{t\setminus s}$ from a vertex set R such that no vertex in W is touched during reconfiguration. We use exhaustive enumeration to generate all partial edit sequences that touch only vertices in X; if (after cleaning) any produces a tight sequence that transforms S_s to S_t, we have a yes-instance. Otherwise, we consider all such sequences, say σ, which are valid and transform S_s to S_t but exceed the maximum allowed capacity constraint. By finding an appropriate labeled filling sequence β that touches vertices in R, we can free up capacity so that $merge(\sigma, \beta)$ is tight.

We can find such a β trivially if there is a sufficiently large independent set in the vertices of $S_s \cap R$ with no neighbours in O_{st}, as our reconfiguration sequence will consist of removing the vertices in the independent set to free up capacity, applying σ, and then adding back the vertices in the independent set. Otherwise, we reduce the problem of finding β to one instance of VC-Walk for each suitable unlabeled edit sequence γ of size the number of blank markers in σ. The walk must correspond to a walk from the node corresponding to $S_s \cap R$ in $R_{VC}(G[R], 0, k)$ back to the node itself. Any labeled edit sequence β returned by the Solve-VC-Walk algorithm (Algorithm 3) is nice, and hence we have a yes-instance if $merge(\sigma, \beta)$ is tight and transforms S_s into S_t.

To try all possible ways of generating β, we start by enumerating all d-well-formed unlabeled edit sequences γ of the appropriate size, and for each try all possible choices for the starting piece; as any reconfiguration sequence can be converted to a nice one, it suffices to restrict our examination to nice sequences (and correspondingly, unlabeled d-well-formed sequences). Given γ and a starting piece, we create t instances of VC-Walk, where instance $\mathcal{J}_y$ corresponds to the graph induced by the labeled central piece having y connected components. To solve instance $\mathcal{J}_y$, we consider all ways of assigning the d-add-remove segments to connected components. This allows us to create a sequence for each component, where γ_h touches only vertices in component h, and all vertices touched by γ'_h can be found in a ball of radius $|\gamma'_h|$.

We can reduce each subproblem to an instance of MIS, where a color c_h corresponds to component h. In our auxiliary graph G_A (Algorithm 4), we create vertices for each labeled edit sequence λ, where $G[V(\lambda)]$ is connected and λ can be derived by adding labels to γ_h. There is an edge between two vertices in G_A if they have different colors and sets of vertices associated with their edit sequences are not separated. Thus, a solution to MIS, i.e. an independent set of size y, indicates that there are y labeled edit sequences with separated vertex sets, as required to complete the central piece of γ. As the ending piece of γ' will be determined by the starting and central pieces, this completes the algorithm.

Technical details

We now discuss the technical details of our algorithms and give a bound on the overall running times. We make use of *Ramsey Numbers*: for any positive integers p and q, there exists a minimum number $\mathcal{R}(p, q)$ (Ramsey Number), such that any graph on at least $\mathcal{R}(p, q)$ vertices contains either a clique of size p or an independent set of size q. Moreover, $\mathcal{R}(p, q) \leq \binom{p+q-2}{q-1}$ [76].

Lemma 4.5.20. *For any instance of* VC-Bound*, there exists a set of* 2ℓ *instances* $\{\mathcal{I}_1, \ldots, \mathcal{I}_{2\ell}\}$ *of* AVC-Bound *such that the original instance is a yes-instance for* VC-Bound *if and only if at least one* $\mathcal{I}_x$ *is a yes-instance for* AVC-Bound*,* $1 \leq x \leq 2\ell$ *and for each* X_x*,* $S_{s\setminus t} \cup S_{t\setminus s} \subseteq X_x$*.*

Proof. For an instance (G, S_s, S_t, k, ℓ) of VC-Bound, we consider the partition of $V(G)$ into the sets C_{st}, $S_{s\setminus t}$, $S_{t\setminus s}$, and the independent set O_{st}. We further subdivide some of these sets (Figure 4.9). We let C_i be the set of vertices in C_{st} that are at distance i from $S_{s\setminus t} \cup S_{t\setminus s}$. That is, we set $C_i = \{B(S_{s\setminus t} \cup S_{t\setminus s}, i) \setminus B(S_{s\setminus t} \cup S_{t\setminus s}, i-1)\} \cap C_{st}$, for $i \geq 1$. By Proposition 4.5.1, there can be no edges between vertices in $S_{s\setminus t} \cup S_{t\setminus s}$ and vertices in O_{st}. Therefore, we let $O_i = \{B(S_{s\setminus t} \cup S_{t\setminus s}, i) \setminus B(S_{s\setminus t} \cup S_{t\setminus s}, i-1)\} \cap O_{st}$ be the set of vertices in O_{st} that are at distance i from $S_{s\setminus t} \cup S_{t\setminus s}$, for $i \geq 2$. We let C_∞ and O_∞ be the sets of vertices in C_{st} and O_{st}, respectively, which are not in the r-neighborhood of $S_{s\setminus t} \cup S_{t\setminus s}$ for any value of r. Note that $C_\infty \cup O_\infty$ is not necessarily connected.

By our definition of C_i, every vertex $v \in C_i$, $i \geq 1$, must be at distance i from some vertex in $S_{s\setminus t} \cup S_{t\setminus s}$. Similarly, every vertex $v \in O_i$, $i \geq 2$, must be at distance i from some vertex in $S_{s\setminus t} \cup S_{t\setminus s}$. Therefore, since O_i is an independent set, for any vertex $v \in O_i$, $i \geq 2$, all the neighbors of v must be in $C_{i-1} \cup C_i \cup C_{i+1}$. Since there are no edges between O_{st} and $S_{s\setminus t} \cup S_{t\setminus s}$, the set C_1 separates $S_{s\setminus t} \cup S_{t\setminus s}$ from the rest of the graph. Moreover, for any $i \geq 2$, the vertices in $C_i \cup C_{i+1}$ separate $X_i = S_{s\setminus t} \cup S_{t\setminus s} \cup \bigcup_{1 \leq j < i} C_j \cup \bigcup_{2 \leq j \leq i} O_j$ from the

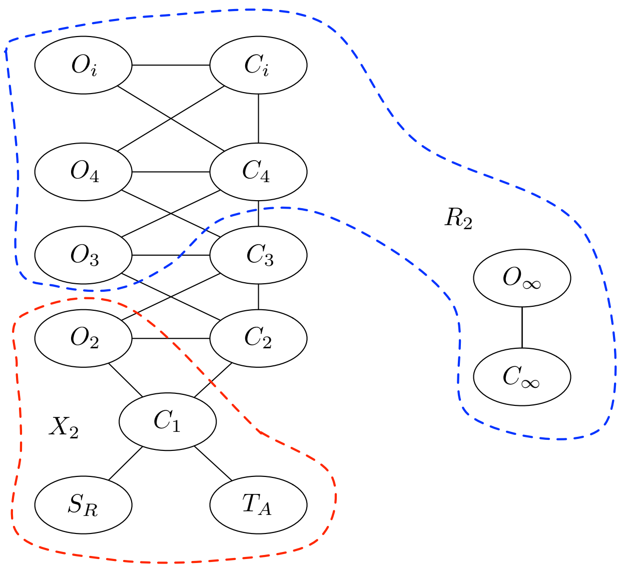

Figure 4.9: The subdivision of the sets C_{st} and O_{st} along with the sets X_2 and R_2.

rest of the graph $R_i = V(G) \setminus \{X_i \cup C_i \cup C_{i+1}\}$ (Figure 4.9). We say $W_i = V(G) \setminus \{X_i \cup R_i\}$ is a *wall-set*. The separation of X_i from R_i plays a crucial role in our algorithm. Since X_i is a subset of $B(S_{s\setminus t} \cup S_{t\setminus s}, i)$, it follows from Proposition 4.5.14 that:

Proposition 4.5.21. $|X_i| \leq |S_{s\setminus t} \cup S_{t\setminus s}|d^{i+1} \leq \ell d^{i+1}$.

In the first instance $\mathcal{I}_1$, the inputs to the Solve-AVC-Bound algorithm consist of the sets $X = S_{s\setminus t} \cup S_{t\setminus s}$, $W = C_1$, and $R = V(G) \setminus \{X_1 \cup C_1\}$. For the $2\ell - 1$ remaining instances, $X = X_x$, $W = C_x \cup C_{x+1}$, and $R = R_x$, for $2 \leq x \leq 2\ell$. In all instances, the vertices in X are separated from the vertices in R. This separation allows us to "divide" the problem into two "subproblems", where the first can be solved by brute-force enumeration (Algorithm 2, lines 3–11) and the second can be solved via the Solve-VC-Walk algorithm (Algorithm 2, lines 12–18). In each instance $\mathcal{I}_x$, $1 \leq x \leq 2\ell$, we force the vertices in the wall-set W to remain in every vertex cover throughout any reconfiguration sequence, i.e. we do not allow any of those vertices to be touched. Recall that all vertices in the wall-set must be in C_{st}. Assume that the Solve-AVC-Bound algorithm (Algorithm 2) can solve each of those 2ℓ instances in `FPT` time without touching the vertices in the wall-set. If any of those instances is a yes-instance we are done, since we have found a reconfiguration sequence of length at most ℓ from S_s to S_t. Otherwise, we know from the first no-instance that if a reconfiguration sequence of length at most ℓ from S_s to S_t exists, then it must touch some vertex in $W = C_1$. Generally, we know from the xth

Algorithm 2 Solve-AVC-Bound

```
Input: A graph $G$ of degree at most $d$, a positive integer $k$, two vertex covers $S_s$ and $S_t$
    of $G$ of size at most $k$, a positive integer $\ell$, and partition $X$, $W$, and $R$ of $V(G)$.
Output: A reconfiguration sequence of length at most $\ell$ from $S_s$ to $S_t$ if one exists and $\varnothing$
    otherwise.
 1: Enumerate every partial labeled edit sequence $\sigma$ of length at most $\ell$ that only touches
    vertices in $X$;
 2: for each $\sigma$
 3:   if ($clean(\sigma)$ is tight and transforms $S_s$ into $S_t$)
 4:     return $clean(\sigma)$;
 5:   if ($clean(\sigma)$ is valid and transforms $S_s$ into $S_t$ and $cap(clean(\sigma)) - k \leq \ell$)
 6:     $c = cap(clean(\sigma)) - k$;
 7:     $S'_s = \{S_s \cap R\} \setminus \{v \mid v \in S_s \cap R \wedge |N_G(v) \cap O_{st}| = 0\}$;
 8:     if ($|S'_s| > \mathcal{R}(d+2, c)$)
 9:       Find an independent set of size $c$ in $S'_s$;
10:       Let $L$ be the set of vertex labels (in any order) in the independent set;
11:       return $concat(concat(label((\varnothing^-)^c, L), clean(\sigma)), label((\varnothing^+)^c, L))$;
12:     Generate every $d$-well-formed unlabeled $\gamma$ of size $|\sigma| - |clean(|\sigma|)|$;
13:     for each $\gamma$
14:       $\beta$ = Solve-VC-Walk($G[R], S_s \cap R, \gamma$);
15:       if ($\beta$ touches some vertex an odd number of times)
16:         continue;
17:       if ($merge(\sigma, \beta)$ is tight and transforms $S_s$ into $S_t$)
18:         return $merge(\sigma, \beta)$;
19: return $\varnothing$;
```

no-instance, for $x > 1$, that if a reconfiguration sequence of length at most ℓ from S_s to S_t exists, then it must touch some vertex in $W = C_x \cup C_{x+1}$. When all 2ℓ instances are no-instances, we know that any reconfiguration sequence from S_s to S_t must touch a vertex from each of the following sets: $C_1, C_2 \cup C_3, C_3 \cup C_4, \ldots, C_{2\ell} \cup C_{2\ell+1}$. However, $C_1 \cap \{C_2 \cup C_3\} \cap \{C_4 \cup C_5\} \cap \ldots \cap \{C_{2\ell} \cup C_{2\ell+1}\} = \emptyset$ and therefore at least ℓ vertices from C_{st} must be touched, which implies that our original VC-Bound instance is a no-instance. □

Lemma 4.5.22. *If* VC-Walk *is solvable in $\mathcal{O}^*(f(d,\ell))$ time, for some computable function f, on a graph G of degree at most d, then* VC-Bound *is solvable in $\mathcal{O}^*(g(d,\ell)f(d,\ell))$ time on G, for some computable function g.*

Proof. Assuming an algorithm which solves the VC-Walk in $\mathcal{O}^*(f(d,\ell))$ time, the worst case running time of the Solve-AVC-Bound algorithm (Algorithm 2) is in $\mathcal{O}^*(3^{\ell+1}$ $(\ell d^{2\ell+1})^{2\ell+2} \quad (d+\ell)^{\ell} \; 2^{\ell} \quad f(d,\ell))$; the size of X is bounded above by $\ell d^{2\ell+1}$ in the worst case (Proposition 4.5.21). Hence, we have a total of at most $3^{\ell+1}(\ell d^{2\ell+1})^{2\ell+2}$ partial edit sequences to enumerate (Proposition 4.5.17). Since $G[S'_s]$ has degree at most d, it cannot contain a clique of size $d+2$. Hence, if $|S'| > \mathcal{R}(d+2,c)$ (Algorithm 2, line 8), then $G[S'_s]$ must contain an independent set of size $c < \ell$, which we can find in $\mathcal{O}^*(\binom{R(d+2,\ell)}{\ell})$ or $\mathcal{O}^*((d+\ell)^{\ell})$ time [93]. When $|S'_s| \leq \mathcal{R}(d+2,c)$, the unlabeled edit sequences we consider must be of size at most $\ell - 1$. By Proposition 4.1.1, there are at most 2^{ℓ} such sequences to check for each σ (Algorithm 2, lines 12–18). Merging, cleaning, and concatenating sequences as well as checking whether they are valid or tight can be accomplished in time polynomial in the size of the sequences and n. Since there are at most 2ℓ instances of AVC-Bound to solve, we obtain the desired bound. □

Lemma 4.5.23. *The* Solve-VC-Walk *algorithm (Algorithm 3) returns a labeled edit sequence $label(\gamma, L)$ that is nice starting at $S_s \cap R$, if one exists.*

Proof. The inputs to the Solve-VC-Walk algorithm will consist of the graph $G[R]$, the vertex cover $S_s \cap R$ of $G[R]$, and a d-well-formed unlabeled edit sequence γ. We let γ_s, γ_c, and γ_e denote the starting, central, and ending pieces of γ respectively. Moreover, we let $\gamma_c = \{s_1, \ldots, s_t\}$ where each s_i, $1 \leq i \leq t$, is a d-add-remove segment in γ_c. The end-goal of the algorithm is to find an ordered set of vertex labels L such that $\gamma' = label(\gamma, L)$ is nice starting from $S_s \cap R$. We let γ'_s, $\gamma'_c = \{s'_1, \ldots, s'_t\}$, and γ'_e denote the starting, central, and ending pieces of γ' respectively. Since every valid edit sequence can be converted into a nice edit sequence (Lemma 4.1.10) and given that the Solve-AVC-Bound algorithm tries all possible γ's, we can assume without loss of generality that γ' will be nice. To find γ', we consider each of the pieces of γ separately. Note that if γ' exists, then we know from Definition 4.1.6 that the graph induced by $V(s'_i)$, $1 \leq i \leq t$, must be a connected subgraph of $G[R]$. If $V(s'_i)$ and $V(s'_j)$ are separated for all $1 \leq i, j \leq t$ and $i \neq j$, then $G[V(\gamma'_c)]$ is a graph with $t < |\gamma|$ connected components. The number of vertices in and the diameter of each connected component is at most $2d$. However, $V(s'_i)$ and $V(s'_j)$ need not be separated. In fact, it might be the case that $G[V(\gamma'_c)]$ is connected.

The Solve-VC-Walk algorithm starts by building the set S', which is equal to the set S'_s of Algorithm 2 and whose size cannot be "too large" due to line 9 of Algorithm 2. Whenever $|\gamma_s| > 0$, $V(\gamma'_s)$ must be an independent set in S'. Therefore, we enumerate all the possible independent sets of size $|\gamma_s|$ in S'. Each such set will result in a set of $|\gamma_s|$ vertex labels we can assign to the removal markers in γ_s (in any order) to get γ'_s (Algorithm 3, lines 5–8). For each γ'_s, we then attempt to label γ_c. To do so, we generate t instances $\{\mathcal{J}_1, \ldots, \mathcal{J}_t\}$ to cover all possible scenarios in which $G[V(\gamma'_c)]$ has anywhere between 1 and

Algorithm 3 SOLVE-VC-WALK

Input: A graph G of degree at most d, a vertex cover S of G, and a unlabeled edit sequence γ (assumed to be d-well-formed).

Output: A labeled edit sequence $label(\gamma, L)$ which is nice starting from S if one exists and $\varnothing$ otherwise, where L is some ordered set of vertex labels.

```
1: Let $\gamma_s$ be the starting piece of $\gamma$;
2: Let $\gamma_c = \{s_1, \ldots, s_t\}$ be the central piece of $\gamma$;
3: Let $\gamma_e$ be the ending piece of $\gamma$;
4: $S' = \{v \mid v \in S \wedge |N_G(v) \cap \{V(G) \setminus S\}| = 0\}$;
5: Enumerate all independent sets of size $|\gamma_s|$ in $S'$;
6: for each independent set
7:    Let $L$ be the set of vertex labels (in any order) in the independent set;
8:    $\gamma'_s = label(\gamma_s, L)$;
9:    for each instance $\mathcal{J}_y$, $1 \leq y \leq t$
10:       $S'' = V(S, \gamma'_s)$;
11:       Create the connected components $\mathcal{CC}_1, \ldots, \mathcal{CC}_y$ (initially empty sets);
12:       Generate all mappings between components and $d$-add-remove segments;
13:       for each mapping
14:          Split $\gamma_c$ into $y$ pieces $\gamma_1, \ldots, \gamma_y$;
15:          $G_A$ = CONST-AUX-GRAPH$(G, S'', \gamma_1, \ldots, \gamma_y)$;
16:          Find a multicolored independent set in $G_A$ of size $y$;
17:          if ($G_A$ does not contain such a set)
18:             continue;
19:          else
20:             Let $Q = \{u, \ldots, v\}$ be such a set;
21:             for each $\lambda \in mix(seq(u), \ldots, seq(v))$
22:                Let $L$ be the set of ordered labels in $\lambda$;
23:                if ($\lambda$ is nice starting from $S''$ and $\lambda = label(\gamma_c, L)$)
24:                   $\gamma'_c = label(\gamma_c, L)$;
25:                   Complete the ending piece $\gamma'_e$;
26:                   $\gamma' = concat(concat(\gamma'_s, \gamma'_c), \gamma'_e)$;
27:                   if ($\gamma'$ is labeled and nice starting from $S$)
28:                      return $\gamma'$;
29: return $\varnothing$;
```

t connected components. Note that we construct the set $S'' = V(S \cap R, \gamma'_s)$ on line 10 of Algorithm 3 so that γ'_c starts from S'' and we can ignore γ'_s for now. With each instance $\mathcal{J}_y$,

$1 \leq y \leq t$, we associate y initially empty sets $\{\mathcal{CC}_1, \ldots, \mathcal{CC}_y\}$. Each set will correspond to a connected component in $G[V(\gamma'_c)]$. Then for each instance $\mathcal{J}_y$, $1 \leq y \leq t$, we enumerate all possible mappings between $\{\mathcal{CC}_1, \ldots, \mathcal{CC}_y\}$ and the segments of γ_c, i.e. every segment of γ'_c will be forced to touch vertices from a single connected subgraph of $G[R]$ which will correspond to some component $\mathcal{CC}_i$ of $G[V(\gamma'_c)]$, $1 \leq i \leq y$. Every connected component $\mathcal{CC}_i$, $1 \leq i \leq y$, has at most $2dt < |\gamma_c|$ vertices (at most t d-add-remove segments) and therefore its diameter is bounded above by $2dt$. Hence, for every $\mathcal{CC}_i$ there exists a center vertex $v \in V(G[R])$, such that $\mathcal{CC}_i \subseteq B(v, 2dt)$. Lines 13–28 of the Solve-VC-Walk algorithm exploit this fact to construct an auxiliary graph G_A used to find a set L of ordered labels such that $\gamma'_c = \mathit{label}(\gamma_c, L)$ is nice starting from S''.

After mapping each segment of γ_c to a component $\mathcal{CC}_i$ in line 12 of Algorithm 3, we split γ_c into y pieces $\gamma_1, \ldots, \gamma_y$, where each piece γ_i is obtained from γ_c by cutting all segments which were not mapped to $\mathcal{CC}_i$, $1 \leq i \leq y$. Each piece corresponds to a d-well-formed unlabeled edit sequence. Hence, if we call the Const-Aux-Graph algorithm (Algorithm 4) with inputs the graph $G[R]$, the vertex cover S'' (or $V(S \cap R, \gamma'_s)$) of $G[R]$, and the y pieces of γ_c, the resulting auxiliary graph G_A will satisfy Lemma 4.5.24. That is, the existence of a multicolored independent set of size y in G_A implies the existence of $L_1, \ldots, L_y$ such that $\gamma'_1 = \mathit{label}(\gamma_1, L_1), \ldots, \gamma'_y = \mathit{label}(\gamma_y, L_y)$ are nice starting from S'', each subgraph $G[V(\gamma'_1)], \ldots, G[V(\gamma'_y)]$ is connected, and $\gamma'_1, \ldots, \gamma'_y$ are compatible. Therefore, $\gamma'_c \in \mathit{mix}(\gamma'_1, \ldots, \gamma'_y)$ is valid (Definition 4.1.4) and nice; as γ'_c consists only of d-add-delete segments all satisfying the connectivity and early removal invariants of Definition 4.1.6. If we have a no-instance of the MIS problem, then either our mapping from segments to connected components was incorrect or the number of connected components was incorrect. The former case is handled by the fact that we are trying all possible mappings and the latter case is handled by one of the t generated instances.

To complete the ending piece of γ', we simply iterate over $\mathit{concat}(\gamma'_s, \gamma'_c)$ and add an addition marker for every vertex touched an odd number of times in $\mathit{concat}(\gamma'_s, \gamma'_c)$; as γ' must be a walk from $S \cap R$ in $R_{\text{VC}}(G[R], 0, k)$ back to $S \cap R$. If the concatenation of all three sequences γ'_s, γ'_c, and γ'_e produces a nice labeled edit sequence starting from $S \cap R$ then we return it. Otherwise, Algorithm 3 proceeds to the next iteration. If all t instances are no-instances, then we know that the original VC-Walk is a no-instance. □

Lemma 4.5.24. *Given a graph G of degree at most d, a vertex cover S of G, and t d-well-formed unlabeled edit sequences $\gamma_1, \ldots, \gamma_t$ each of size at most ℓ, the* Const-Aux-Graph *algorithm (Algorithm 4) generates a graph G_A with at most t color classes in* FPT *time. Each vertex $v \in V(G_A)$ has at most $(d+1)\ell 2^\ell \ell! d^{\ell^2(\ell+1)}$ neighbors in every color class other than $c(v)$. Each color class of G_A has at most $|V(G)| 2^{\ell+1} (d^{\ell+1})^{2\ell+2}$ vertices. Moreover, the existence of a multicolored independent set of size t in G_A implies the existence of*

$L_1, \dots, L_t$ such that $\gamma'_1 = label(\gamma_1, L_1), \dots, \gamma'_t = label(\gamma_t, L_t)$ are nice starting from S, each subgraph $G[V(\gamma'_1)], \dots, G[V(\gamma'_t)]$ is connected, and $\gamma'_1, \dots, \gamma'_t$ are compatible.

Proof. We construct a graph G_A such that a multicolored independent set of size t in G_A implies the existence of $\gamma'_1, \dots, \gamma'_t$ where:

(1) $\gamma'_1, \dots, \gamma'_t$ are labeled,

(2) $G[V(\gamma'_1)], \dots, G[V(\gamma'_t)]$ are all connected,

(3) $V(\gamma'_1), \dots, V(\gamma'_t)$ are all separated, and

(4) $\gamma'_1, \dots, \gamma'_t$ are all nice starting from S.

When conditions (1), (3), and (4) are satisfied, we know from Proposition 4.1.5 that $\gamma'_1, \dots, \gamma'_t$ are compatible. Conditions (2) and (3) are enforced by our mapping from components to segments in the Solve-VC-Walk algorithm (Algorithm 3, lines 11–14) and line 8 of Algorithm 4. If there exists $\gamma'_1, \dots, \gamma'_t$ satisfying conditions (1) to (4), we know that there must exist a set of distinct vertices $\{u_1, \dots, u_t\} \in V(G)$ such that $V(\gamma'_1) \subseteq B(u_1, |\gamma'_1|), \dots, V(\gamma'_t) \subseteq B(u_t, |\gamma'_t|)$.

G_A will contain a vertex assigned color c_i, $1 \leq i \leq t$, for each $C \subseteq V(G)$ satisfying the following properties:

(1) $G[C]$ is connected,

(1) $|C| \in \{|\gamma_1|, \dots, |\gamma_t|\}$, and

(2) $label(\gamma_i, L)$ is nice starting from S, for L the set of vertex labels in C.

For every γ_i, $1 \leq i \leq t$, we attempt to assign each $v \in V(G)$ the role of a center and enumerate every possible labeled edit sequence λ of size $|\gamma_i|$ touching vertices in $B(v, |\gamma_i|)$ only. For each λ satisfying the conditions on line 8 of Algorithm 4, we add a vertex in $V(G_A)$ and assign it color c_i. Each vertex in $V(G_A)$ will be associated with a corresponding λ, i.e. for every vertex $u \in V(G_A)$ we let $seq(u) = \lambda$ (Algorithm 4, lines 6–11). After repeating the same process for every γ_i, $V(G_A)$ will have at most t color classes. The edges of G_A are added as follows: For every two vertices $u, v \in V(G_A)$ such that $c(u) \neq c(v)$, there is an edge $\{u, v\}$ if $V(seq(u))$ and $V(seq(v))$ are not separated (Algorithm 4, lines 12–14).

Algorithm 4 Const-Aux-Graph

Input: A graph G of degree at most d, a vertex cover S of G, and t unlabeled edit sequences $\gamma_1, \ldots, \gamma_t$ that are d-well-formed.

Output: A graph G_A.

```
1: $G_A = (V, E)$;
2: $V(G_A) = E(G_A) = \emptyset$;
3: for each $\gamma_i$, $1 \leq i \leq t$
4:     for each $v \in V(G)$
5:         Enumerate every labeled edit sequence $\lambda$ of size $|\gamma_i|$ that only touches vertices in $B(v, |\gamma_i|)$;
6:         for each $\lambda$
7:             Let $L$ be the set of ordered labels in $\lambda$;
8:             if ($\lambda$ is nice starting from $S$ and $\lambda = label(\gamma_i, L)$ and $G[V(\lambda)]$ is connected)
9:                 $V(G_A) = V(G_A) \cup \{v\}$;
10:                $c(v) = c_i$ ($v$ is assigned color $c_i$)
11:                Store $\lambda$ in $seq(v)$;
12: for each $(u, v) \in V(G_A)$ such that $c(u) \neq c(v)$
13:     if ($V(seq(u))$ and $V(seq(v))$ are not separated)
14:         $E(G_A) = E(G_A) \cup \{\{u, v\}\}$ (no double edges);
15: return $G_A$;
```

The running time of the Const-Aux-Graph algorithm follows from the description. For each γ_i, $1 \leq i \leq t$, and each $v \in V(G)$, $|B(v, |\gamma_i|)| \leq d^{|\gamma_i|+1} \leq d^{\ell+1}$ by Proposition 4.5.14. Hence, there are at most $2^{\ell+1}(d^{\ell+1})^{2\ell+2}$ possible labeled edit sequences λ of size $|\gamma_i| \leq \ell$ to enumerate (Proposition 4.5.17). Whenever λ is a nice edit sequence starting from S and $G[V(\lambda)]$ is connected, a corresponding vertex u assigned color c_i is added to $V(G_A)$ and $seq(u)$ is set to λ (Algorithm 4, lines 3–11). Hence, every color class of G_A will have at most $|V(G)|2^{\ell+1}(d^{\ell+1})^{2\ell+2}$ vertices.

Note that some color classes in G_A could be empty. To prove the bound on the degree of each vertex, we assume that at least two color classes are non-empty. By Proposition 4.5.15, each vertex $v \in V(G)$ can appear in at most $d^{\ell^2(\ell+1)}$ vertex-differing connected subgraphs of G of size at most ℓ. For each of those vertex-differing subgraphs of G, there are at most $\ell!$ ways we can order their corresponding vertex labels. Therefore, there are at most $2^\ell \ell! d^{\ell^2(\ell+1)}$ distinct labeled edit sequences λ such that λ touches vertex v, $|\lambda| \leq \ell$, and $G[V(\lambda)]$ is connected.

We let $V_i(G_A)$ and $V_j(G_A)$ be any two non-empty color classes in G_A, where $1 \leq$

$i, j \leq t$. We let u be a vertex in $V_i(G_A)$ and we count the number of possible neighbors of u in $V_j(G_A)$. For any vertex $w \in V_j(G_A)$, an edge $\{u, w\}$ in $E(G_A)$ implies that $\{V(seq(u)) \cup N_G(V(seq(u)))\} \cap \{V(seq(w)) \cup N_G(V(seq(w)))\} \neq \emptyset$, as otherwise $V(seq(u))$ and $V(seq(w))$ are separated. Moreover, we know that $G[V(seq(u)) \cup N_G(V(seq(u)))]$ is a connected subgraph of G of size at most $(d+1)|V(seq(u))| \leq (d+1)\ell$. Hence, there are at most $(d+1)\ell 2^\ell \ell! d^{\ell^2(\ell+1)}$ labeled edit sequences λ such that λ touches some vertex in $V(seq(u)) \cup N_G(V(seq(u)))$, $|\lambda| \leq \ell$, and $G[V(\lambda)]$ is connected. Therefore, there can be at most $(d+1)\ell 2^\ell \ell! d^{\ell^2(\ell+1)}$ vertices $w \in V_j(G_A)$ such that $\{V(seq(u)) \cup N_G(V(seq(u)))\} \cap \{V(seq(w)) \cup N_G(V(seq(w)))\} \neq \emptyset$, as needed. □

Lemma 4.5.25. *Every instance of the* VC-WALK *problem generated in the* SOLVE-AVC-BOUND *algorithm (Algorithm 2) can be solved in* FPT *time on graphs of degree at most* d.

Proof. We know from Algorithm 2 (line 8) that the size of S' (Algorithm 3, line 10) is less than $\mathcal{R}(d+2, \ell)$. Hence enumerating all independent sets of fixed size ℓ (or less) in S' can be accomplished in $\mathcal{O}^*(\binom{\mathcal{R}(d+2,\ell)}{\ell})$ time. For each instance $\mathcal{J}_y$, $1 \leq y \leq t \leq \ell$, there are at most y^t possible mappings. So in the worst case, we have $\sum_{y=1}^{t} y^t < t^{t+1} < \ell^{\ell+1}$ mappings to consider. Constructing G_A can be done in FPT time (Lemma 4.5.24) and the vertices of G_A will satisfy the properties needed to apply Lemma 4.5.19 and solve the MIS problem in FPT time. Every remaining operation can be accomplished in time polynomial in n and ℓ. Therefore, every instance of the VC-WALK problem can be solved in FPT time. □

Combining Lemmas 4.5.22 and 4.5.25, we know that there exists an algorithm which solves the VC-BOUND problem in FPT time on graphs of degree at most d. Hence:

Theorem 4.5.26. VC-BOUND *parameterized by* ℓ *is fixed-parameter tractable on graphs of bounded degree.*

Combining Theorem 4.5.26 with Proposition 2.4.9, we get:

Corollary 4.5.27. IS-BOUND *parameterized by* ℓ *is fixed-parameter tractable on graphs of bounded degree.*

4.5.3 Graphs of bounded treewidth

In this section, we present a dynamic programming algorithm solving VC-BOUND in $\mathcal{O}(f(\ell, tw(G))n^{\mathcal{O}(1)})$ time on graph G, for some computable function f. Throughout, we will consider one fixed instance (G, S_s, S_t, k, ℓ) of VC-Bound and a nice tree decomposition $\mathcal{T} = (T, \chi)$ of G. Recall that for node $i \in V(T)$, we use T_i to denote the subtree of T rooted at i and V_i to denote the set of vertices of G contained in the bags of T_i. Moreover, similarly to the previous section, we will ask, for a fixed unlabeled edit sequence σ, whether there is a reconfiguration sequence which at the i^{th} step, $1 \leq i \leq |\sigma| = \ell$, removes a vertex when $\sigma[i] = \varnothing^-$ and adds a vertex when $\sigma[i] = \varnothing^+$. The final algorithm then asks such a question for every unlabeled edit sequence σ that does not violate the maximum allowed capacity constraint: $cap(\sigma) = max_{1 \leq p \leq |\sigma|}(|S_s| + \sum_{i=1}^{|p|} \text{sign-}\sigma[i]) \leq k$. This will add a factor of at most $2^{\ell+1}$ to the overall running time.

Signatures as equivalence classes

To obtain a more succinct representation of edit sequences, we observe that in order to propagate information up from the leaves to the root of a nice tree decomposition, we can ignore vertices outside of the currently considered bag (X_i for $i \in V(T)$) and indicate only whether a slot in an edit sequence has been used by a vertex in any previously processed bags, i.e. a vertex in $V_i \setminus X_i$, $i \in V(T)$. To do so, we introduce the notion of signatures, which are edit sequences with only one minor modification.

Definition 4.5.28. *A* signature *over a set $X \subseteq V(G)$ is an edit sequence τ such that vertex-$\tau[i] \in X \cup \{\texttt{used}, \texttt{unused}\}$, $1 \leq i \leq |\tau|$.*

In other words, instead of allowing vertex-$\tau[i] \in V(G) \cup \{\varnothing\}$, we restrict vertex markers to a set $X \subseteq V(G)$ and introduce the `used` and `unused` markers. An example of a signature is given in Figure 4.10, where `used` markers are shown in black and `unused` markers in white. The total number of signatures over a bag X of at most t vertices is $(t+3)^\ell$. Our dynamic programming algorithm starts by considering a signature with only `unused` markers in each leaf node, specifies when a vertex may be added/removed in introduce nodes by replacing `unused` markers with vertex markers ($\tau[i] = \texttt{unused}^+$ becomes $\tau[i] = v^+$ and $\tau[i] = \texttt{unused}^-$ becomes $\tau[i] = v^-$ for the introduced vertex v and $1 \leq i \leq \ell$), merges signatures in join nodes, and replaces vertex markers with `used` markers in forget nodes.

−	−	+	−	+	+	−	+	−	+	−	+
2	■	1		2		1	■			2	1

Figure 4.10: Example of a signature.

A feasible signature must ensure that no step deletes a vertex that is absent or adds a vertex that is already present, and that the set of vertices obtained after applying reconfiguration steps to $S_s \cap X$ is the set $S_t \cap X$. Additionally, because the property of a graph being edgeless is hereditary, we can check whether this property is at least locally satisfied (in $G[X]$) after each step of the sequence. More formally, we have the following definition.

Definition 4.5.29. *A signature over $X \subseteq V(G)$ is* feasible *if for all $1 \leq i \leq \ell$*

- $V(\tau[i]) \in V(S_s \cap X, \tau[1, i-1])$ *whenever* $V(\tau[i]) \in X$ *and sign-*$\tau[i] = -1$*,*
- $V(\tau[i]) \notin V(S_s \cap X, \tau[1, i-1])$ *whenever* $V(\tau[i]) \in X$ *and sign-*$\tau[i] = +1$*,*
- $V(S_s \cap X, \tau) = S_t \cap X$*, and*
- $G[X \setminus V(S_s \cap X, \tau[1, i])]$ *is an independent set.*

It is not hard to see that a signature τ over X is feasible if and only if $S_s \cap X, V(S_s \cap X, \tau[1, 1]), V(S_s \cap X, \tau[1, 2]), \ldots, V(S_s \cap X, \tau[1, \ell])$ is a well-defined path between $S_s \cap X$ and $S_t \cap X$ in $R_{VC}(G[X], 0, n)$. We will consider only feasible signatures. The dynamic programming algorithm will enumerate exactly the signatures that can be extended to feasible signatures over V_i, $i \in V(T)$, in the following sense:

Definition 4.5.30. *A signature τ over $Y \subseteq V(G)$* extends *a signature τ' over $X \subseteq V(G)$ if it is obtained by replacing some* `used` *markers with vertex markers from $Y \setminus X$.*

For many problems, the fact that S_s is a solution for $G[X]$, for each bag X, does not imply that S_s is a solution for G, and checking this 'local' notion of feasibility will not be enough – the algorithm will have to maintain additional information. One such example is the Odd Cycle Transversal problem, which we discuss in Chapter 5.

To process nodes of the tree decomposition, we now define ways of generating signatures from other signatures. The introduce operation determines all ways that an introduced vertex can be represented in a signature, replacing `unused` markers in the signature of its child.

Definition 4.5.31. *Given a signature τ over X and a vertex $v \notin X$, the* introduce operation*, introduce(τ, v), returns the following set of signatures over $X \cup \{v\}$: For every subset I of indices i for which vertex-$\tau[i] =$* `unused`*, consider a copy τ' of τ where for all $i \in I$ we set vertex-$\tau'[i] = v$, check if it is feasible, and if so, add it to the set.*

In particular $\tau \in \mathit{introduce}(\tau, v)$ and $|\mathit{introduce}(\tau, v)| \leq 2^{\ell}$. All signatures obtained through the introduce operation are feasible, because of the explicit check.

Definition 4.5.32. *Given a signature τ over X and a vertex $v \in X$, the* forget operation *returns a new signature $\tau' = \mathit{forget}(\tau, v)$ over $X \setminus \{v\}$ such that for all $1 \leq i \leq \ell$, we have vertex-$\tau'[i]$ =* `used` *if vertex-$\tau[i] = v$ and vertex-$\tau'[i]$ = vertex-$\tau[i]$ otherwise.*

Since $V(S_s \cap X \setminus \{v\}, \tau'[1, i]) = V(S_s \cap X, \tau[1, i]) \setminus \{v\}$ for $1 \leq i \leq \ell$, it is easy to check that the forget operation preserves feasibility.

Definition 4.5.33. *Given two signatures τ_1 and τ_2 over $X \subseteq V(G)$, we say τ_1 and τ_2 are* joinable *if for all $1 \leq i \leq \ell$:*

- *vertex-$\tau_1[i]$ = vertex-$\tau_2[i]$ =* `unused`*,*
- *vertex-$\tau_1[i]$ = vertex-$\tau_2[i] = v$ for some $v \in X$, or*
- *one of vertex-$\tau_1[i]$ or vertex-$\tau_2[i]$ is equal to* `used` *and the other is equal to* `unused`*.*

For two joinable signatures τ_1 and τ_2, the join operation *returns a new signature $\tau' = \mathit{join}(\tau_1, \tau_2)$ over X such that for all $1 \leq i \leq \ell$ we have, respectively:*

- *vertex-$\tau'[i]$ =* `unused`*,*
- *vertex-$\tau'[i] = v$, and*
- *vertex-$\tau'[i]$ =* `used`*.*

Since $\tau' = \mathit{join}(\tau_1, \tau_2)$ is a signature over the same set as τ_1 and differs from τ_1 only by replacing some `unused` markers with `used` markers, the join operation preserves feasibility, that is, if two joinable signatures τ_1 and τ_2 are feasible then so is $\tau' = \mathit{join}(\tau_1, \tau_2)$. An example of the join operation is given in Figure 4.11, where the top two signatures are joined to obtain the bottom one.

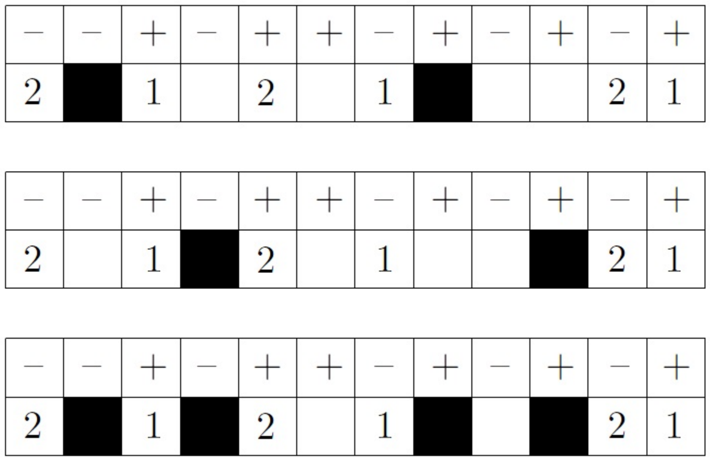

Figure 4.11: The join operation.

Dynamic programming algorithm

Let us now describe the algorithm. For each $i \in V(T)$ we assign an initially empty table A_i. All tables corresponding to internal nodes of T will be updated by simple applications of the introduce, forget, and join operations.

Leaf nodes. Let i be a leaf node, that is $X_i = \{v\}$ for some vertex v. We let $A_i = introduce(\tau, v)$, where τ is the signature with only `unused` markers.

Introduce nodes. Let j be the child of an introduce node i, that is $X_i = X_j \cup \{v\}$ for some $v \notin X_j$. We let $A_i = \bigcup_{\tau \in A_j} introduce(\tau, v)$.

Forget nodes. Let j be the child of a forget node i, that is $X_i = X_j \setminus \{v\}$ for some $v \in X_j$. We let $A_i = \{forget(\tau, v) \mid \tau \in A_j\}$.

Join nodes. Let j and h be the children of a join node i, that is $X_i = X_j = X_h$. We let $A_i = \{join(\tau_j, \tau_h) \mid \tau_j \in A_j, \tau_h \in A_h,$ and τ_j is joinable with $\tau_h\}$.

The algorithm accepts, i.e. we have a yes-instance, when A_{root} contains a signature τ with no `unused` markers. We now prove correctness of our dynamic programming algorithm in the following lemmas. First, we note that all signatures in the algorithm are obtained through join, forget, or introduce operations, which preserve feasibility and thus for each $i \in V(T)$, the table A_i contains only feasible signatures over X_i.

Lemma 4.5.34. *If a signature τ over X is obtained from a feasible signature by replacing all vertex markers not in X by* `used` *or* `unused` *markers, then τ is feasible.*

Proof. Let τ be obtained from a feasible signature τ' over X' by replacing all vertex markers in $X' \backslash X$ by `used` or `unused` markers. First note that $V(S_s \cap X, \tau[1,i]) = V(S_s \cap X', \tau'[1,i]) \cap X$. The first three conditions of Definition 4.5.29 follow immediately. As $G[X' \setminus S]$ being an independent set implies that $G[X \setminus (S \cap X)]$ is also an independent set, the fourth condition also follows. □

Lemma 4.5.35. *Let G be a graph S, X_1, and X_2 be subsets of $V(G)$ such that every edge of $G[X_1 \cup X_2]$ is contained in $G[X_1]$ or $G[X_2]$. If $S \cap X_1$ is a vertex cover of $G[X_1]$ and $S \cap X_2$ is a vertex cover of $G[X_2]$, then S is a vertex cover of $G[X_1 \cup X_2]$.*

Proof. Let uv be an edge of $G[X_1 \cup X_2]$. Then it is an edge of $G[X_i]$ for some $i \in \{1,2\}$. Hence one of u, v must be a member of $S \cap X_i$. Thus every edge of $G[X_1 \cup X_2]$ has an endpoint in S. □

Corollary 4.5.36. *Let τ, τ_1, and τ_2 be signatures over X, X_1, and X_2, respectively, such that $X = X_1 \cup X_2$ and every edge of $G[X]$ is contained in $G[X_1]$ or $G[X_2]$. Assume furthermore that for all $1 \le i \le \ell$:*

- *vertex-$\tau[i]$ = vertex-$\tau_1[i]$ whenever vertex-$\tau[i] \in X_1$ or vertex-$\tau_1[i] \in X_1$ and*
- *vertex-$\tau[i]$ = vertex-$\tau_2[i]$ whenever vertex-$\tau[i] \in X_2$ or vertex-$\tau_2[i] \in X_2$.*

Then if τ_1 and τ_2 are feasible, then so is τ.

Proof. The assumption means that τ and τ_1 agree over all changes within X_1, i.e. $V(S_s \cap X_1, \tau_1[1,i]) = V(S_s \cap X, \tau[1,i]) \cap X_1$ (and similarly for τ_2), for all $1 \le i \le \ell$. The first two conditions of Definition 4.5.29 for τ follow immediately: If vertex-$\tau[i] \in X$ then vertex-$\tau[i] \in X_1$ or vertex-$\tau[i] \in X_2$, so the statement is equivalent to the first two conditions for τ_1 or for τ_2. To show that the third condition holds for τ, observe that, for $1 \le i \le \ell$, $V(S_s \cap X, \tau[1,i]) = (V(S_s \cap X, \tau[1,i]) \cap X_1) \cup (V(S_s \cap X, \tau[1,i]) \cap X_2) = V(S_s \cap X_1, \tau_1[1,i]) \cup V(S_s \cap X_2, \tau_2[1,i]) = (S_t \cap X_1) \cup (S_t \cap X_2) = S_t \cap X$. For the last condition, it suffices to use Lemma 4.5.35 for $S = V(S_s \cap X, \tau[1,i])$. □

Lemma 4.5.37. *For $i \in V(T)$ and a signature τ over X_i, $\tau \in A_i$ if and only if τ can be extended to a signature over V_i that is feasible.*

Proof. We prove the statement by induction over the tree T. That is, we prove the statement to be true at $i \in V(T)$ assuming we have already proved it for all other nodes in the subtree of T rooted at i. Depending on whether i is a leaf, forget, introduce, or join node, we have the following cases.

Leaf nodes. Let v be the only vertex of X_i, i.e. $V_i = X_i = \{v\}$. Since $V_i = X_i$, a signature τ over X_i can be extended to a feasible signature over V_i if and only if τ is feasible and contains no `used` markers, i.e. if and only if τ has only `unused` and v markers and is feasible (over X_i), which happens if and only if $\tau \in A_i$.

Forget nodes. Let j be the child of i, thus $X_i = X_j \setminus \{v\}$ for some $v \in X_j$ and $V_i = V_j$.

For one direction, suppose $\tau \in A_i$ over X_i. Then there is a τ_j in A_j over X_j such that $\tau = forget(\tau_j, v)$. By the inductive assumption, τ_j has a feasible extension π over $V_j = V_i$. Since τ_j is be obtained from τ by replacing some `used` markers with v markers, π is also an extension of τ. Thus τ has a feasible extension over V_i.

For the other direction, suppose τ has a feasible extension π over V_i. Then π is obtained from τ by replacing some `used` markers with vertex markers from $V_i \setminus X_i$. Since $V_i \setminus X_i = (V_j \setminus X_j) \cup \{v\}$, we can consider the signature τ_j over $X_j \cup \{v\}$ obtained by only using the replacements with v steps. This signature τ_j can be extended to π by using the remaining replacements, so by the inductive assumption $\tau_j \in A_j$. Furthermore, $forget(\tau_j, v) = \tau$. Thus $\tau \in A_i$.

Introduce nodes. Let j be the child of i, thus $X_i = X_j \cup \{v\}$ for some $v \in X_i$ and $V_i = V_j \cup \{v\}$.

For one direction, suppose $\tau \in A_i$ is a signature over X_i. Then there is a $\tau_j \in A_j$ such that τ can be obtained from τ_j by replacing some `unused` markers with v markers. By the inductive assumption τ_j has a extension π_j over V_j that is feasible. As π_j can be obtained from τ_j by replacing `used` markers with vertex markers from $V_j \setminus X_j$ and τ has `used` markers at the same positions, we can use the same replacements to obtain an extension π over $V_j \cup \{v\}$ of τ. As π agrees with π_j over V_j and with τ over X_i, it is thus valid over V_i by Corollary 4.5.36. Therefore τ has an extension over V_i that is feasible.

For the other direction, suppose τ has an extension π over V_i that is feasible. Let π_j be the signature over $V_j = V_i \setminus \{v\}$ obtained by replacing all v markers of π with `unused` markers. By Lemma 4.5.34, π_j is feasible. Let τ_j be the signature over $X_j = X_i \setminus \{v\}$ obtained by replacing all v markers of τ with `unused` markers. Then since π_j is an extension of τ_j, $\tau_j \in A_j$ by the inductive assumption. Since π is feasible, so is τ (Lemma 4.5.34), and thus $\tau \in introduce(\tau_j, v)$ and $\tau \in A_i$.

Join nodes. Let j, h be the children of i, thus $V_i = V_j \cup V_h$ and we will write X for $X_i = X_j = X_h$.

For one direction suppose $\tau \in A_i$ is feasible over X. Then there are two joinable signatures $\tau_j \in A_j, \tau_h \in A_h$ such that $\tau = join(\tau_j, \tau_h)$. By the inductive assumption, they have feasible extensions, π_j over V_j and π_h over V_h, respectively. Let I_i, I_j, and I_h be the sets of indices of `used` markers in τ, τ_j, and τ_h, respectively. By Definition 4.5.33, I_i is the sum of disjoint sets I_j and I_h. Since π_j is obtained from τ_j by replacing markers at indices I_j with vertex markers from $V_j \setminus X$ and similarly for π_h, we can define a signature π obtained from τ over $X \cup (V_j \setminus X) \cup (V_h \setminus X) = V_i$ by using both sets of replacements. Signature π is an extension of τ. Moreover, π agrees with π_j over V_j and with π_h over V_h, so by Corollary 4.5.36, π is feasible over $V_j \cup V_h = V_i$. Therefore τ has a extension over V_i that is feasible.

For the other direction, suppose τ has an extension π over V_i that is feasible. Let I_j be the set of indices of vertex markers from $V_j \setminus X$ in π and define I_h accordingly. Let τ_j and π_j be obtained from τ and π by replacing all markers at indices I_h by `unused` markers. Since $V_j \cap V_h = X$, π_j is an extension of τ_j over V_j. By Lemma 4.5.34 π_j is feasible, thus by the inductive assumption $\tau_j \in A_j$. We define τ_h and π_h accordingly and observe that $\tau_h \in A_h$. It is easy to see that τ has `used` markers exactly at the indices $I_j \cup I_h$ and τ_j and τ_h have `used` markers exactly at the disjoint sets of indices I_j and I_h, respectively. This implies τ_j and τ_h are joinable and $\tau = join(\tau_j, \tau_h)$, so $\tau \in A_i$. □

Lemma 4.5.38. *Given a signature τ of length exactly ℓ, there exists a reconfiguration sequence σ of length exactly ℓ from S_s to S_t such that sign-τ = sign-σ if and only if A_{root} contains a signature over X_{root} that contains no* `unused` *markers.*

Proof. From Lemma 4.5.37, we know that A_{root} contains a signature τ over X_{root} with no `unused` markers if and only if there is a signature π over $V_{root} = V$ that is feasible and such that π contains no `unused` markers. This means that π contains only vertex markers and by definition of feasibility, the corresponding sequence S_s, $V(S_s, \pi[1,1])$, $V(S_s, \pi[1,2])$, $\ldots$, $V(S_s, \pi[1,\ell])$ is a reconfiguration sequence of length exactly ℓ from S_s to S_t such that the i^{th} step, $1 \leq i \leq \ell$, is a vertex removal marker if sign-$\sigma[i]$ = sign-$\tau[i] = -1$ and a vertex addition marker if sign-$\sigma[i]$ = sign-$\tau[i] = +1$. □

Theorem 4.5.39. VC-Bound *parameterized by ℓ is fixed-parameter tractable on graphs of bounded treewidth. Moreover, there exists an algorithm which solves the problem in $\mathcal{O}^*(4^\ell(t+3)^\ell)$ time for graphs of treewidth t.*

Proof. The correctness of our algorithm follows from Lemma 4.5.38. We now prove the bound on the running time of our algorithm. The number of nodes in T is in $\mathcal{O}(n)$. Checking the joinability and feasibility of signatures can be accomplished in time polynomial in

ℓ, t, and n. For each node $i \in V(T)$, the table A_i contains at most $(t+3)^\ell$ signatures. Updating tables at the leaf nodes requires $\mathcal{O}^*(2^\ell)$ time, since we check the feasibility of 2^ℓ signatures obtained from one introduce operation. In the worst case, updating the table of an introduce node requires $\mathcal{O}^*(2^\ell(t+3)^\ell)$ time, i.e. applying the introduce operation on each signature in a table of size $(t+3)^\ell$. For forget nodes, the time spent is polynomial in the maximum size of a table, i.e. $\mathcal{O}^*((t+3)^\ell)$. Finally, updating the table of a join node can be implemented in $\mathcal{O}^*(2^\ell(t+3)^\ell)$ time by checking for each of the $(t+3)^\ell$ possible signatures all possible ways to split `used` steps among the two children. The algorithm needs to be run for every signature of length at most ℓ that does not violate the maximum allowed capacity constraint, giving in total the claimed $\mathcal{O}^*(4^l(t+3)^\ell)$ time bound. □

Combining Theorem 4.5.39 with Proposition 2.4.9, we get:

Corollary 4.5.40. IS-Bound *parameterized by ℓ is fixed-parameter tractable on graphs of bounded treewidth. Moreover, there exists an algorithm which solves the problem in $\mathcal{O}^*(4^\ell(t+3)^\ell)$ time for graphs of treewidth t.*

4.5.4 Planar graphs

Using an adaptation of Baker's approach for decomposing planar graphs [7], also known as the *shifting technique* [12, 40, 50], we show, using Theorem 4.5.39, that VC-Bound (and hence IS-Bound) parameterized by ℓ remains fixed-parameter tractable on planar graphs. The general idea is that at most ℓ vertices of the input graph will be touched, and thus if we divide the graph into $\ell+1$ parts, one of these parts will contain only untouched vertices. The shifting technique allows the definition of the $\ell+1$ parts so that removing one (and replacing it with simple gadgets to preserve all needed information) yields a graph of treewidth at most $3\ell-1$.

Given a plane embedding of a planar graph G, the vertices of G are divided into layers $\{L_1, \ldots, L_r\}$ as follows: Vertices incident to the exterior face are in layer L_0. For $i \geq 0$, we let G' be the graph obtained by deleting all vertices in $L_0 \cup \ldots \cup L_i$ from G. All the vertices that are incident to the exterior face in G' are in layer L_{i+1} in G. L_r is thus the last non-empty layer. A planar graph that has an embedding where the vertices are in r layers is called *r-outerplanar*. The following result is due to Bodlaender [11].

Lemma 4.5.41 (Bodlaender [11])**.** *The treewidth of an r-outerplanar graph G is at most $3r-1$. Moreover, a tree decomposition of width at most $3r-1$ can be constructed in time polynomial in $|V(G)|$.*

From Lemma 4.5.41, we have the following corollary.

Corollary 4.5.42 ([11, 12]). *For a planar graph G, we let $\mathcal{E}$ be an arbitrary plane embedding of G and $\{L_1, \ldots, L_r\}$ be the collection of layers corresponding to $\mathcal{E}$. Then for any $i, \ell \geq 1$, the treewidth of the subgraph $G[L_{i+1} \cup \ldots \cup L_{i+\ell}]$ is at most $3\ell - 1$.*

We now summarize the main ideas behind how we use the shifting technique. Note that every vertex in $S_s \Delta S_t$ must be touched at least once in any reconfiguration sequence σ from S_s to S_t. In other words, $S_s \Delta S_t \subseteq V(\sigma)$. Moreover, we know that $|V(\sigma)|$ is at most ℓ, as otherwise the corresponding VC-Bound instance is a no-instance. For an arbitrary plane embedding of a planar graph G and every fixed $j \in \{0, \ldots, \ell\}$, we let G_j be the graph obtained by deleting all vertices in $L_{i(\ell+1)+j}$, for all $i \in \{0, 1, \ldots, \lfloor n/(\ell + 1) \rfloor\}$. Note that $tw(G_j) \leq 3\ell - 1$.

Proposition 4.5.43. *If there exists a reconfiguration sequence σ of length at most ℓ between two vertex covers S_s and S_t of a planar graph G, then for some fixed $j \in \{0, \ldots, \ell\}$ we have $V(\sigma) \subseteq V(G_j)$.*

We still need a few gadgets before we can apply Theorem 4.5.39 on each graph G_j and guarantee correctness. In particular, we need to handle deleted vertices and "border" vertices correctly, i.e. vertices incident to the exterior face in G_j.

We solve at most $\lfloor n/(\ell + 1) \rfloor + 1$ instances of the VC-Bound problem as shown in Algorithm 5. We use DP-VC-Bound to denote the dynamic programming algorithm whose existence was shown in Theorem 4.5.39.

On lines 5 and 6, we make sure that no vertices from the symmetric difference of S_s and S_t lie in the deleted layers of G, as otherwise G^*_j can be ignored, by Proposition 4.5.43. Hence, we know that D^*_j can only include vertices common to both S_s and S_t (vertices in $S_s \cap S_t$) and we can partition D^*_j into two sets accordingly (line 7). In the remaining steps, we add gadgets to account for the capacity used by vertices in A^*_j and the fact that the neighbors of any vertex in B^*_j must remain untouched. In other words, we assume that there exists a reconfiguration sequence σ of length at most ℓ from $S^*_{s,j}$ to $S^*_{t,j}$ in $R_{VC}(G^*_j, 0, k)$. Then σ is a reconfiguration sequence of length at most ℓ from S_s to S_t in $R_{VC}(G, 0, k)$ only if:

(1) $|S^*_{s,j}| + cap(\alpha) \leq k - |A^*_j|$ and

(2) no vertex deletion in α leaves an edge uncovered in G.

Algorithm 5 PLANAR-VC-BOUND

```
Input: A planar graph $G$, positive integers $k$ and $\ell$, and two vertex covers $S_s$ and $S_t$ of $G$
    of size at most $k$.
Output: A "yes-instance" if and only if there exists a reconfiguration of length at most $\ell$
    from $S_s$ to $S_t$ and a "no-instance" otherwise.
1: Find an arbitrary plane embedding of $G$;
2: for each $j \in \{0, \dots, \ell\}$
3:     Let $G^*_j = G_j$;
4:     Let $D^*_j$ denote the set of vertices deleted from $G$ to obtain $G^*_j$;
5:     if $(\{S_s \Delta S_t\} \cap D^*_j \neq \emptyset)$
6:         continue (move on to the next value of $j$);
7:     Partition $D^*_j$ into $A^*_j = D^*_j \cap \{S_s \cap S_t\}$ and $B^*_j = D^*_j \setminus A^*_j$;
8:     Let $S^*_{s,j} = S_s \cap V(G^*_j)$ and $S^*_{t,j} = S_t \cap V(G^*_j)$;
9:     if $(\{v \in S^*_{s,j} \Delta S^*_{t,j} \mid |N_G(v) \cap B^*_j| > 0\} \neq \emptyset)$
10:        continue (move on to the next value of $j$);
11:    for each $v \in A^*_j$
12:        Add an $(\ell + 1)$-star centered at $u$ to $G^*_j$;
13:        Add $u$ to $S^*_{s,j}$ and $S^*_{t,j}$;
14:    for each $\{v \in \{S^*_{s,j} \cap S^*_{t,j}\} \mid |N_G(v) \cap B^*_j| > 0\}$
15:        Add $\ell + 1$ degree-one neighbors to $v$ in $G^*_j$;
16:    temp = DP-VC-BOUND($G^*_j, S^*_{s,j}, S^*_{t,j}, k, \ell$);
17:    if (temp = "yes-instance")
18:        return "yes-instance";
19: return "no-instance";
```

To guarantee property (1), we add an $(\ell+1)$-star to G^*_j for every vertex in A^*_j and then add the center of the star to both $S^*_{s,j}$ and $S^*_{t,j}$ (lines 11, 12, and 13). Therefore, for every value of j we have $|S| = |S^*_{s,j}|$, $|T| = |S^*_{t,j}|$, and $|S^*_{s,j}| + cap(\alpha) \leq k - |A^*_j|$. For property (2), we add $\ell + 1$ degree-one neighbors to every vertex in $\{v \in \{S^*_{s,j} \cap S^*_{t,j}\} \mid |N_G(v) \cap B^*_j| > 0\}$ (lines 14 and 15). Those vertices, as well as the centers of the stars, will have to remain untouched in σ, as otherwise deleting any such vertex would require more than ℓ additions.

Since adding degree-one vertices and $(\ell + 1)$-stars to a graph does not increase its treewidth, we have $tw(G^*_j) \leq 3\ell - 1$ for all j (Corollary 4.5.42). Hence, for each graph G^*_j we can now apply Theorem 4.5.39 and solve the VC-BOUND instance $(G^*_j, S^*_{s,j}, S^*_{t,j}, k, \ell)$ in $\mathcal{O}^*(4^\ell(3\ell + 1)^\ell)$ time. We prove in Lemma 4.5.44 that our original instance on planar G is a yes-instance if and only if $(G^*_j, S^*_{s,j}, S^*_{t,j}, k, \ell)$ is a yes-instance for some fixed $j \in \{0, 1, \dots, \lfloor n/(\ell + 1)\rfloor\}$.

Lemma 4.5.44. *(G, S_s, S_t, k, ℓ) is a yes-instance of* VC-Bound *if and only if $(G^*_j, S^*_{s,j}, S^*_{t,j}, k, \ell)$ is a yes-instance for some fixed $j \in \{0, 1, \ldots, \lfloor n/(\ell+1) \rfloor\}$.*

Proof. For (G, S_s, S_t, k, ℓ) a yes-instance of VC-Bound, there exists a reconfiguration sequence σ of length at most ℓ from S_s to S_t. Then by Corollary 4.5.43, we know that for some fixed $j \in \{0, 1, \ldots, \lfloor n/(\ell+1) \rfloor\}$ we have $V(\sigma) \subseteq V(G^*_j)$ and $V(\sigma) \cap N_G(B^*_j) = \emptyset$, as otherwise $V(\sigma) \cap B^*_j \neq \emptyset$. By our construction of G^*_j, the maximum capacity constraint is never violated. Therefore, σ is also a reconfiguration sequence from $S^*_{s,j}$ to $S^*_{t,j}$.

For the converse, suppose that $(G^*_j, S^*_{s,j}, S^*_{t,j}, k, \ell)$ is a yes-instance for some fixed $j \in \{0, 1, \ldots, \lfloor n/(\ell+1) \rfloor\}$ and let σ denote the corresponding reconfiguration sequence from $S^*_{s,j}$ to $S^*_{t,j}$. Since the maximum capacity constraint will not be violated, we only need to make sure that (i) no reconfiguration step in σ leaves an uncovered edge in G and that (ii) none of the degree-one gadget vertices are in $V(\sigma)$. For (ii), it is not hard to see that any such vertex must be touched an even number of times and hence we can delete those reconfiguration steps to obtain a shorter reconfiguration sequence. For (i), assume that σ leaves an uncovered edge in G. By our construction of G^*_j, such an edge must have one endpoint in B^*_j. But since we added $\ell + 1$ degree-one neighbors to every vertex in the neighborhood of B^*_j, this is not possible. □

Theorem 4.5.45 follows by combining Proposition 2.4.9, Lemma 4.5.41, Lemma 4.5.44, Theorem 4.5.39, and the fact that $tw(G^*_j) \leq 3\ell - 1$, for all $j \in \{0, 1, \ldots, \lfloor n/(\ell+1) \rfloor\}$.

Theorem 4.5.45. VC-Bound *and* IS-Bound *are fixed-parameter tractable on planar graphs when parameterized by ℓ. Moreover, there exists an algorithm which solves both problems in $\mathcal{O}^*(4^\ell(3\ell+2)^\ell)$ time.*

We note that, by a simple application of the following result of Demaine et al. [37], Theorem 4.5.45 generalizes to H-minor-free graphs (Figure 2.2) and only the constants of the overall running time of the algorithm are affected.

Theorem 4.5.46. *[37] For a fixed graph H, there exists a constant c_H such that for any integer $\ell > 1$ and for every H-minor-free graph G, the vertices of G can be partitioned into $\ell + 1$ sets such that any ℓ of the sets induce a graph of treewidth at most $c_H\ell$. Moreover, such a partition can be found in time polynomial in $|V(G)|$.*

Corollary 4.5.47. VC-Bound *and* IS-Bound *are fixed-parameter tractable on H-minor-free graphs when parameterized by ℓ.*

Chapter 5

Identifying tractable instances

In this last chapter on reachability and bounded reachability variants of subset problems, we generalize some of the earlier techniques we have seen and provide additional examples of how they can be applied. Moreover, we present a set of "tools" which can serve as a starting point for studying the parameterized complexity of almost any reconfiguration problem where feasible solutions correspond to subsets of a finite domain $\mathcal{D}$.

In Section 5.1, we introduce the notion of a reconfiguration kernel and apply it to show that both BHS-Reach and BHS-Bound parameterized by k are fixed-parameter tractable. This provides an alternative proof of Theorem 4.5.2 and states that whenever we can enumerate all minimal solutions "efficiently", then we can solve the reachability and bounded reachability problems efficiently. In other words, we can do reconfiguration via enumeration. For some problems, Feedback Vertex Set being one example, although the problem is fixed-parameter tractable parameterized by solution size, the number of solutions could still be exponential in the input size (not the parameter) and no "efficient" enumeration algorithms are possible. In this case, we show how to use the notion of compact representations of minimal solutions in order to solve the reachability and bounded reachability variants of Feedback Vertex Set. What we show is in fact more general. We show that we can efficiently solve the reachability and bounded reachability variants of any subset problem which satisfies certain properties and admits a compact representation that can be computed efficiently. We present this result in Section 5.2.

As the reader might have noticed, our dynamic programming algorithm for VC-Bound from Section 4.5.3 makes very little use of the fact that feasible solutions correspond to vertex covers of the graph. We formalize this phenomenon in Section 5.3 by showing that a host of known dynamic programming algorithms can be modified to solve the bounded reachability version of a problem. We use the Odd Cycle Transversal problem as an example. Moreover, we provide another meta-theorem, this time a positive one, which clearly captures the relationship between finding a single solution for a problem and finding a sequence of at most ℓ solutions such that the size of the symmetric difference of every two consecutive solutions is small. That is, we show, using Courcelle's celebrated result [30], that $\mathcal{Q}$-Bound parameterized by ℓ is fixed-parameter tractable on graphs of bounded treewidth for any vertex-subset problem $\mathcal{Q}$ expressible in monadic second-order logic.

In Section 5.4 we revisit irrelevant vertices and prove that the reachability variant of Independent Set is fixed-parameter tractable on graphs of bounded degeneracy and nowhere-dense graphs, generalizing recent results of Ito et al. [83, 84] who showed that the problem is fixed-parameter tractable on graphs of bounded degree, planar graphs, and $K_{3,d}$-free graphs, for some constant d (Figure 2.2). The complexity of the problem remains open for biclique-free graphs, which we believe to be an interesting problem to study for the following reason. The results we have seen so far suggest a pattern relating a problem and its reachability variant: whenever a problem that is W-hard in general is fixed-parameter tractable parameterized by solution size on a graph class $\mathcal{C}$, then the reachability version of the problem (with the same parameter) is also fixed-parameter tractable on $\mathcal{C}$. We think that either proving or disproving this pattern would be quite interesting; IS-Reach on biclique-free graphs is the best contender we know of so far.

5.1 Reconfiguration via enumeration

Recall that a problem is in FPT if and only if it has a kernel (Definition 2.3.3). We introduce the related notion of a *reconfiguration kernel*; it follows from the definition that a reconfiguration problem that has such a kernel is in FPT.

Definition 5.1.1. *A* reconfiguration kernel *of an instance (x, p) of a parameterized reachability or bounded reachability problem, where x is the input and p the parameter, is a set of $f(p)$ instances, for a computable function f, such that for $1 \leq i \leq f(p)$:*

- *for each instance (x_i, p_i) in the set, p_i can be computed in polynomial time,*
- *the size of each x_i is bounded by $g(p)$, for a computable function g, and*
- *(x, p) is a yes-instance if and only if at least one (x_i, p_i) is a yes-instance.*

We prove the parameterized tractability of reachability and bounded reachability variants of certain superset-closed k-subset problems when parameterized by k; a *k-subset problem* is a parameterized minimization problem $\mathcal{Q}$ whose solutions for an instance $(\mathcal{I}, k)$ are all subsets of size at most k of a finite domain set $\mathcal{D}$. We say $\mathcal{Q}$ is *superset-closed* if any superset of size at most k of a solution of $\mathcal{Q}$ is also a solution of $\mathcal{Q}$. For example, VERTEX COVER is a superset-closed problem but INDEPENDENT SET is not.

Theorem 5.1.2. *If a k-subset problem $\mathcal{Q}$ is superset-closed and has an* FPT *algorithm to enumerate all its minimal solutions of cardinality at most k, the number of which is bounded by a function of k, then $\mathcal{Q}$-*REACH *and $\mathcal{Q}$-*BOUND *parameterized by k are in* FPT.

Proof. By enumerating all minimal solutions of $\mathcal{Q}$ of cardinality at most k, we compute the set M of all elements v of the domain set such that v is in a minimal solution to Q. For $(\mathcal{I}, S_s, S_t, k, \ell)$ an instance of $\mathcal{Q}$-BOUND, we show that there exists a reconfiguration sequence from S_s to S_t if and only if there exists a reconfiguration sequence from $S_s \cap M$ to $S_t \cap M$ that touches only elements of M.

Each set U in the reconfiguration sequence from S_s to S_t is a solution, and hence contains at least one minimal solution in $U \cap M$; $U \cap M$ is a superset of the minimal solution and hence also a solution. Moreover, since any two consecutive solutions U and U' in the sequence differ by a single element, $U \cap M$ and $U' \cap M$ differ by at most a single element. By replacing each subsequence of identical sets by a single set, we obtain a reconfiguration sequence from $S_s \cap M$ to $S_t \cap M$ that uses only subsets of M.

The reconfiguration sequence from $S_s \cap M$ to $S_t \cap M$ using only subsets of M can be extended to a reconfiguration sequence from S_s to S_t by transforming S_s to $S_s \cap M$ in $|S_s \setminus M|$ steps and transforming $S_t \cap M$ to S_t in $|S_t \setminus M|$ steps. Note that this might not be a shortest reconfiguration sequence from S_s to S_t. In this sequence, each vertex in $C_{st} \setminus M$ is removed from S_s to form $S_s \setminus M$ and then readded to form S_t from $S_t \setminus M$. For each vertex $v \in C_{st} \setminus M$, we can choose instead to add v to each solution in the sequence, thereby decreasing ℓ by two (the steps needed to remove and then readd v) at the cost of increasing by one the capacity used in the sequence from $S_s \cap M$ to $S_t \cap M$.

This choice can be made independently for each of these $\mathcal{E} = |C_{st} \setminus M|$ vertices. Consequently, $(\mathcal{I}, S_s, S_t, k, \ell)$ is a yes-instance for $\mathcal{Q}$-BOUND if and only if one of the $\mathcal{E} + 1$ reduced instances $(\mathcal{I}, S_s \cap M, S_t \cap M, k - e, \ell - 2(\mathcal{E} - e))$, for $0 \leq e \leq \mathcal{E}$ and $\mathcal{E} = |C_{st} \setminus M|$, is a yes-instance for $\mathcal{Q}'$-BOUND, where we define $\mathcal{Q}'$ as a k-subset problem whose solutions for an instance $(\mathcal{I}, k)$ are solutions of instance $(\mathcal{I}, k)$ of $\mathcal{Q}$ that are contained in M.

To show that $\mathcal{Q}'$-BOUND is in FPT, we observe that the number of nodes in the reconfiguration graph for $\mathcal{Q}'$ is bounded by a function of k. Each solution of $\mathcal{Q}'$ is a subset of

M, yielding at most $2^{|M|}$ nodes, and $|M|$ is bounded by a function of k. □

Corollary 5.1.3. BHS-Reach *and* BHS-Bound *parameterized by k are in* FPT.

Proof. Bounded Hitting Set is superset-closed. Furthermore, standard techniques give an FPT algorithm to enumerate all minimal solutions of a Bounded Hitting Set instance, and the number of minimal solutions is bounded by a function of k, as required by Theorem 5.1.2. We include the proof for completeness.

We can devise a search-tree algorithm that gradually constructs minimal hitting sets, producing all minimal hitting sets of size at most k in its leaves. Consider an instance of Bounded Hitting Set, where the cardinality of each set is bounded by a constant c. At each non-leaf node in the search tree, the algorithm chooses a set that is not hit, and branches on all possible ways of hitting this set, including one of the (at most c) elements in the set in each branch. Since we are not interested in solutions of cardinality more than k, we do not need to search beyond depth k in the tree, proving an upper bound of c^k on the number of leaves, and an upper bound of $O^*(c^k)$ on the enumeration time. □

If we restrict our attention to only Bounded Hitting Set, the proof of Theorem 5.1.2 can be strengthened to develop a polynomial reconfiguration kernel. In fact, we use the ideas in Theorem 5.1.2 to adapt a special kernel that retains all minimal hitting sets of size at most k in the reduced instances [33].

Theorem 5.1.4. BHS-Reach *and* BHS-Bound *parameterized by k admit a polynomial reconfiguration kernel.*

Proof. We let (G, S_s, S_t, k, ℓ) be an instance of BHS-Bound: Here G is a family of sets of vertices of size at most c and each of S_s and S_t is a hitting set of size at most k, that is, a set of vertices intersecting each set in G. In other words, G is a hypergraph.

We form a reconfiguration kernel using the polynomial-time reduction algorithm $\mathcal{A}$ of Damaschke and Molokov [33]: $G' = \mathcal{A}(G)$ contains all minimal hitting sets of size at most k, and is of size at most $(c-1)k^c + k$.

$V(G')$ includes all minimal k-hitting sets, and the k-hitting sets for G' are actually those k-hitting sets for G that are completely included in $V(G')$. Therefore, as in the proof of Theorem 5.1.2, (G, S_s, S_t, k, ℓ) is a yes-instance for BHS-Bound if and only if one of the $\mathcal{E}+1$ reduced instances $(G', S_s \cap V(G'), S_t \cap V(G'), k-e, \ell - 2(\mathcal{E}-e))$, for $0 \le e \le \mathcal{E}$, is a yes-instance.

Notice that unlike in the proof of Theorem 5.1.2, here the set containing all minimal solutions can be computed in polynomial time, whereas Theorem 5.1.2 guarantees only a fixed-parameter tractable procedure. □

Bounded Hitting Set generalizes any deletion (minimization) problem for a hereditary property with a finite forbidden set:

Corollary 5.1.5. *If* Π *is a hereditary graph property with a finite forbidden set, then* Π-Del-Reach *and* Π-Del-Bound *parameterized by* k *admit a polynomial reconfiguration kernel.*

5.2 Reconfiguration via compact representations

The Feedback Vertex Set problem is a k-subset problem that is superset closed. However, Theorem 5.1.2 does not apply since the number of minimal feedback vertex sets of a graph of size at most k could be exponential in n. Moreover, the forbidden set is infinite for Π the collection of all forests, as it consists of all cycles. Hence, neither Theorem 5.1.4 nor Corollary 5.1.5 apply in this case.

Guo et al. [71], who showed that FVS is fixed-parameter tractable parameterized by solution size, introduced the notion of compact representations of minimal feedback vertex sets. We generalize this notion to k-subset problems and show that we can in fact replace the enumeration of minimal solutions in Theorem 5.1.2 with the use of compact representations.

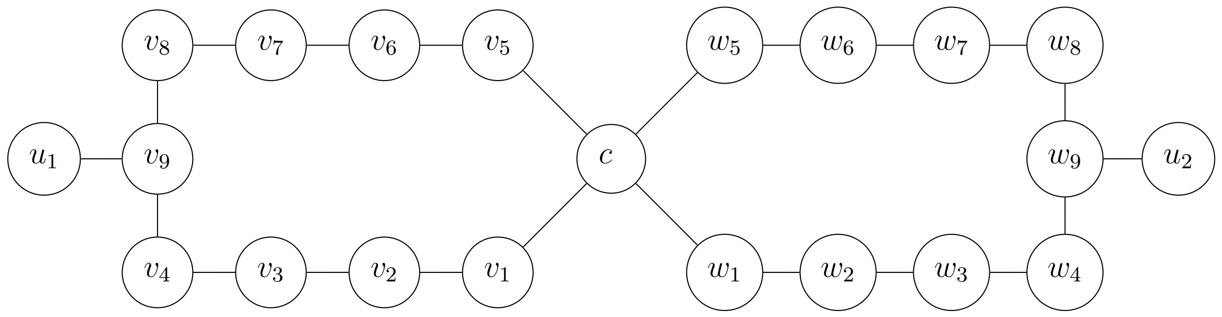

Figure 5.1: Example of a graph with no minimal feedback vertex set of size three.

Before introducing the formal definitions, we illustrate an example of a compact representation of minimal feedback vertex sets of size at most $k = 2$ of the graph G shown in Figure 5.1. Clearly, the only minimal feedback vertex set of size one is the vertex c and vertices u_1 and u_2 do not participate in any minimal feedback vertex set of G. However, any pair of vertices (v_i, w_j), $1 \leq i, j \leq 9$, is a minimal feedback vertex set of size two. Therefore, if the two cycles in G were each of size n, the total number of possible minimal feedback vertex sets of size two is a function of n, not a function of k. Instead of enumerating all those solutions, we can "compactly" represent them using only two sets $B_1 = \{v_1, \ldots, v_9\}$ and $B_2 = \{w_1, \ldots, w_9\}$, which we call *equivalence sets*. The general idea is that B_1 and B_2 are disjoint and picking one vertex from each set always produces a minimal solution of size two, i.e. vertices in the same equivalence set are "equivalent", and each minimal feedback vertex set of G of size two is contained in those sets. Hence, a compact representation of all minimal feedback vertex sets of G of size at most two can be described by:

$$\mathcal{C}_{\text{FVS}}(G, 2) = \left\{ \begin{array}{l} \mathcal{C}_2 = \{B_1 = \{v_1, \ldots, v_9\}, B_2 = \{w_1, \ldots, w_9\}\} \\ \mathcal{C}_1 = \{B_1 = \{c\}\} \end{array} \right\} \tag{5.1}$$

That is, G has a unique minimal feedback vertex of size one, the vertex c, and every minimal feedback vertex set of size two can be obtained by picking exactly one vertex from each equivalence set in the *collection* $\mathcal{C}_2$.

Generally, given an instance $(\mathcal{I}, k)$ of a k-subset problem $\mathcal{Q}$, we consider the multiset $\mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$, i.e. a family of collections of subsets. Each collection $C_i^j \in \mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$, $1 \leq i \leq k$ and $j \geq 1$, consists of exactly i pairwise disjoint subsets $\{B_1, \ldots, B_i\}$ of the domain $\mathcal{D}$. A "visual representation" of $\mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$ is shown in Equation 5.2.

Definition 5.2.1. *We say a k-subset problem $\mathcal{Q}$ admits a* compact representation *if for each instance $\mathcal{I}$ there exists a multiset $\mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$ satisfying the following properties:*

(1) For any minimal solution S of size at most k, there exists a collection $\mathcal{C}_i^j \in \mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$, $1 \leq i \leq k$ and $j \geq 1$, such that $S \cap B_p \neq \emptyset$, for all $1 \leq p \leq i$ and $B_p \in \mathcal{C}_i^j$. We say S is represented *by the collection $\mathcal{C}_i^j$ and we write $S \sqsubset \mathcal{C}_i^j$.*

(2) For any collection $\mathcal{C}_i^j \in \mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$, $1 \leq i \leq k$ and $j \geq 1$, if we pick exactly one vertex from each set $B \in \mathcal{C}_i^j$ to form the set S, then S is a minimal solution of size i.

(3) $|\mathcal{C}_\mathcal{Q}(\mathcal{I}, k)| \leq f(k)$ and $\mathcal{C}_\mathcal{Q}(\mathcal{I}, k)$ can be computed in $O^*(g(k))$ time, for some computable functions f and g.

$$\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k) = \left\{ \begin{array}{l} \mathcal{C}_k^1 = \{B_1, \ldots, B_k\} \\ \mathcal{C}_k^2 = \{B_1, \ldots, B_k\} \\ \ldots \\ \mathcal{C}_{k-1}^1 = \{B_1, \ldots, B_{k-1}\} \\ \mathcal{C}_{k-1}^2 = \{B_1, \ldots, B_{k-1}\} \\ \ldots \\ \mathcal{C}_{k-2}^1 = \{B_1, \ldots, B_{k-2}\} \\ \mathcal{C}_{k-2}^2 = \{B_1, \ldots, B_{k-2}\} \\ \ldots \\ \mathcal{C}_1^1 = \{B_1\} \end{array} \right\} \tag{5.2}$$

Note that the size of individual equivalence sets need not be bounded by a function of k, as otherwise generating compact representations amounts to enumerating all minimal solutions. In other words, problems such as Bounded Hitting Set trivially admit a compact representation where each collection $\mathcal{C}$ corresponds to a minimal solution and each equivalence set $B \in \mathcal{C}$ contains exactly one element from the domain. Moreover, equivalence sets in different collections are not related (they might intersect). For a set S such that $S \sqsubset \mathcal{C}$, we let $S|\mathcal{C}$ denote the *restriction* of S to $\mathcal{C}$, i.e. $S|\mathcal{C}$ is of size $|\mathcal{C}|$ and contains exactly one vertex from each equivalence set in $\mathcal{C}$ (picked arbitrarily). We say two (not necessarily minimal) solutions S and S' are *equivalent* with respect to $\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$, $S \equiv S'$, if there exists a collection $\mathcal{C} \in \mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$ such that $S, S' \sqsubset \mathcal{C}$. We now discuss why compact representations are helpful when dealing with reachability and bounded reachability problems.

Lemma 5.2.2. *If $S, S' \in R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ and $S \equiv S'$ then either S and S' are isolated nodes in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ or there exists a reconfiguration sequence between the two whose length is at most $4k$.*

Proof. Since $S \equiv S'$, we know that there exists a collection $\mathcal{C}_i \in \mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$, $|\mathcal{C}_i| \leq k$ and $1 \leq i \leq k$, such that for each set $B_j \in \mathcal{C}_i$, $1 \leq j \leq i$, we have $S \cap B_j \neq \emptyset$ and $S' \cap B_j \neq \emptyset$. If $|\mathcal{C}_i| = i = k$, then $|S| = |S'| = k$ and both S and S' are minimal solutions and their corresponding nodes in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ have degree zero, i.e. both are isolated. Moreover, if either S or S' is minimal and of size k then clearly $|\mathcal{C}_i| = k$ and again both are isolated nodes in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$. We claim that in all other cases, we can always find a reconfiguration sequence of length at most $4k$ between S and S'.

To see why, consider the sets $S|\mathcal{C}_i$ and $S'|\mathcal{C}_i$. Both of these sets have size exactly $i < k$ and are minimal solutions (Definition 5.2.1). Therefore, it is possible to add an element to S without violating the maximum capacity. Now consider any set $B_j \in \mathcal{C}_i$, $1 \leq j \leq i$, such that $(S|\mathcal{C}_i) \cap B_j \neq (S'|\mathcal{C}_i) \cap B_j$. If no such set exists then $S|\mathcal{C}_i = S'|\mathcal{C}_i$. Otherwise, we modify $S|\mathcal{C}_i$ by adding the element in $(S'|\mathcal{C}_i) \cap B_j$ and removing the element in $(S|\mathcal{C}_i) \cap B_j$. Repeating this procedure for every B_j transforms $S|\mathcal{C}_i$ to $S'|\mathcal{C}_i$ in at most $2k$ steps. Reaching $S|\mathcal{C}_i$ from S and S' from $S'|\mathcal{C}_i$ can be accomplished in at most $2k$ steps as well; k removals for the former and k additions for the latter case. □

Lemma 5.2.3. *If a k-subset problem $\mathcal{Q}$ admits a compact representation then the diameter of each connected component of $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ is bounded above by a function of k.*

Proof. We let $\sigma = \langle S_0, \ldots, S_\ell \rangle$ denote a reconfiguration sequence in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$. Every set S_i, $0 \leq i \leq \ell$, contains at least one minimal solution and can therefore be represented by at least one collection in $\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$. Hence, if $\ell \geq (4k+1)|\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)|$ then there exist at least $4k+1$ sets in σ which can be represented by the same collection in $\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$. We let S_p and S_q, $0 \leq p < q \leq \ell$, denote the first and last such sets in σ. By Lemma 5.2.2, there exists a reconfiguration sequence between S_p and S_q whose length is at most $4k$, as S_p and S_q are not isolated nodes in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$. Therefore, there exists a reconfiguration sequence σ' from S_0 to S_ℓ whose length is at least one less than $|\sigma|$. As the size of $\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$ is bounded by a function of k, we get the desired result. □

Having established a bound on the length of any reconfiguration sequence, we can now consider the bounded reachability problem parameterized by $k + \ell$, as ℓ is bounded by a function of k. For instance, the treewidth of any graph that has a feedback vertex set of size k is at most k. If FVS-Bound parameterized by ℓ were fixed-parameter tractable on graphs of bounded treewidth, Lemma 5.2.3 immediately implies that FVS-Bound parameterized by k is also fixed-parameter tractable. To see why, assume an algorithm which solves FVS-Bound in $\mathcal{O}^*(f(\ell, tw(G)))$ time existed, for some computable function f. Then the same algorithm would solve the general problem in $\mathcal{O}^*(f(g(k), k))$ time, where g is the function whose existence was shown in Lemma 5.2.3. We give such an algorithm for FVS-Bound in the next section. For now, we prove a somewhat stronger result.

Theorem 5.2.4. *If a k-subset problem $\mathcal{Q}$ admits a compact representation then $\mathcal{Q}$-*Reach *and $\mathcal{Q}$-*Bound *parameterized by k are fixed-parameter tractable.*

Proof. We let $\mathcal{D}^+ = \mathcal{D} \cap \bigcup_{\mathcal{C} \in \mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)} \bigcup_{B \in \mathcal{C}} B$ denote the set of all elements in the domain that appear in some equivalence set of the compact representation. Using the same arguments made in the proof of Theorem 5.1.2, it is not hard to see that, for an instance $(\mathcal{I}, S_s, S_t, k, \ell)$ of $\mathcal{Q}$-Bound, there exists a reconfiguration sequence from S_s to S_t of length at most ℓ in

$R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ if and only if there exists a reconfiguration sequence of length at most ℓ that touches only elements of $S_s \cup S_t \cup \mathcal{D}^+$. However, as the size of $\mathcal{D}^+$ need not be bounded by a function of k, our goal is to construct a set $\mathcal{D}^-$ of bounded size.

To that end, we construct a bipartite graph G^+ with bipartition (X^+, Y^+). For every element in $\mathcal{D}^+$ there exists a corresponding vertex in X^+ and for every equivalence set B in some collection $\mathcal{C}$ there exists a corresponding vertex in Y^+. We add an edge between a vertex $u \in X^+$ and a vertex $v \in Y^+$ whenever the element corresponding to u is in the equivalence set corresponding to v. The size of Y^+ is at most $k|\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)|$. We partition the vertices of X^+ into at most $2^{k|\mathcal{C}_{\mathcal{Q}}(\mathcal{I},k)|}$ subsets, where two vertices $u, v \in X^+$ belong to partition P_Z if and only if $N_{G^+}(u) = N_{G^+}(v) = Z \subseteq Y^+$. We construct the graph G^-, with bipartition (X^-, Y^-), by deleting from X^+ all but k vertices from every partition of X^+ of size greater than k and we let Y^- be a copy of Y^+. Note that the total number of vertices in G^- is at most $|Y^-| + |X^-| \leq k|\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)| + k2^{k|\mathcal{C}_{\mathcal{Q}}(\mathcal{I},k)|}$. Finally, we let $\mathcal{D}^-$ consist of all elements having corresponding vertices in X^- and we delete from the compact representation $\mathcal{C}_{\mathcal{Q}}(\mathcal{I}, k)$ all elements which are not in $\mathcal{D}^-$ to obtain $\mathcal{C}^-_{\mathcal{Q}}(\mathcal{I}, k)$. $\mathcal{C}^-_{\mathcal{Q}}(\mathcal{I}, k)$ might no longer be a compact representation as property (1) of Definition 5.2.1 might not be satisfied. We show however that $\mathcal{C}^-_{\mathcal{Q}}(\mathcal{I}, k)$ retains enough information in order to solve the bounded reachability problem. That is, we show that if there exists a reconfiguration sequence from S_s to S_t of length at most ℓ in $R_{\mathcal{Q}}(\mathcal{I}, 0, k)$ then there exists a reconfiguration sequence of length at most ℓ that touches only elements of $S_s \cup S_t \cup \mathcal{D}^-$ (the other direction trivially holds).

We simulate any reconfiguration sequence from S_s to S_t on the bipartite graph G^+ in the following sense. Since every element in $\mathcal{D}^+$ has a corresponding vertex in X^+, any set S in a reconfiguration reconfiguration sequence from S_s to S_t corresponds to a subset of the vertices in $X^+ \cup S_s \cup S_t$. Moreover, as $|S| \leq k$, S intersects each partition P_Z of X^+, $Z \subseteq Y^+$, in at most k elements. Therefore, combining the fact that X^- retains k elements from each partition (when they exist) and that those elements belong to the same equivalence sets with the definition of compact representations (i.e. vertices in the same equivalence sets are "equivalent"), we can simulate any reconfiguration sequence from S_s to S_t on the bipartite graph G^-. Since the number of vertices in G^- is bounded by a function of k, the statement of the theorem follows. □

An interesting consequence of Theorem 5.2.4 is that it allows us to assert the non-existence of compact representations for problem $\mathcal{Q}$ by showing that either $\mathcal{Q}$-Reach or $\mathcal{Q}$-Bound are W-hard parameterized by k. This is trivially true for problems such as Dominating Set but for some problems, e.g. Odd Cycle Transversal, we do not know whether they admit compact representations. Another challenging problem would be to determine whether the converse of Theorem 5.2.4 also holds, i.e. if being able to solve reconfiguration efficiently implies the existence of a compact representation. Going back to the Feedback Vertex Set problem, Guo et al. [71] proved the following theorem.

Theorem 5.2.5. *[71]* FVS *admits a compact representation.*

Combining Theorems 5.2.4 and 5.2.5, we obtain:

Corollary 5.2.6. FVS-Reach *and* FVS-Bound *parameterized by k are fixed-parameter tractable.*

5.3 Reconfiguration over tree decompositions

5.3.1 Dynamic programming algorithms

In this section we show how known dynamic programming algorithms [12] for problems on graphs of treewidth at most t can be adapted to bounded reachability problems parameterized by ℓ. We reuse the notions of signatures and operations defined on edit sequences defined in Section 4.5.3 and 4.1, respectively. The general idea is to maintain a view of the reconfiguration sequence just as we did for VC-Bound (Section 4.5.3) and in addition check if every reconfiguration step gives a solution, which can be accomplished by maintaining (independently for each step) any information that the original algorithm would maintain. We present the details for OCT-Bound as an example.

In a dynamic programming algorithm for VC on graphs of bounded treewidth, it is enough to maintain information about how the solution intersects with a bag of the tree decomposition. This is not the case for OCT. One algorithm for OCT runs in $\mathcal{O}^*(3^t)$ time by additionally maintaining a bipartition of the bag (with the solution deleted) [59, 62]. That is, at every bag X_i, $i \in V(T)$, we maintain a list of assignments $X_i \to \{\texttt{used}, \texttt{left}, \texttt{right}\}$ with the property that there exists a subset S of V_i and a bipartition (L, R) of $G[V_i \setminus S]$ such that $X_i \cap S$, $X_i \cap L$, and $X_i \cap R$ are the `used`, `left`, and `right` vertices, respectively. A signature for OCT-Bound will hence additionally store a bipartition for each step (except for the first and last sets S_s and S_t, as we already assume them to be solutions).

Definition 5.3.1. *An* OCT-signature τ *over a set* $X \subseteq V(G)$ *is an edit sequence* τ *such that vertex-*$\tau[i] \in X \cup \{\texttt{used}, \texttt{unused}\}$, $1 \leq i \leq |\tau|$, *together with an entry* $\tau[i][v] \in \{\texttt{left}, \texttt{right}\}$ *for every* $1 \leq i \leq \ell - 1$ *and* $v \in X \setminus V(S_s \cap X, \tau[1, i])$.

There are at most $(t+3)^{\ell}2^{t\ell}$ different OCT-signatures. In the definition of feasibility, we replace the last condition of Definition 4.5.29 with the following, stronger one:

- For all $1 \leq i \leq \ell - 1$, the sets $\{v \mid \tau[i][v] = \texttt{left}\}$ and $\{v \mid \tau[i][v] = \texttt{right}\}$ give a bipartition of $G[X \setminus V(S_s \cap X, \tau[1,i])]$.

In the definition of the join operation (Definition 4.5.33), we additionally require two signatures to have equal $\tau[i][v]$ entries (whenever defined) to be considered joinable; the operation copies them to the new signature. In the definition of the forget operation (Definition 4.5.32), we delete any $\tau[i][v]$ entries where v is the vertex being forgotten. In the introduce operation (Definition 4.5.31), we consider (and check the feasibility of) a different copy of a signature for each way of replacing `unused` markers with v markers and each way of assigning $\{\texttt{left}, \texttt{right}\}$ values to new $\tau[i][v]$ entries, where v is the vertex being introduced. As before, to each node we assign an initially empty table of OCT-signatures and fill them bottom-up using these operations.

The proof of correctness proceeds very similarly as for VC-Bound. We only need to argue that the strengthened last condition for feasibility (which uses the additional information about bipartitions in an essential way) is now strong enough to carry through the main inductive proof.

Lemma 5.3.2. *If an OCT-signature τ over X is obtained from a feasible OCT-signature by replacing all vertex markers not in X by* `used` *or* `unused` *markers, then τ is feasible as well.*

Proof. Let τ be obtained from a feasible OCT-signature τ' over X' by replacing all vertex markers in $X' \setminus X$ by `used` or `unused` markers. First note that $V(S_s \cap X, \tau[1,i]) = V(S_s \cap X', \tau'[1,i]) \cap X$. The first three conditions of Definition 4.5.29 follow immediately. Moreover, if $G[X' \setminus S]$ has a bipartition (L, R), then $(L \cap X, R \cap X)$ is a bipartition of $G[X \setminus (S \cap X)]$, hence the fourth condition also follows. □

Lemma 5.3.3. *Let G be a graph S, X_1, X_2 be subsets of $V(G)$ such that every edge of $G[X_1 \cup X_2]$ is contained in $G[X_1]$ or $G[X_2]$. Let L, R be subsets of $X_1 \cap X_2$. If $(L \cap X_1, R \cap X_1)$ is a bipartition of $G[X_1 \setminus S]$ and $(L \cap X_2, R \cap X_2)$ is a bipartition of $G[X_2 \setminus S]$, then (L, R) is a bipartition of $G[(X_1 \cup X_2) \setminus S]$.*

Proof. Let uv be an edge of $G[(X_1 \cup X_2) \setminus S]$. Then it is contained in $G[X_i]$ for some $i \in \{1, 2\}$. It has no endpoint in $S \cap X_i$, hence it is an edge of $G[X_i \setminus S]$. Thus one endpoint is in $L \cap X_i$ and the other in $R \cap X_i$. In particular every edge of $G[(X_1 \cup X_2) \setminus S]$ has one endpoint in L and the other in R. □

Corollary 5.3.4. *Let τ, τ_1, τ_2 be OCT-signatures over X, X_1, X_2 respectively, such that $X = X_1 \cup X_2$ and every edge of $G[X]$ is contained in $G[X_1]$ or $G[X_2]$. Assume furthermore that for all $i \leq \ell$:*

- *$\tau[i] = \tau_1[i]$ whenever $\tau[i] \in X_1$ or $\tau_1[i] \in X_1$,*
- *$\tau[i] = \tau_2[i]$ whenever $\tau[i] \in X_2$ or $\tau_2[i] \in X_2$,*
- *$\tau[i][v] = \tau_1[i][v]$ whenever $v \in X_1$ and $\tau[i][v]$ is defined and*
- *$\tau[i][v] = \tau_2[i][v]$ whenever $v \in X_2$ and $\tau[i][v]$ is defined.*

If τ_1 and τ_2 are feasible, then so is τ.

Proof. The assumptions above imply that τ and τ_1 agree over all changes within X_1, that is, $V(S_s \cap X_1, \tau_1[1,i]) = V(S_s \cap X, \tau[1,i]) \cap X_1$ (and similarly for τ_2). The first three conditions of feasibility for τ follow as for VC-Bound. For the last condition, it suffices to use Lemma 5.3.3 for $S = V(S_s \cap X, \tau[1,i])$, $L = \{v \mid \tau[i][v] = \texttt{left}\}$, and $R = \{v \mid \tau[i][v] = \texttt{right}\}$. □

The following lemma is proved by induction exactly as for VC-Bound, only with Lemma 5.3.2 and Corollary 5.3.4 used when feasibility needs to be argued.

Lemma 5.3.5. *For $i \in V(T)$ and an OCT-signature τ over X_i, $\tau \in A_i$ if and only if τ can be extended to an OCT-signature over V_i that is feasible.*

Theorem 5.3.6. OCT-Bound *and* IBS-Bound *can be solved in $\mathcal{O}^*(2^{t\ell}4^{\ell}(t+3)^{\ell})$ time on graphs of treewidth t.*

Proof. The correctness of our dynamic programming algorithm follows from Lemma 5.3.5. It only remains to consider the running time. The number of possible OCT-signatures is $(t+3)^{\ell}2^{t\ell}$ (instead of the $(t+3)^{\ell}$ for VC-Bound). In the join operation, we required the new $\tau[i][v]$ entries to be equal and thus the running time is again 2^{ℓ} times the number of possible OCT-signatures. In the forget operation the algorithm does only a polynomial number of calculations for each of the OCT-signatures. In the introduce operation, for each of the OCT-signatures we consider in the worst case 2^{ℓ} possible subsets of `unused` markers and 2^{ℓ} possible assignments of `left` or `right` to new $\tau[i][v]$ entries. The total running time is thus $\mathcal{O}^*(4^{\ell}(t+3)^{\ell}2^{t\ell})$.

Combining the same complementing technique we used for VC-Bound and IS-Bound with Proposition 2.4.9, the result for IBS-Bound follows. □

Similarly, using the classical $\mathcal{O}^*(2^{\mathcal{O}(t \log t)})$ algorithm for FVS and IF (which maintains what partition of X_i the connected components of V_i can produce), we get the following running times for bounded reachability variants of these problems.

Theorem 5.3.7. FVS-Bound *and* IF-Bound *can be solved in $\mathcal{O}^*(t^{\ell t} 4^{\ell} (t+3)^{\ell})$ time on graphs of treewidth t.*

5.3.2 A meta-theorem

We now show that a host of bounded reachability problems definable in monadic second-order logic (MSOL) become fixed-parameter tractable when parameterized by $\ell + t$. First, we review the syntax and semantics of MSOL over graphs. The reader is referred to the excellent survey by Grohe [68] for more details.

We have an infinite set of *individual variables*, denoted by lowercase letters x, y, and z, and an infinite set of *set variables*, denoted by uppercase letters X, Y, and Z. A *monadic second-order formula (MSOL-formula)* ϕ over a graph G is constructed from *atomic formulas* $\mathcal{E}(x, y)$, $x \in X$, and $x = y$ using the usual Boolean connectives as well as existential and universal quantification over individual and set variables. We write $\phi(x_1, \dots, x_r, X_1, \dots, X_s)$ to indicate that ϕ is a formula with free variables $x_1, \dots, x_r$ and $X_1, \dots, X_s$, where free variables are variables not bound by quantifiers.

For a formula $\phi(x_1,\dots,x_r,X_1,\dots,X_s)$, a graph G, vertices $v_1, \dots, v_r$, and sets $V_1, \dots, V_r$, we write $G \models \phi(v_1, \dots, v_r, V_1, \dots, V_r)$ if ϕ is satisfied in G when $\mathcal{E}$ is interpreted by the adjacency relation $E(G)$, the variables x_i are interpreted by v_i, and the variables X_i are interpreted by V_i. We say that a vertex-subset problem $\mathcal{Q}$ is *definable in monadic second-order logic* if there exists an MSOL-formula $\phi(X)$ with one free set variable such that $S \subseteq V(G)$ is a feasible solution of problem $\mathcal{Q}$ for instance G if and only if $G \models \phi(S)$. For example, an independent set is definable by the formula $\phi_{\text{IS}}(X) = \forall_x \forall_y (x \in X \land y \in X) \to \neg\mathcal{E}(x, y)$.

Theorem 5.3.8 (Courcelle [30])**.** *There is an algorithm that given a MSOL-formula $\phi(x_1, \dots, x_r, X_1, \dots, X_s)$, a graph G, vertices $v_1,\dots,v_r \in V(G)$, and sets $V_1,\dots,V_s \subseteq V(G)$ decides whether $G \models \phi(v_1, \dots, v_r, V_1, \dots, V_s)$ in $\mathcal{O}(f(tw(G), |\phi|) \cdot n)$ time, for some computable function f.*

Theorem 5.3.9. *If a vertex-subset problem $\mathcal{Q}$ is definable in monadic second-order logic by a formula $\phi(X)$, then Q-Bound parameterized by $\ell + tw(G) + |\phi|$ is fixed-parameter tractable.*

Proof. We provide a proof for $\mathcal{Q}$ a minimization problem as the proof for maximization problems is analogous. Given an instance (G, S_s, S_t, k, ℓ) of Q-Bound, we build an MSOL-formula $\omega(X_0, X_\ell)$ such that $G \models \omega(S_s, S_t)$ if and only if the corresponding instance is a yes-instance. Since the size of ω will be bounded by a function of $\ell + |\phi|$, the statement will follow from Theorem 5.3.8.

As MSOL does not allow cardinality constraints, we overcome this limitation using sign-sequences. That is, we let $L \subseteq \{-1, +1\}^\ell$ be the set of all sequences of length ℓ over $\{-1, +1\}$ which do not violate the maximum allowed capacity. In other words, given S_s and k, a sequence σ is in L if and only if for all $\ell' \leq \ell$ it satisfies $|S_s| + \sum_{i=1}^{\ell'} \sigma[i] \leq k$, where $\sigma[i]$ is the i^{th} element in sequence σ. We let $\omega = \bigvee_{\sigma \in L} \omega_\sigma$ and

$$\omega_\sigma(X_0, X_\ell) = \exists_{X_1,\dots,X_{\ell-1}} \bigwedge_{0 \leq i \leq \ell} \phi(X_i) \wedge \bigwedge_{1 \leq i \leq \ell} \psi_{\sigma[i]}(X_{i-1}, X_i)$$

where $\psi_{-1}(X_{i-1}, X_i)$ means X_i is obtained from X_{i-1} by removing one element and $\psi_{+1}(X_{i-1}, X_i)$ means it is obtained by adding one element. Formally, we have:

$$\psi_{-1}(X_{i-1}, X_i) = \exists_x\ x \in X_{i-1} \wedge x \notin X_i \wedge \forall y\, (y \in X_i \leftrightarrow (y \in X_{i-1} \wedge y \neq x))$$

$$\psi_{+1}(X_{i-1}, X_i) = \exists_x\ x \notin X_{i-1} \wedge x \in X_i \wedge \forall y\, (y \in X_i \leftrightarrow (y \in X_{i-1} \vee y = x))$$

It is easy to see that $G \models \omega_\sigma(S_s, S_t)$ if and only if there is a reconfiguration sequence from S_s to S_t (corresponding to $X_0, X_1, \dots, X_\ell$) such that the i^{th} step removes a vertex if $\sigma[i] = -1$ and adds a vertex if $\sigma[i] = +1$. Since $|L| \leq 2^\ell$, the size of the MSOL-formula ω is bounded by an exponential function of $\ell + |\phi|$. $\square$

5.4 Reconfiguration and irrelevant vertices

The IS-Reach problem parameterized by k is `W[1]`-hard on general graphs (Theorem 4.2.2). Ito et al. [83, 84] showed that the problem becomes fixed-parameter tractable for graphs of bounded degree, planar graphs, and graphs which exclude $K_{3,d}$ as a (not necessarily induced) subgraph, for any constant d. Using the notion of irrelevant vertices, we simplify these earlier results and push the boundary further by showing that the problem remains fixed-parameter tractable for graphs of bounded degeneracy and nowhere-dense graphs (Figure 2.2). As a corollary, we answer positively an open question concerning the parameterized complexity of the problem on graphs of bounded treewidth.

Clearly, the main open question is whether IS-Reach remains fixed-parameter tractable on graphs excluding $K_{d,d}$ as a subgraph. Intuitively, and in contrast to Theorem 3.3.11, one would not expect a sparse graph to have "too many" dominating sets of fixed small size k as n becomes larger and larger. For independent sets, the situation is reversed. As n grows larger, so does the number of independent sets of fixed size k. So it remains to be seen whether some structural properties of graphs excluding $K_{d,d}$ as a subgraph can be used to settle our open question or whether the problem is `W[1]`-hard. Another open question is whether we can adapt our results to IS-Bound parameterized by k, i.e. can we find shortest reconfiguration sequences or strongly irrelevant vertices in the same running time.

In what follows, we will consider a slightly different formulation of the IS-Reach problem, which we call TJ-IS-Reach.

TJ-IS-Reach	
Input:	A graph G, positive integer k, and two independent sets I_s and I_t of G of size exactly k
Question:	Is there a path from I_s to I_t in $R_{IS}(G, k, k + 1)$?

The TJ-IS-Reach can be seen as the reachability variant of the Independent Set problem under the token jumping model since the input independent sets must be of the same size and any vertex addition must be followed by a vertex removal; the minimum and maximum allowed capacities are k and $k + 1$, respectively. We note that for an instance (G, I_s, I_t, k) of IS-Reach, we can construct an instance (G, I'_s, I'_t, k) of TJ-IS-Reach where $I'_s \subseteq I_s$ and $I'_t \subseteq I_t$ are both of size exactly k (picked arbitrarily). Proposition 5.4.1, which is due to Kaminski et al. [90], immediately implies that our results for TJ-IS-Reach can be extended to IS-Reach.

Proposition 5.4.1 ([90]). *Given a graph G, integer k, and two independent sets I_s and I_t of G of size exactly k, there exists a reconfiguration sequence from I_s to I_t in $R_{IS}(G, k, n)$ if and only if there exists a reconfiguration from I_s to I_t in $R_{IS}(G, k, k + 1)$.*

5.4.1 Graphs of bounded degeneracy

To show that the TJ-IS-Reach problem is fixed-parameter tractable on d-degenerate graphs, for some constant d, we will proceed in two stages. In the first stage, we will show, for an instance (G, I_s, I_t, k), that as long as the number of low-degree vertices in G is "large enough" we can find an irrelevant vertex (Definition 3.3.9). Once the number of low-degree vertices is bounded, a simple counting argument (Proposition 5.4.2) shows that the size of the remaining graph is also bounded and hence we can solve the instance by exhaustive enumeration.

Proposition 5.4.2. *Let G be an n-vertex d-degenerate graph, $S_1 \subseteq V(G)$ be the set of vertices of degree at most $2d$, and $S_2 = V(G) \setminus S_1$. If $|S_1| < s$, then $|V(G)| \leq (2d+1)s$.*

Proof. The number of edges in a d-degenerate graph is at most dn and hence its average degree is at most $2d$ (Proposition 2.1.6). If $|V(G)| = (2d+1)s + c$, for $c \geq 1$, then $|S_2| = |V(G) \setminus S_1| > 2ds + c$, $\sum_{v \in S_2} |N_G(v)| > (2ds+c)(2d+1)$, and we obtain the following contradiction:

$$\begin{aligned} \frac{\sum_{v \in S_1} |N_G(v)| + \sum_{v \in S_2} |N_G(v)|}{|V(G)|} &> \frac{(2ds+c)(2d+1)}{(2d+1)s+c} \\ &= \frac{4d^2s + 2ds + 2dc + c}{(2d+1)s+c} \\ &= \frac{2d(2ds+s+c)+c}{2ds+s+c} > 2d. \end{aligned}$$

□

To find irrelevant vertices, we make use of the following classical result of Erdõs and Rado [51], also known in the literature as the Sunflower Lemma. We first define the terminology used in the statement of the theorem. A *sunflower* with k *petals* and a *core* Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are petals and we require none of them to be empty. Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

Theorem 5.4.3 (Sunflower Lemma [51])**.** *Let $\mathcal{A}$ be a family of sets (without duplicates) over a universe $\mathcal{U}$, such that each set in $\mathcal{A}$ has cardinality at most d. If $|\mathcal{A}| > d!(k-1)^d$, then $\mathcal{A}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$, $|\mathcal{U}|$, and k.*

Lemma 5.4.4. *Let (G, I_s, I_t, k) be an instance of* TJ-IS-Reach *where G is d-degenerate and let B be the set of vertices in $V(G) \setminus \{I_s \cup I_t\}$ of degree at most $2d$. If $|B| > (2d+1)!(2k-1)^{2d+1}$, then there exists an irrelevant vertex $v \in V(G) \setminus \{I_s \cup I_t\}$ such that (G, I_s, I_t, k) is a yes-instance if and only if (G', I_s, I_t, k) is a yes-instance, where G' is obtained from G by deleting v and all edges incident on v.*

Proof. Let b_1, b_2, $\ldots$, $b_{|B|}$ denote the vertices in B and let $\mathcal{A} = \{N_G[b_1], N_G[b_2], \ldots, N_G[b_{|B|}]\}$ denote the family of sets corresponding to the closed neighborhoods of each vertex in B and set $\mathcal{U} = \bigcup_{b \in B} N[b]$. Since $|B|$ is greater than $(2d+1)!(2k-1)^{2d+1}$, we know from Theorem 5.4.3 that $\mathcal{A}$ contains a sunflower with $2k$ petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$ and k. Note that we assume, without loss of generality, that there are no two vertices u and v in $V(G)\setminus\{I_s \cup I_t\}$ such that $N_G[u] = N_G[v]$, as we can safely delete one of them from the input graph otherwise, i.e. one of the two is (strongly) irrelevant. Let v_{ir} be a vertex whose closed neighborhood corresponds to one of those $2k$ petals. We claim that v_{ir} is irrelevant and can therefore be deleted from G to obtain G'.

To see why, consider any reconfiguration sequence $\sigma = \langle I_s = I_0, I_1, \ldots, I_t = I_\ell \rangle$ from I_s to I_t in $R_{IS}(G, k, k+1)$. Since $v_{ir} \notin I_s \cup I_t$, we let p, $0 < p < \ell$, be the first index in σ at which v_{ir} is added, i.e. $v_{ir} \in I_p$ and $v_{ir} \notin I_i$ for all $i < p$. Moreover, we let $q + 1$, $p < q + 1 \leq \ell$ be the first index after p at which v_{ir} is removed, i.e. $v_{ir} \in I_q$ and $v_{ir} \notin I_{q+1}$. We will consider the subsequence $\sigma_s = \langle I_p, \ldots, I_q \rangle$ and show how to modify it so that it does not touch v_{ir}. Applying the same procedure to every such subsequence in σ suffices to prove the lemma.

Since the sunflower constructed to obtain v_{ir} has $2k$ petals and the size of any independent set in σ (or any reconfiguration sequence in general) is at most $k + 1$, there must exist another *free* vertex v_{fr} whose closed neighborhood corresponds to one of the remaining $2k - 1$ petals which we can add at index p instead of v_{ir}, i.e. $v_{fr} \notin N_G[I_p]$. We say v_{fr} *represents* v_{ir}. Assume that no such vertex exists. Then we know that either some vertex in the core of the sunflower is in I_p, contradicting the fact that we are adding v_{ir}, or every petal of the sunflower contains a vertex in I_p, which is not possible since the size of any independent set is at most $k + 1$ and the number of petals is larger. Hence, we first modify the subsequence σ_s by adding v_{fr} instead of v_{ir}. Formally, we have $\sigma'_s = \langle (I_p \setminus \{v_{ir}\}) \cup \{v_{fr}\}, \ldots, (I_q \setminus \{v_{ir}\}) \cup \{v_{fr}\} \rangle$.

To be able to replace σ_s by σ'_s in σ and obtain a reconfiguration sequence from I_s to I_t, then all of the following conditions must hold:

(1) $|(I_q \setminus \{v_{ir}\}) \cup \{v_{fr}\}| = k + 1$.

(2) $(I_i \setminus \{v_{ir}\}) \cup \{v_{fr}\}$ is an independent set of G for all $p \leq i \leq q$,

(3) $|(I_i \setminus \{v_{ir}\}) \cup \{v_{fr}\} \Delta (I_{i+1} \setminus \{v_{ir}\}) \cup \{v_{fr}\}| = 1$ for all $p \leq i < q$, and

(4) $k \leq |(I_i \setminus \{v_{ir}\}) \cup \{v_{fr}\}| \leq k + 1$ for all $p \leq i \leq q$.

It is not hard to see that if there exists no i, $p < i \leq q$, such that σ'_s adds a vertex in $N[v_{fr}]$ at position i, then all four conditions hold. If there exists such a position, we will modify σ'_s into yet another subsequence σ''_s by finding a new vertex to represent v_{ir}. The length of σ''_s will be one greater than the length of σ'_s.

We let i, $p < i \leq q$, be the first position in σ'_s at which a vertex in $u \in N[v_{fr}]$ (possibly equal to v_{fr}) is added. Using the same arguments for finding v_{fr}, and since we constructed a sunflower with $2k$ petals, we can find another vertex v'_{fr} such that $N[v_{fr}] \cap I_{i-1} = \emptyset$. This new vertex will represent v_{ir} instead of v_{fr}. We construct σ''_s from σ'_s as follows: $\sigma''_s = \langle I_p \setminus \{v_{ir}\} \cup \{v_{fr}\}, \ldots, I_{i-1} \setminus \{v_{ir}\} \cup \{v_{fr}\}, I_{i-1} \setminus \{v_{ir}\} \cup \{v'_{fr}\}, I_i \setminus \{v_{ir}\} \cup \{v'_{fr}\}, \ldots, I_q \setminus \{v_{ir}\} \cup \{v'_{fr}\}\rangle$. If σ''_s now satisfies all four conditions then we are done. Otherwise, we repeat the same process (which can occur at most $q - p$ times) until we obtain such a subsequence. □

Theorem 5.4.5. TJ-IS-Reach *on d-degenerate graphs is fixed-parameter tractable parameterized by $k + d$.*

Proof. For an instance (G, I_s, I_t, k) of TJ-IS-Reach, we know from Lemma 5.4.4 that as long as $V(G) \setminus \{I_s \cup I_t\}$ contains more than $(2d + 1)!(2k - 1)^{2d+1}$ vertices of degree at most $2d$ we can find an irrelevant vertex and reduce the size of the graph. After exhaustively reducing the graph to obtain G', we known that $G'[V(G') \setminus \{I_s \cup I_t\}]$, which is also d-degenerate, has at most $(2d + 1)!(2k - 1)^{2d+1}$ vertices of degree at most $2d$. Hence, applying Proposition 5.4.2, we know that $|V(G') \setminus \{I_s \cup I_t\}| \leq (2d+1)(2d+1)!(2k-1)^{2d+1}$ and $|V(G')| \leq (2d + 1)(2d + 1)!(2k - 1)^{2d+1} + 2k$. □

5.4.2 Nowhere dense graphs

Nesetril and Ossona de Mendez [113] showed an interesting relationship between nowhere-dense classes and a property of classes of graphs introduced by Dawar [34, 35] called *quasi-wideness*. We will use quasi-wideness and show a rather interesting relationship between TJ-IS-Reach on graphs of bounded degeneracy and nowhere-dense graphs. That is, our algorithm for nowhere-dense graphs will closely mimic the previous algorithm in the following sense. Instead of using the sunflower lemma to find a large sunflower, we will use quasi-wideness to find a "large enough almost sunflower" with an initially "unknown" core and then use structural properties of the graph to find this core and complete the sunflower. We first state some of the results that we need. Given a graph G, a set $S \subseteq V(G)$ is called *r-scattered* if $N^r_G(u) \cap N^r_G(v) = \emptyset$ for all distinct $u, v \in S$.

Proposition 5.4.6. *Let G be a graph and let $S = \{s_1, s_2, ..., s_k\} \subseteq V(G)$ be a 2-scattered set of size k in G. Then the closed neighborhoods of the vertices in S form a sunflower with k petals and an empty core.*

Definition 5.4.7. *A class $\mathcal{C}$ of graphs is* uniformly quasi-wide *with* margin *$s_{\mathcal{C}} : \mathbb{N} \to \mathbb{N}$ and $N_{\mathcal{C}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ if for all $r, k \in \mathbb{N}$, if $G \in \mathcal{C}$ and $W \subseteq V(G)$ with $|W| > N_{\mathcal{C}}(r, k)$,* *then there is a set $S \subseteq W$ with $|S| < s_{\mathcal{C}}(r)$, such that W contains an r-scattered set of size at least k in $G[V(G) \setminus S]$. $\mathcal{C}$ is* effectively uniformly quasi-wide *if $s_{\mathcal{C}}(r)$ and $N_{\mathcal{C}}(r,k)$ are computable.*

Theorem 5.4.8 ([36]). *A class $\mathcal{C}$ of graphs is effectively nowhere dense if and only if $\mathcal{C}$ is effectively uniformly quasi-wide.*

Theorem 5.4.9 ([36]). *Let $\mathcal{C}$ be an effectively nowhere-dense class of graphs and h be the computable function such that $K_{h(r)} \not\preceq_m^r G$ for all $G \in \mathcal{C}$. Let G be an n-vertex graph in $\mathcal{C}$, $r,k \in \mathbb{N}$, and $W \subseteq V(G)$ with $|W| \geq N(h(r),r,k)$, for some computable function N. Then in $\mathcal{O}(n^2)$ time, we can compute a set $B \subseteq V(G)$, $|B| \leq h(r)-2$, and a set $A \subseteq W$ such that $|A| \geq k$ and A is an r-scattered set in $G[V(G) \setminus B]$.*

Lemma 5.4.10. *Let $\mathcal{C}$ be an effectively nowhere-dense class of graphs and h be the computable function such that $K_{h(r)} \not\preceq_m^r G$ for all $G \in \mathcal{C}$. Let (G, I_s, I_t, k) be an instance of* TJ-IS-Reach *where $G \in \mathcal{C}$ and let R be the set of vertices in $V(G) \setminus \{I_s \cup I_t\}$. Moreover, let $\mathcal{P} = \{P_1, P_2, \ldots\}$ be a family of sets which partitions R such that for any two distinct vertices $u,v \in R$, $u,v \in P_i$ if and only if $N_G(u) \cap \{I_s \cup I_t\} = N_G(v) \cap \{I_s \cup I_t\}$. If there exists a set $P_i \in \mathcal{P}$ such that $|P_i| > N(h(2), 2, 2^{h(2)+1}k)$, for some computable function N, then there exists an irrelevant vertex $v \in V(G) \setminus \{I_s \cup I_t\}$ such that (G, I_s, I_t, k) is a yes-instance if and only if (G', I_s, I_t, k) is a yes-instance, where G' is obtained from G by deleting v and all edges incident on v.*

Proof. By construction, we known that the family $\mathcal{P}$ contains at most 4^k sets, as we partition R based on their neighborhoods in $I_s \cup I_t$. Note that some vertices in R have no neighbors in $I_s \cup I_t$ and will therefore belong to the same set in $\mathcal{P}$.

Assume that there exists a $P \in \mathcal{P}$ such that $|P| > N(h(2), 2, 2^{h(2)+1}k)$. Consider the graph $G[R]$. By Theorem 5.4.9, we can, in $\mathcal{O}(|R|^2)$ time, compute a set $B \subseteq R$, $|B| \leq h(2)-2$, and a set $A \subseteq P$ such that $|A| \geq 2^{h(2)+1}k$ and A is a 2-scattered set in $G[R \setminus B]$. Now let $\mathcal{P}' = \{P'_1, P'_2, \ldots\}$ be a family of sets which partitions A such that for any two distinct vertices $u,v \in A$, $u,v \in P'_i$ if and only if $N_G(u) \cap B = N_G(v) \cap B$. Since $|A| \geq 2^{h(2)+1}k$ and $|\mathcal{P}'| \leq 2^{h(2)}$, we know that at least one set in $\mathcal{P}'$ will contain at least $2k$ vertices of A. Denote these $2k$ vertices by A'. All vertices in A' have the same neighborhood in B and the same neighborhood in $I_s \cup I_t$ (as all vertices in A' belonged to the same set $P \in \mathcal{P}$). Moreover, A' is a 2-scattered set in $G[R \setminus B]$. Hence, the sets $\{N_G[a'_1], N_G[a'_2], \ldots, N_G[a'_{2k}]\}$, i.e. the closed neighborhoods of the vertices in A', form a sunflower with $2k$ petals (Proposition 5.4.6); the core of this sunflower is contained in $B \cup I_s \cup I_t$. Using the same arguments as we did in the proof of Lemma 5.4.4, we can show that there exists at least one irrelevant vertex $v \in V(G) \setminus \{B \cup I_s \cup I_t\}$. □

Theorem 5.4.11. TJ-IS-Reach *restricted to any effectively nowhere-dense class $\mathcal{C}$ of graphs is fixed-parameter tractable parameterized by k.*

Proof. If after partitioning $V(G) \setminus \{I_s \cup I_t\}$ into at most 4^k sets the size of every set $P \in \mathcal{P}$ is bounded by $N(h(2), 2, 2^{h(2)+1}k)$, then we can solve the problem by exhaustive enumeration, as $|V(G)| \leq 2k + 4^k N(h(2), 2, 2^{h(2)+1}k)$. Otherwise, we can apply Lemma 5.4.10 and reduce the size of the graph in polynomial time. □

Chapter 6

Reconfiguration of constraint satisfaction problems

In this chapter, we study reconfiguration questions related to graph coloring, and more generally constraint satisfaction problems. In Section 6.2, we explore how the complexity of the bounded reachability variant of the COLORING problem depends on the parameters k (the number of colors) and ℓ (the length of reconfiguration sequences). Recall that COL-REACH and COL-BOUND are known to be in P for $k \leq 3$ [25, 27, 87] and PSPACE-complete for $k \geq 4$ [18]. Therefore, both problems parameterized by $k \geq 4$ are para-PSPACE-complete, i.e. PSPACE-complete for every value of the parameter, and are not likely to be solvable in $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ time (not in FPT), for any computable function f. We show that COL-BOUND is W[1]-hard (but in XP) when parameterized only by ℓ. On the positive side, we give an $\mathcal{O}(f(k, \ell)n^{\mathcal{O}(1)})$ time algorithm for the problem, for a computable function f. Hence, we show that the problem is fixed-parameter tractable when parameterized by $k + \ell$.

In Section 6.3, we show that the more general problem CSP-BOUND is fixed-parameter tractable parameterized by $k + \ell + r$, where r is the maximum constraint arity (maximum number of variables participating in a single constraint) and k is the maximum domain size (maximum number of permissible values for each variable). Moreover, we show that in some sense this result is the best possible, i.e. dropping any of the three values in the parameter makes the problem hard. In particular, we show that for parameter ℓ the problem is W[2]-hard, even for $k = 2$. For p the number of variables with different values in the two given assignments, we show that the problem is W[2]-hard when parameterized by $\ell - p$, even for $k = 2$ and $r = 3$.

Finally, we consider weighted constraint satisfaction problems with Boolean domains in Section 6.4 in an attempt to combine results from earlier chapters with our results on CSP-Bound. We provide a series of classifications under Schaefer's [125] framework using known results from the literature along with some of the results proved throughout this work.

6.1 Notation and definitions

In contrast to previous chapters, feasible solutions to a Coloring or CSP instance assign a color/value to every vertex/variable of the input and we therefore require additional terminology. We define notions related to graph coloring here and defer the formal definitions related to constraint satisfaction problems to Section 6.3.

A *k-color assignment* for a graph G is a function $\alpha : V(G) \to \{1, \ldots, k\}$ of *colors* to the vertices of G. It is a *proper k-coloring*, or a *k-coloring* for short, if there are no edges $uv \in E(G)$ with $\alpha(u) = \alpha(v)$. On the other hand, if there exists such an edge uv, then this edge is said to give a *color conflict*. A graph that admits a k-coloring is called *k-colorable*. The minimum k such that G is k-colorable is called the *chromatic number* of G, denoted by $\chi(G)$. For a k-coloring α of G, the set of colors *used by α* is $\{\alpha(v) \mid v \in V(G)\}$.

For any graph G and integer k, the vertex set of $R_{COL}(G, 0, k)$ corresponds to all k-colorings of G and two colorings are adjacent if and only if they differ on exactly one vertex. The integer k is called the *number of admissible colors*. Walks in $R_{COL}(G, 0, k)$ from α to β are also called *k-recoloring sequences* from α to β. If there exists an integer k such that $\alpha_0, \ldots, \alpha_m$ is a k-recoloring sequence, then this is called a *recoloring sequence* from α_0 to α_m. Clearly, if there exists a k-recoloring sequence from α to β of length ℓ, then $R_{COL}(G, 0, k)$ contains a path from α to β of length at most ℓ.

A *k-color list assignment* for a graph G is a mapping L that assigns a *color list* $L(v) \subseteq \{1, \ldots, k\}$ to each vertex $v \in V(G)$. A k-coloring α of G is an *L-coloring* if $\alpha(v) \in L(v)$ for all v. By $R_{COL}(G, L)$ we denote the subgraph of $R_{COL}(G, 0, k)$ induced by all L-colorings of G and walks in $R_{COL}(G, L)$ are called *L-recoloring sequences*. We are now ready to define the following problems:

Col-Bound
Input: A graph G, positive integers k and ℓ, and two k-colorings α and β of G.
Question: Is there a path of length at most ℓ from α to β in $R_{COL}(G, 0, k)$?

L-Col-Bound

Input: A graph G, positive integers k and ℓ, a k-color list assignment L of G, and two L-colorings α and β of G.

Question: Is there a path of length at most ℓ from α to β in $R_{COL}(G, L)$?

For a positive integer $k \geq 1$, we let $[k] = \{1, \ldots, k\}$. For a function $f : D \to I$ and subset $D' \subseteq D$, we denote by $f|_{D'}$ the restriction of f to the domain D'. The (unique) trivial function with empty domain is denoted by $f^{\emptyset}$. Note that for any function g, $g|_{\emptyset} = f^{\emptyset}$. We use $\text{poly}(x_1, \ldots, x_p)$ to denote a polynomial function on variables $x_1, \ldots, x_p$.

6.2 Graph recoloring

In this section we explore fully how the complexity of the Col-Bound problem depends on the problem parameters k and ℓ. Firstly, Col-Bound is easily observed to be PSPACE-hard in general, for $k \geq 4$: Since there are at most k^n different k-colorings of a graph on n vertices, a path from α to β exists if and only if there exists one of length at most k^n. So setting $\ell = k^n$ yields a trivial reduction from the PSPACE-hard Col-Reach to Col-Bound. Nevertheless, this only establishes *weak* PSPACE-hardness, since the chosen value of ℓ is exponential in the instance size. In other words, if we require that all integers be encoded in unary, then this is not a polynomial reduction. And indeed, the complexity status of the problem changes under that requirement: in that case, Col-Bound is easily observed to be in NP. If $(G, k, \ell, \alpha, \beta)$ is a yes-instance, then a path of length ℓ in $R_{COL}(G, 0, k)$ from α to β is a polynomial certificate. Bonsma et al. [19] showed that Col-Bound is in fact NP-complete when ℓ is encoded in unary, or in other words it is *strongly NP-hard.*

We show that the problem can be solved in $\mathcal{O}(2^{k(\ell+1)} \cdot \ell^{\ell} \cdot \text{poly}(n))$ time. This establishes that Col-Bound is fixed parameter tractable when parameterized by $k + \ell$. One may ask whether the problem is still in FPT when parameterized only by ℓ. We show that this is not the case (unless W[1]=FPT). We observe however that a straightforward branching algorithm can solve the problem in time $n^{\mathcal{O}(\ell)}$, hence in polynomial time for any constant ℓ. In other words, Col-Bound is in XP, parameterized by ℓ.

The complexity status of Col-Bound is summarized in Table 6.1. Our main results are marked by (*). Unmarked results follow immediately from results in an adjacent row or column. We note that the FPT result was also obtained very recently and independently by Johnson et al. [87], although they use an algorithm very different from ours.

Table 6.1: Complexity landscape of the COL-BOUND problem.

	ℓ as binary variable	ℓ as unary variable	ℓ as parameter
k as input variable (unary or binary)	PSPACE-complete	NP-complete	W[1]-hard (*) XP
k as parameter	(para-)PSPACE-complete	(para-)NP-complete	FPT (*)
$k \geq 4$ as constant	PSPACE-complete [18]	NP-complete [19]	FPT
$k \leq 3$	P [87]	P	P

6.2.1 W[1]-hardness

We give a reduction from the INDEPENDENT SET problem, known to be W[1]-hard [46, 62] when parameterized by solution size. We will also use the following result, which was shown independently by Cereceda [24], Marcotte et al. [102] and Jacob [86]: There exists a graph G such that for every pair of k-colorings α and β of G, there exists a path from α to β in $R_{COL}(G, 0, 2k-1)$, and there are examples where at least $2k-1$ colors are necessary. The graphs constructed to prove the latter result [24, 86, 102] are in fact very similar. We will use these graphs for our reduction, though we will need a slightly stronger claim, so we need to restate the proof.

For any integer $k \geq 1$, we let $B_k = \overline{K_k \times K_k}$ (the complement of the Cartesian product graph of two complete graphs on k vertices). More precisely, we let $V(B_k) = \{b^i_j \mid i, j \in \{1, \ldots, k\}\}$, and two vertices b^i_j and $b^{i'}_{j'}$ are adjacent if and only if $i \neq i'$ and $j \neq j'$. We define two k-colorings α^k and β^k for B_k by setting $\alpha^k(b^i_j) = i$ and $\beta^k(b^i_j) = j$ for all vertices b^i_j.

Theorem 6.2.1. *For every integer $k \geq 1$, there exists B_k, α^k, and β^k as defined above, such that every recoloring sequence from α^k to β^k contains a $(2k-1)$-coloring. Moreover, there exists a $(2k-1)$-recoloring sequence of length at most $2k^2$ from α^k to β^k.*

Proof. We consider a recoloring sequence $\gamma_0, \ldots, \gamma_\ell$ from α^k to β^k. For $i \in \{1, \ldots, k\}$, the vertex set $R_i = \{b^i_j \mid j \in \{1, \ldots, k\}\}$ is called a *row* of B_k. For every i, the coloring α^k colors the vertices of row R_i all with the same color, and β^k colors them all differently. So we may choose the lowest index p such that γ_p colors all vertices of at least one row R_i differently. The choice of p guarantees that, for γ_p, there exists a color that is used for at least two different vertices in every row other than R_i, and thus is not used in any other row (this follows easily from the definition of B_k). We conclude that γ_p uses at least $2k-1$ different colors in total, which proves the first statement.

To obtain a $(2k-1)$-recoloring sequence from α^k to β^k, we can apply the following general method (which applies in fact to any two k-colorings of any k-colorable graph): We choose an arbitrary k-coloring γ of B_k. For every vertex $v \in V(B_k)$ with $\gamma(v) \leq k-1$, we recolor v to the color $k+\gamma(v)$ (this is a color in $\{k+1, \ldots, 2k-1\}$, so it is not used by α^k or β^k). Next, we recolor all vertices v to their target color $\beta(v)$, starting with those vertices v with $\gamma(v) = k$. It can be verified that this way, a k-coloring is maintained throughout, and that every vertex is recolored at most twice. □

We remark that the distance from α^k to β^k in $R_{COL}(B_k, 0, 2k-1)$ is strictly smaller than $2k^2$, but we do not need to know or prove the exact distance. We can now state the reduction from Independent Set to Col-Bound that we use to prove `W[1]`-hardness. Given an instance $(G, t-1)$ of Independent Set, where $V = \{v_1, \ldots, v_n\}$, we construct a graph G' in time polynomial in $n+m+t$ as follows. G' contains a copy of G and a copy of B_t with all edges between them. In addition, G' contains $n+t+1$ independent sets C_1, $\ldots$, C_{n+t+1}, each of size $2t+2t^2$. We say that C_i (for $1 \leq i \leq n+t+1$) is a *color-guard set*, as it will be used to enforce some coloring constraints: In the colorings we define, and all colorings reachable from them using at most $|C_i|-1$ recolorings, C_i will contain at least one vertex of color i. We let $V_G = \{g_1, \ldots, g_n\}$, $V_B = \{b^i_j \mid i, j \in \{1, \ldots, t\}\}$, $V_C = C_1 \cup \ldots \cup C_{n+t+1}$, and hence $V(G') = V_G \cup V_B \cup V_C$. The total number of vertices in G' is therefore $n + t^2 + (n+t+1)(2t+2t^2)$. For every vertex $g_i \in V_G$, we add all edges between g_i and the vertices in $V_C \setminus (C_i \cup C_{n+t+1})$. Similarly, for every vertex $b \in V_B$, we add all edges between b and the vertices in C_{n+t+1}. We define α as follows. For every vertex $g_i \in V_G$, $1 \leq i \leq n$, we set $\alpha(g_i) = i$. For every $i \in \{1, \ldots, n+t+1\}$ and every vertex $c \in C_i$, we set $\alpha(c) = i$. For every vertex $b^i_j \in V_B$ (with $i, j \in \{1, \ldots, t\}$), we choose $\alpha(b^i_j) = n+i$.

Proposition 6.2.2. *For G' and α as constructed above, from a graph G on n vertices, if $k = n+t+1$ and $\alpha_0, \ldots, \alpha_\ell$ is a k-recoloring sequence of length at most $2t+2t^2$ starting from α, then for all $g_i \in V_G$ and $0 \leq x \leq \ell$, we have $\alpha_x(g_i) \in \{i, n+t+1\}$. Moreover, for all $b \in V_B$ and $0 \leq x \leq \ell$, we have $\alpha_x(b) \neq n+t+1$.*

Proof. Since every vertex $g_i \in V_G$ is adjacent to all vertices in $V_C \setminus \{C_i \cup C_{n+t+1}\}$, and $|C_j| = 2t+2t^2$ for all j, $1 \leq j \leq n+t+1$, it follows that if g_i receives a color $j \notin \{i, n+t+1\}$, then all vertices in C_j must have been recolored earlier, which contradicts the fact that we consider a recoloring sequence of length at most $2t+2t^2$. Similarly, every vertex $b \in V_B$ is adjacent to all vertices in C_{n+t+1}. Therefore, if b receives color $n+t+1$ then all vertices in C_{n+t+1} must have been recolored first. □

Finally, we define the target coloring β: for every vertex $v \in V_G \cup V_C$ we set $\beta(v) = \alpha(v)$. For every vertex $b^i_j \in V_B$ (with $i, j \in \{1, \ldots, t\}$), we choose $\beta(b^i_j) = n + j$. In other words, the goal is to change from a 'row coloring' to a 'column coloring' for V_B, while maintaining the same coloring for vertices in $V_G \cup V_C$. The corresponding Col-Bound instance is denoted by $(G', k, \ell, \alpha, \beta)$ with $k = n + t + 1$ and $\ell = 2t + 2t^2$.

Lemma 6.2.3. *For G', k, ℓ, α, and β as constructed above, from a graph G on n vertices, if G has an independent set of size at least $t - 1$, then $R_{COL}(G', 0, k)$ contains a path from α to β of length at most $\ell = 2t + 2t^2$.*

Proof. We let S be an independent set of size $t-1$ in G. The following recoloring sequence is an $(n + t + 1)$-recoloring sequence from α to β of length at most $2t + 2t^2$:

- We assign the vertices in G' corresponding to the vertices in S with color $n + t + 1$ (one by one). Since by construction vertices in V_G are not connected to vertices in C_{n+t+1}, none of these steps will introduce any color conflicts.
- From Theorem 6.2.1, we can recolor the vertices in V_B using $2t-1$ colors and at most $2t^2$ recoloring steps. Since $t - 1$ vertices have been recolored in the previous step, we can now use the corresponding $t - 1$ colors to apply the $(2t - 1)$-recoloring sequence to V_B.
- We recolor the $t - 1$ vertices that were initially recolored by assigning them their original color again (which is also their target color in β). None of these steps will introduce any color conflicts since at this point every vertex in V_B is again assigned a color greater than n.

Clearly, the described recoloring sequence consists of at most $2(t - 1) + 2t^2 < 2t + 2t^2$ recoloring steps. □

Lemma 6.2.4. *For G', k, ℓ, α, and β as constructed above, from a graph G on n vertices, if $R_{COL}(G', 0, k)$ contains a path from α to β of length at most $\ell = 2t + 2t^2$, then G has an independent set of size at least $t - 1$.*

Proof. We assume that there exists some k-recoloring sequence from α to β of length at most $\ell = 2t + 2t^2$. Theorem 6.2.1 shows that this sequence contains a coloring γ that assigns at least $2t - 1$ different colors to V_B. Let $U = \{\gamma(b) \mid b \in V_B\}$ denote this color set with $|U| \geq 2t - 1$. Vertices in V_B cannot be assigned color $n + t + 1$ in a sequence of this length (Proposition 6.2.2), so $|U \cap \{1, \ldots, n\}| \geq t - 1$.

Since G' contains all edges between V_B and V_G, all vertices in $S = \{g_i \in V_G \mid i \in U\}$ must have $\gamma(g_i) \neq i$, and thus $\gamma(g_i) = n + t + 1$ (Proposition 6.2.2). It follows that S is an independent set of G of size at least $t - 1$. □

Theorem 6.2.5. COL-BOUND *is* `W[1]`*-hard when parameterized by ℓ.*

Proof. Given an instance $(G, t - 1)$ of INDEPENDENT SET, we construct the corresponding COL-BOUND instance $(G', k, \ell, \alpha, \beta)$, where $k = n + t + 1$ and $\ell = 2t + 2t^2$. From Lemma 6.2.3, we know that if G has an independent set of size $t - 1$ then there exists a k-recoloring sequence from α to β of length at most ℓ. From Lemma 6.2.4, if there exists a k-recoloring sequence from α to β of length at most ℓ then G has an independent set of size at least $t - 1$. Combining Lemmas 6.2.3 and 6.2.4 with the fact that INDEPENDENT SET parameterized by t is `W[1]`-hard yields the statement of the theorem. □

6.2.2 `FPT` algorithm

For any constant ℓ, the COL-BOUND problem can be solved in polynomial time using the following simple branching algorithm. We denote the given instance by $(G, k, \ell, \alpha, \beta)$, with $|V(G)| = n$. Starting with the initial k-coloring α, for each coloring generated by the algorithm, we consider all possible k-colorings that can be obtained from it using one recoloring step. We recurse on these choices, up to a recursion depth of at most ℓ. We have a yes-instance if and only if in one of the recursion branches the target coloring β is obtained. Clearly, this algorithm yields the correct answer. For one coloring, there are at most kn possible recoloring steps, so branching with depth ℓ implies that at most $\mathcal{O}((kn)^\ell)$ colorings will be considered. This shows that for parameter ℓ, the problem is in `XP`.

Because COL-BOUND is `NP`-hard for every constant $k \geq 4$, a similar result cannot be obtained for the parameter k unless `P` = `NP`. Since the COL-BOUND problem is `W[1]`-hard when parameterized by ℓ, we also do not expect to obtain any algorithm solving the problem in $\mathcal{O}(h(\ell)(kn)^{\mathcal{O}(1)})$ time (for some computable function h). However, we shall see in this section that the problem is fixed-parameter tractable when parameterized by $k + \ell$; it can be solved in $\mathcal{O}(f(k, \ell)n^{\mathcal{O}(1)})$ time, for some computable function f.

We first give a high-level description of the algorithm. We let $S = \{v \in V(G) \mid \alpha(v) \neq \beta(v)\}$ denote the set of vertices assigned different colors in α and β. Clearly, when $|S| > \ell$ we have a no-instance and when $|S| = 0$ we have a trivial yes-instance. In what follows, we assume $0 < |S| \leq \ell$. The main challenge that we need to overcome is that the number of vertices that potentially need to be recolored cannot easily be bounded by a function of ℓ; in particular, there may be too many vertices at distance at most ℓ from S. However, once we know which vertices will be recolored, the problem can be solved using a branching algorithm similar to the one above.

We let $R = \alpha_0, \ldots, \alpha_\ell$ be a recoloring sequence for a graph G. For every vertex $v \in V(G)$, the *used-color lists* for R are defined as $U(v) = \{\alpha_i(v) \mid i \in \{0, \ldots, \ell\}\}$. Note that a vertex v is recolored at least once in R if and only if $|U(v)| \geq 2$. In addition, we have the following simple but useful proposition.

Proposition 6.2.6. *If R is a recoloring sequence for G of length ℓ and U is the used-color lists for R, then $\sum_{v \in V(G)}(|U(v)| - 1) \leq \ell$.*

Our main algorithm to solve the Col-Bound problem is Algorithm 7, which uses the subroutine given in Algorithm 6. The Recolor algorithm (Algorithm 7) consists of a two-stage branching algorithm. The first stage of the algorithm ignores the ordering of recoloring steps and simply tries to "guess" the used-color lists for each vertex, assuming that a recoloring sequence R from α to β of length at most ℓ exists. These guesses are stored in the lists $L(v)$. Clearly, $\{\alpha(v), \beta(v)\} \subseteq L(v)$ should hold. To construct these lists, the algorithm maintains two disjoint sets of vertices A and B as follows. All vertices in $A \cup B$ will be recolored at least once, according to our current guess. Vertices are in B if we have already guessed a used-color list for them. Initially, we have $A = S$ and $B = \emptyset$. While A is not empty, we pick a vertex $v \in A$ and branch on all possible lists $L(v)$. We then delete v from A and add it to B. Before continuing with the next vertex from A, we inspect the neighbors of v. If there exists $u \in N_G(v) \setminus (A \cup B)$ such that $\alpha(u) \in L(v)$, we add u to A since u must also be recolored at least once, assuming that v will indeed receive all colors in $L(v)$.

If we reach a state where $\sum_{v \in B}(|L(v)| - 1) > \ell$, then the current branch is ignored since these lists cannot correspond to used-color lists of a recoloring sequence of length at most ℓ. On the other hand, when $A = \emptyset$ and $\sum_{v \in B}(|L(v)| - 1) \leq \ell$, we have a "possible solution". That is, we still need to make sure that there exists a feasible ordering of the recoloring steps that transforms α to β. This is handled by the ListRecolor subroutine (Algorithm 6), which is a branching algorithm similar to the one sketched in the beginning of this section, although we only assign colors from the lists $L(v)$. Since $\sum_{v \in B}(|L(v)| - 1) \leq \ell$, according to the lists at any time there are at most ℓ different ways to recolor a vertex. So, branching up to a depth of ℓ, this yields an FPT algorithm for (such instances of) the L-Col-Bound problem, parameterized by ℓ. Combined with Algorithm 7, which generates all relevant guesses for the used-color lists, this yields an FPT algorithm for the Col-Bound problem, parameterized by $k + \ell$. We now present the details of these algorithms, starting with the ListRecolor subroutine (Algorithm 6).

Algorithm 6 ListRecolor($G, L, \alpha, \beta, \ell$)

```
   Input: A graph $G$, nonnegative integer $\ell$, color lists $L(v)$ for all $v \in V(G)$,
   and two $L$-colorings $\alpha$ and $\beta$ of $G$.
   Output: "YES" if and only if there exists an $L$-recoloring sequence
   from $\alpha$ to $\beta$ of length at most $\ell$.

1: return ListRecolorRecurse($\alpha, \ell$);

   Subroutine ListRecolorRecurse($\gamma, \ell'$):

2: if $\gamma = \beta$ and $\ell' \geq 0$ then return "YES";
3: if $\ell' \leq 0$ then return "NO";
4: for each $L$-colorings $\delta$ that can be obtained from $\gamma$ by changing the
             color of a single vertex $x$ to a different color in $L(x)$:
5:     if ListRecolorRecurse($\delta, \ell' - 1$) = "YES" then return "YES";
6: return "NO";
```

Lemma 6.2.7. *For input $(G, L, \alpha, \beta, \ell)$ and $p = \sum_{x \in V(G)}(|L(x)| - 1)$, Algorithm 6 decides in time $\mathcal{O}(p^\ell \cdot poly(|V(G)|))$ whether there exists an L-recoloring sequence from α to β of length at most ℓ.*

Proof. An easy induction proof shows that a recursive call ListRecolorRecurse(γ, ℓ') in Algorithm 6 returns "YES" if and only if there exists an L-recoloring sequence from γ to β of length at most ℓ': Line 4 guarantees that every new δ that is generated is again an L-coloring, which is adjacent to γ in $R_{COL}(G, L)$. This shows that the algorithm is correct.

For every L-coloring γ, there are p ways to change the color of some vertex $x \in V(G)$ to a different color in $L(x)$. Thus, at most p new L-colorings are generated in one recursive call. Obviously, the recursion depth is at most ℓ, so this shows that at most $\mathcal{O}(p^\ell)$ recursive calls are made in total. One recursive call takes time poly($|V(G)|$), so this yields the stated complexity bound. □

Lemma 6.2.8. *For α and β two k-colorings for a graph G and $\ell \in \mathbb{N}$, the following conditions hold for every recursive call* RecolorRecurse(A, B, L) *made by Algorithm 7 on this input:*

1. *A and B are disjoint subsets of $V(G)$.*

2. *For every $u \in V(G) \setminus B$: $u \in A$ if and only if*

    (a) *$\alpha(u) \neq \beta(u)$, or*

    (b) *there exists an edge $uv \in E(G)$ with $v \in B$ and $\alpha(u) \in L(v)$.*

Proof. The first time the subroutine RecolorRecurse is called (in Line 4), $B = \emptyset$, and A contains exactly those vertices that have different colors in α and β, so clearly the above conditions are satisfied.

We now consider a call RecolorRecurse(A, B, L) where the arguments satisfy the given conditions. We need to show that for an iteration of the for-loop in Line 9, where a subsequent call RecolorRecurse(A', B', L') is made, all conditions still hold. Lines 8, 13 and 15 show that A' and B' are again disjoint (in Line 15, only vertices outside of B' are added), so Condition (1) is satisfied. For vertices in $A \setminus \{v\}$, it still holds that at least one of the Conditions (2a) and (2b) is satisfied (also with respect to the new lists L'). Lines 14 and 15 show that the newly added vertices in A' are exactly those that now satisfy Condition (2b), since $L'(v) = U$ and v is the only new vertex in B'. We conclude that both Conditions (1) and (2) are maintained for A', B', and L'. By induction, it follows that for every recursive call RecolorRecurse(A, B, L), the above conditions hold for A, B, and L. □

Lemma 6.2.9. *For α and β two k-colorings for a graph G and $\ell \in \mathbb{N}$, if Algorithm 7 returns "YES" then there exists a k-recoloring sequence for G from α to β of length at most ℓ.*

Proof. Consider a recursive call RecolorRecurse(A, B, L). If "YES" is returned in Line 6, then $A = \emptyset$, and there exists an L-recoloring sequence for $G[B]$ from $\alpha|_B$ to $\beta|_B$ of length at most ℓ. Since $A = \emptyset$, this also yields a valid recoloring sequence for the entire graph G, starting from α, of the same length. This is because any color that is assigned to a vertex $v \in B$ is chosen from $L(v) \subseteq [k]$, and therefore does not conflict with the color $\alpha(w)$ of any vertex $w \in V(G) \setminus B$ (by Condition (2b) from Lemma 6.2.8). In addition, Condition (2a) shows that the recoloring sequence that we obtain for G this way ends with β. We conclude that there exists a k-recoloring sequence from α to β for G, of length at most ℓ. If "YES" is returned by a subsequent recursive call in Line 16, then the claim follows by induction. □

Lemma 6.2.10. *For α and β two k-colorings for a graph G and $\ell \in \mathbb{N}$, if there exists a k-recoloring sequence for G from α to β of length at most ℓ then Algorithm 7 returns "YES".*

Algorithm 7 Recolor$(G, k, \ell, \alpha, \beta)$

```
    Input: A graph $G$, nonnegative integers $k$ and $\ell$, and
    two $k$-colorings $\alpha$ and $\beta$.
    Output: "YES" if and only if there exists a $k$-recoloring sequence
    from $\alpha$ to $\beta$ of length at most $\ell$.

 1: $S := \{v \in V(G) \mid \alpha(v) \neq \beta(v)\}$;
 2: if $|S| > \ell$ then return "NO";
 3: if $|S| = 0$ then return "YES";
 4: return RecolorRecurse$(S, \emptyset, \{f^{\emptyset}\})$;

    Subroutine RecolorRecurse$(A, B, L)$:

 5: if $\sum_{v \in B}(|L(v)| - 1) > \ell$ then return "NO";
 6: if $A = \emptyset$ then return ListRecolor$(G[B], L, \alpha|_B, \beta|_B)$;
 7: Choose $v \in A$;
 8: $B' := B \cup \{v\}$;
 9: for each $U \subseteq [k]$ with $2 \leq |U| \leq \ell$ and $\{\alpha(v), \beta(v)\} \subseteq U$:
10:     $L'(v) := U$;
11:     for each $u \in B$:
12:         $L'(u) := L(u)$;
13:     $A' := A \setminus \{v\}$;
14:     for each $u \in N(v) \setminus (A \cup B)$ with $\alpha(u) \in U$:
15:         $A' := A' \cup \{u\}$;
16:     if RecolorRecurse$(A', B', L')$ = "YES" then return "YES";
17: return "NO";
```

Proof. We prove by induction that for every call RecolorRecurse(A, B, L), "YES" is returned if (i) there exists a k-recoloring sequence R from α to β for G, of length at most ℓ, such that for every vertex $v \in B$, $L(v)$ is (exactly) the set of colors used by R for v. Applying this statement to the initial call RecolorRecurse$(S, \emptyset, \{f^{\emptyset}\})$ in Line 4 proves the lemma.

We show that if Condition (i) is satisfied for A, B, and L, then "YES" is returned. Let R be a corresponding recoloring sequence from α to β, of length at most ℓ, which uses exactly the colors $L(v)$ for each $v \in B$. First, Proposition 6.2.6 shows that "NO" is not returned in Line 5. If there are no vertices $u \in V(G) \setminus B$ that satisfy Condition (2a) or (2b), then $A = \emptyset$ (Lemma 6.2.8), so a call to ListRecolor is made in Line 6. Restricting all colorings in R to B yields a k-recoloring sequence of length at most ℓ for $G[B]$ from $\alpha|_B$ to $\beta|_B$ that uses exactly the colors in L for each vertex, so in this case, "YES" is returned.

In the remaining case, we may assume that A is nonempty, and hence a vertex $v \in A$ is chosen in Line 7. We let U be the set of colors used for v by the sequence R. Thus, $\{\alpha(v), \beta(v)\} \subseteq U$. In addition, we argue that $|U(v)| \geq 2$. If $\alpha(v) \neq \beta(v)$, this is obvious. Otherwise, by Condition (2) from Lemma 6.2.8, v has a neighbor $u \in B$ with $\alpha(v) \in L(u)$. Since R uses exactly the colors $L(u)$ for u, v must be recolored at least once, and thus $|U| \geq 2$. We conclude that U will be considered in an iteration of the for-loop in Line 9. Let L', A', and B' be the lists and sets constructed in this iteration. We observe that R satisfies the properties from Condition (i), with respect to this L' and B'. Indeed, $B' = B \cup \{v\}$, and $L'(v) = U$, which is exactly the set of colors used by R for v. For all other vertices $w \in B'$, $L(w) = L'(w)$. So we may use induction to conclude that "YES" is returned by the call RecolorRecurse(A', B', L'), and thus in Line 16, "YES" is returned. □

Lemma 6.2.11. *The* Recolor *algorithm (Algorithm 7) runs in $\mathcal{O}(2^{k(\ell+1)} \cdot \ell^\ell \cdot poly(n))$ time, where $n = |V(G)|$.*

Proof. The search tree produced by the Recolor algorithm has depth at most $\ell+1$, since every branch increases the quantity $\sum_{v \in B}(|L(v)| - 1)$ by at least 1 (Line 9) and the base case is reached whenever $\sum_{v \in B}(|L(v)| - 1) > \ell$ (Line 5). The number of sets considered in the for loop in Line 9 is at most 2^k, so every node in the search-tree has at most 2^k children. We conclude that at most $\mathcal{O}(2^{k(\ell+1)})$ recursive calls are made in total.

We now argue that for every recursive call, we spend at most $\mathcal{O}(\ell^\ell \cdot \text{poly}(n))$ time in total. For Line 6, this follows from Lemma 6.2.7, noting that whenever this line is reached, $p = \sum_{v \in B}(|L(v)| - 1) \leq \ell$ holds (Line 5). All other lines can easily be implemented to run in time poly(n). (Note that we may assume without loss of generality that $k \leq n + 1$.) We attribute the time spent in Lines 10–16 to the resulting recursive call in Line 16. This shows that the entire complexity can be bounded by $\mathcal{O}(2^{k(\ell+1)} \cdot \ell^\ell \cdot \text{poly}(n))$. □

Combining Lemmas 6.2.9, 6.2.10, and 6.2.11 yields the main result of this section:

Theorem 6.2.12. Col-Bound *is fixed-parameter tractable when parameterized by $k + \ell$.*

Proof. It follows from Lemmas 6.2.9 and 6.2.10 that the Recolor algorithm (Algorithm 7) is correct and returns "YES" if and only if the corresponding Col-Bound instance is a yes-instance. Moreover, the Recolor algorithm runs in $\mathcal{O}(2^{k(\ell+1)} \cdot \ell^\ell \cdot \text{poly}(n))$ time

(Lemma 6.2.11), hence Col-Bound is fixed-parameter tractable when parameterized by $k + \ell$. $\square$

6.3 General CSPs

An instance $(X, k, \mathcal{C})$ of the Constraint Satisfaction Problem (CSP) consists of a set X of n *variables*, which all can take on the values in $D = [k]$, and a set $\mathcal{C}$ of *constraints*. The set D is called the *domain* of the variables. For $B \subseteq X$, a function $f : B \to D$ is called a *value assignment* from B to D. A set U of value assignments from B to D is called a *VA-set* from B to D. In what follows, we will consider a fixed set X of variables, and consider VA-sets U for many different subsets $B \subseteq X$, but always for the same domain D, so we will omit D from the terminology and simply call U a *VA-set for* B, and elements of U *value assignments for* B. Every constraint $C \in \mathcal{C}$ is a tuple (T, R), where $T \subseteq X$, R is a VA-set for T, and $|T| \leq r$, i.e. constraints have *arity* at most r. The VA-set R is interpreted as the set of all value combinations that are allowed for the variables in T. A value assignment $f : X \to D$ is said to *satisfy* constraint $C = (T, R)$ if and only if $f|_T \in R$. If f satisfies all constraints in $\mathcal{C}$, f is called *valid* (for $\mathcal{C}$).

A Coloring instance can be seen as a CSP instance, where variables correspond to vertices and edges correspond to binary constraints, stating that the two incident vertices/variables cannot have the same color/value. CSP is a decision problem where the question is, given $(X, k, \mathcal{C})$, to determine whether there exists a valid value assignment to all variables in X. For many problems that can be formulated as CSP problems, the constraints $(T, R) \in \mathcal{C}$ are not explicitly given, since R would usually be prohibitively (exponentially) large. Instead, a simple and efficient algorithm is given that can verify whether the constraint is satisfied. The factor $g(\mathcal{C})$ in our complexity bounds accounts for this fact.

In order to study reconfiguration questions for CSP, we define two distinct value assignments $\alpha : X \to D$ and $\beta : X \to D$ to be adjacent if they differ on exactly one variable $v \in X$, i.e. if there exists a $v \in X$ such that $\alpha|_{X \setminus \{v\}} = \beta|_{X \setminus \{v\}}$. For a CSP instance $(X, k, \mathcal{C})$, the reconfiguration graph $R_{\text{CSP}}(X, \mathcal{C}, k)$ has as vertex set all value assignments from X to $[k]$ that are valid for $\mathcal{C}$, with adjacency as defined above. We consider the following problem.

CSP-Bound

Input: A CSP instance $(X, k, \mathcal{C})$ where every constraint has arity at most r, two valid value assignments α and β, and an integer ℓ.

Question: Is there a path of length at most ℓ from α to β in $R_{\text{CSP}}(X, \mathcal{C}, k)$?

Our main result, which we present in Section 6.3.1, is an FPT algorithm for CSP-Bound, parameterized by $\ell + k + r$. This result has many implications, besides the fact that it generalizes our result for Col-Bound. For instance, it follows that the PSPACE-complete SAT-Bound problem, as defined by Gopalan et al. [67], is FPT parameterized by $\ell + r$ ($k = 2$). In addition, it implies that Shortest Path-Bound is FPT parameterized by $\ell + k$ ($r = 2$), where k is an upper bound on the number of vertices in one distance layer [16, 17, 90], i.e. vertices at distance d from some fixed vertex in the graph.

This positive result prompts two further questions: Firstly, is it possible to also obtain an FPT algorithm for CSP-Bound for parameter $\ell + k$? Secondly, clearly any reconfiguration sequence from α to β has length at least p, where $p = |\{x \in X \mid \alpha(x) \neq \beta(x)\}$. Is it also possible to obtain an FPT algorithm for CSP-Bound for the above-guarantee parameter $(\ell - p) + k + r$? In Section 6.3.2, we give two W[2]-hardness results that show that the answer to these questions is negative (unless FPT $=$ W[2]). These W[2]-hardness results hold in fact for the restricted case of SAT instances with only Horn clauses. Together, these hardness results show that our FPT result for CSP-Bound is tight (assuming FPT $\neq$ W[1]). To obtain an FPT algorithm, all three variables ℓ, k, and r need to be part of the parameter. Table 6.2 summarizes our results and shows the complexity status of CSP-Bound for all different parameterizations in terms of ℓ, k, r and p (omitted parameter combinations follow directly from the given rows).

Table 6.2: Complexity landscape of the CSP-Bound problem.

parameter	complexity
$k + \ell + r$	FPT
$k + r$	para-NP-complete (ℓ unary) / para-PSPACE-complete (ℓ binary) (already for $k = 4$ and $r = 2$)
$k + \ell$	W[2]-hard (already for $k = 2$), in XP
$r + \ell$	W[1]-hard (already for $r = 2$), in XP
$k + r + \ell - p$	W[2]-hard (already for $k = 2$ and $r = 3$)

6.3.1 FPT algorithm

Our algorithm for CSP-Bound will closely follow the algorithm for Col-Bound, except here we need to bound the number of variables that potentially need to be reassigned different values from the domain. As every node in $R_{\text{CSP}}(X, \mathcal{C}, k)$ has at most kn neighbors, it follows that the CSP-Bound can be solved in $\mathcal{O}((kn)^\ell)$ time via the same simple branching algorithm discussed in Section 6.2.2 for the Col-Bound problem. Hence, the problem is in XP.

We let $S = \{x \in X \mid \alpha(x) \neq \beta(x)\}$. For a reconfiguration sequence $\mathcal{S} = \gamma_0, \ldots, \gamma_\ell$ and a set $B \subseteq X$, the *set of B-variable combinations used by $\mathcal{S}$* is $\text{Used}(\mathcal{S}, B) = \{\gamma_i|_B \ : \ i \in \{0, \ldots, \ell\}\}$. For U a VA-set for B, we say that $\mathcal{S}$ *follows* U if $\text{Used}(\mathcal{S}, B) \subseteq U$. We now give a branching algorithm for the following variant of CSP-Bound, which is restricted by choices of B and U and in some sense mimics the L-Col-Bound problem.

Lemma 6.3.1. *Let $(X, k, \mathcal{C}, \alpha, \beta, \ell)$ be an instance of* CSP-Bound*, and let $g(\mathcal{C})$ be the complexity of deciding whether a given value assignment for X satisfies $\mathcal{C}$. Let $B \subseteq X$, and U be a VA-set for B. Let $L(x) = \{f(x) \mid f \in U\}$ for all $x \in B$, and $p = \sum_{x \in B}(|L(x)| - 1)$. Then there exists an algorithm* ListCSPBound *(Algorithm 8) with complexity $\mathcal{O}(p^\ell \cdot g(\mathcal{C}) \cdot$ poly$(|U|, |X|))$, that decides whether there exists a reconfiguration sequence $\mathcal{S}$ for $(X, k, \mathcal{C})$ from α to β of length at most ℓ in which only variables in B are changed and such that* $\text{Used}(\mathcal{S}, B) \subseteq U$.

Proof. A recursive call ListCSPBoundRecurse(γ, ℓ') in Algorithm 8 returns "YES" if and only if there exists a reconfiguration sequence from γ to β of length at most ℓ' that follows U and changes variables only in B. To see why, it suffices to note that Lines 6 and 7 ensure that for a given value assignment γ, all value assignments δ that satisfy $\mathcal{C}$, are adjacent to γ in $R_{\text{CSP}}(X, \mathcal{C}, k)$, follow U, and only differ from γ on variables in B, are considered. Hence, the algorithm is correct.

Now we consider the complexity of Algorithm 8. For each value assignment γ, there are exactly p ways to change a single variable $x \in B$ to a value in $L(x) \setminus \{\gamma(x)\}$. (Recall that $p = \sum_{x \in B}(|L(x)| - 1)$.) Therefore, for every recursive call ListCSPBoundRecurse(γ, ℓ') with $\ell' > 0$, at most p adjacent value assignments δ are considered in the for loop in Line 6. Since the recursion depth is at most ℓ, the total number of times that Line 7 is visited during the entire computation is therefore bounded by $\mathcal{O}(p^\ell)$. For every time that Line 7 is visited while considering a value assignment δ, all computations related to this δ can easily be implemented to run in time $g(\mathcal{C}) \cdot \text{poly}(|U|, |X|)$. (This includes the construction of δ, testing the conditions in Line 7, and testing the condition in Line 4 in the subsequent recursive call.) Together, this yields the stated complexity bound. □

It remains to give a branching algorithm that, if there exists a reconfiguration sequence $\mathcal{S}$ of length at most ℓ, can determine a proper guess for the sets B of variables that are

Algorithm 8 ListCSPBound$(X, k, \mathcal{C}, \alpha, \beta, \ell, B, U)$

```
Input: A variable set $X$ with domains $[k]$, a set $\mathcal{C}$ of constraints on $X$,
value assignments $\alpha : X \to [k]$ and $\beta : X \to [k]$, an integer $\ell$, subset $B \subseteq X$,
and a VA-set $U$ for $B$.
Output: "YES" if and only if there exists a reconfiguration sequence for $(X, k, \mathcal{C})$,
from $\alpha$ to $\beta$, of length at most $\ell$, in which only variables of $B$ are changed,
that follows $U$.

1: for each $x \in B$:
2:     $L(x) := \{f(x) \mid f \in U\}$;
3: return ListCSPBoundRecurse$(\alpha, \ell)$;

Subroutine ListCSPBoundRecurse$(\gamma, \ell')$:

4: if $\gamma = \beta$ and $\ell' \geq 0$ then return "YES";
5: if $\ell' \leq 0$ then return "NO";
6: for each value assignments $\delta$ that can be obtained from $\gamma$ by changing the
           value of a single variable $x \in B$ to a different value in $L(x)$:
7:     if $\delta|_B \in U$ and $\forall (T, R) \in \mathcal{C}$: $\delta|_T \in R$ then
8:         if ListCSPBoundRecurse$(\delta, \ell' - 1)$="YES" then return "YES";
9: return "NO";
```

changed in $\mathcal{S}$ such that $U = \text{Used}(\mathcal{S}, B)$. Clearly, $S \subseteq B$ should hold, so we start with $B = S$ and we first consider all possible VA-sets U for this B. We will say that a constraint $C = (T, R)$ is *critical* for B, U, and α, if there exists an $f \in U$ such that the (unique) value assignment $g : X \to D$ that satisfies $g|_B = f$ and $g|_{X \setminus B} = \alpha|_{X \setminus B}$ does not satisfy C. Note that in this case, if we assume that the combination of values f occurs at some point during the reconfiguration, then for at least one variable in $T \setminus B$ the value must change before this point, so one such variable should be added to B, which yields a new set B'. We let $B \subseteq B' \subseteq X$, and let U and U' be VA-sets for B and B', respectively. We say that U' *extends* U if $U = \{f|_B \ : \ f \in U'\}$. In other words, if U and U' are interpreted as guesses of value combinations that will occur during the reconfiguration, then these guesses are consistent with each other.

For a given $B \subseteq X$ and VA-set U for B, we let $L(x) = \{f(x) \mid f \in U\}$ for all $x \in B$. If $\sum_{x \in B}(|L(x)| - 1) > \ell$ then the set U cannot correspond to the set $\text{Used}(\mathcal{S}, B)$ for a

Algorithm 9 CSPBound($X, k, \mathcal{C}, \alpha, \beta, \ell$)

```
    Input: A variable set $X = \{x_1, \ldots, x_n\}$ with domains $[k]$, a set $\mathcal{C}$ of constraints
    on $X$, valid value assignments $\alpha : X \to [k]$ and $\beta : X \to [k]$, and integer $\ell \geq 0$.
    Output: "YES" if and only if there exists a reconfiguration sequence of length at most $\ell$
    from $\alpha$ to $\beta$.

 1: $S := \{x \in X \mid \alpha(x) \neq \beta(x)\}$;
 2: if $|S| > \ell$ then return "NO";
 3: if $|S| = 0$ then return "YES";
 4: return CSPBoundRecurse($\emptyset, \{f^\emptyset\}, \{f^\emptyset\}$);

    Subroutine CSPBoundRecurse($B, U, L$):

 5: if $\sum_{v \in B}(|L(v)| - 1) > \ell$ then return "NO";
 6: if $S \subseteq B$ and there are no critical constraints for $U$, $B$ and $\alpha$ then
 7:         return ListCSPBound($X, k, \mathcal{C}, \alpha, \beta, \ell, B, U$);
 8: if not $S \subseteq B$ then
 9:         Let $i$ be the lowest index such that $x_i \in S \setminus B$;
10:         NewVar := $\{x_i\}$;
11: else
12:         choose a critical constraint $(T, R) \in \mathcal{C}$ for $U$, $B$ and $\alpha$;
13:         NewVar := $T \setminus B$;
14: for each $x \in$ NewVar:
15:         $B' := B \cup \{x\}$;
16:         for each VA-set$U'$ for $B'$ that extends $U$, with $|U'| \leq \ell$ and $\{\alpha|_{B'}, \beta|_{B'}\} \subseteq U'$:
17:                 $L(x) := \{f(x) \mid f \in U'\}$;
18:                 if $|L(x)| \geq 2$ then
19:                         if CSPBoundRecurse($B', U', L$)="YES" then return "YES";
20: return "NO";
```

reconfiguration sequence $\mathcal{S}$ of length at most ℓ, so this guess can be safely ignored. On the other hand, if a guess of B and U is reached where $\sum_{x \in B}(|L(x)| - 1) \leq \ell$ and there are no critical constraints, then the aforementioned ListCSPBound algorithm can be used to test whether there exists a corresponding reconfiguration sequence. Using these observations, it can be shown that Algorithm 9 correctly decides the CSP-Bound problem.

It is easy to see that the total number of recursive calls made by this algorithm is bounded by some function of ℓ, k and r, where $r = \max_{(T,R)\in\mathcal{C}} |T|$. Indeed, Line 18 guarantees that for every recursive call, the quantity $\sum_{v\in B}(|L(v)|-1)$ increases by at least one, so the recursion depth is at most $\ell + 1$ (see Line 5). The number of iterations of the for-loops in Lines 14 and 16 is bounded by $r - 1$, and by some function of ℓ and k, respectively. This shows that Algorithm 9 is an `FPT` algorithm for parameter $k + \ell + r$.

We first give three lemmas that together establish that Algorithm 9 returns "YES" if and only if there exists a reconfiguration sequence of length at most ℓ from α to β.

Lemma 6.3.2. *For every recursive call* CSPBoundRecurse(B, U, L) *made by Algorithm 9 and all variables $v \in B$, it holds that $L(v) = \{f(v) \mid f \in U\}$ and $|L(v)| \geq 2$.*

Proof. Considering Lines 15–19, we see that the conditions hold for every new vertex x that is added to B. Furthermore, for VA-sets U and U' for B and $B \cup \{x\}$ respectively, if U' extends U, then for all $y \in B$, $\{f(y) \mid f \in U\} = \{f(y) \mid f \in U'\}$. So by choice of U' (Line 16), the recursive calls maintain the conditions. □

Lemma 6.3.3. *If Algorithm 9 returns "YES" on input $(X, k, \mathcal{C}, \alpha, \beta, \ell)$, then there exists a reconfiguration sequence from α to β of length at most ℓ.*

Proof. If $\alpha = \beta$, then this is clearly a yes-instance (since $\ell \geq 0$), hence Line 3 is correct. For a recursive call CSPBoundRecurse(B, U, L), if "YES" is returned in Line 7, then Lemma 6.3.1 shows that there exists a reconfiguration sequence from α to β of length at most ℓ. If "YES" is returned by a subsequent recursive call in Line 19, then the claim follows by induction. That is, every such subsequent call will return "YES" only when Line 7 returns "YES". □

Lemma 6.3.4. *For input $(X, k, \mathcal{C}, \alpha, \beta, \ell)$, if there exists a reconfiguration sequence from α to β of length at most ℓ, then Algorithm 9 returns "YES".*

Proof. We assume that there exists such a reconfiguration sequence $\mathcal{S}$. Then $|S| \leq \ell$ holds for the set S constructed in Line 1, so Line 2 does not return "NO". Assuming that $|S| \geq 1$ (otherwise Line 3 returns "YES"), we show that the initial recursive call in Line 4 returns "YES". To this end, we prove the following proposition by induction. This yields the lemma statement, since any reconfiguration sequence $\mathcal{S}$ satisfies Used$(\mathcal{S}, \emptyset) = \{f^{\emptyset}\}$.

Proposition 6.3.5. *For every recursive call* CSPBoundRecurse(B, U, L), *if there exists a reconfiguration sequence $\mathcal{S}$ from α to β of length at most ℓ with* Used$(\mathcal{S}, B) = U$, *then "YES" is returned.*

We consider a recursive call CSPBoundRecurse(B, U, L) and a corresponding reconfiguration sequence $\mathcal{S} = \gamma_0, \dots, \gamma_p$, as stated above. For every $v \in X$, we let $L^{\mathcal{S}}(v) = \{\gamma_i(v) \mid i \in \{0, \dots, p\}\}$. Since $p \leq \ell$, it follows that $\sum_{v \in X}(|L^{\mathcal{S}}(v)| - 1) \leq \ell$. From the fact that $\textsc{Used}(\mathcal{S}, B) = U$ and from Lemma 6.3.2, it follows that for all $v \in B$, $L(v) = L^{\mathcal{S}}(v)$. So it also holds that $\sum_{v \in B}(|L(v)| - 1) \leq \ell$, and thus "NO" is not returned in Line 5.

Suppose that there are no critical constraints for U, B and α, and that $S \subseteq B$ (recall that $S = \{x \in X \mid \alpha(x) \neq \beta(x)\}$). Then for every $i \in \{0, \dots, p\}$, we define γ'_i to be the unique value assignment with $\gamma'_i|_B = \gamma_i|_B$ and $\gamma'_i|_{X \setminus B} = \alpha|_{X \setminus B}$. We argue that the sequence $\mathcal{S}' = \gamma'_0, \dots, \gamma'_p$ is also a reconfiguration sequence from α to β. Since there are no critical constraints, all these value assignments are valid, and because $S \subseteq B$, $\gamma'_p = \beta$. Clearly, $\textsc{Used}(\mathcal{S}', B) = \textsc{Used}(\mathcal{S}, B)$. Consequently, $\mathcal{S}'$ satisfies the conditions from Lemma 6.3.1, and thus "YES" is returned in Line 7.

Next, if $S \not\subseteq B$, then a new variable $x \in S \setminus B$ is chosen in Lines 9 and 10, and $B' = B \cup \{x\}$ is assigned in Line 15. We let $U' = \textsc{Used}(\mathcal{S}, B')$. Then it is easily seen that U' extends $U = \textsc{Used}(\mathcal{S}, B)$ and that U' satisfies the other conditions in Line 16, so U' is considered in an iteration of the for-loop in Line 16. By our choice of x, $|\{f(x) \mid f \in U'\}| \geq 2$ (this set contains $\alpha(x)$ and $\beta(x)$, which are distinct by the definition of S). Therefore, the condition in Line 18 is satisfied, and Line 19 is reached for these choices of x and U'. By induction, it follows that "YES" is returned in this line.

Finally, if $S \subseteq B$ and there exists a critical constraint, we let $(T, R) \in \mathcal{C}$ be such a constraint chosen in Line 12. For some $f \in U$, the value assignment $g : X \to D$ that satisfies $g|_B = f$ and $g|_{X \setminus B} = \alpha|_{X \setminus B}$ does not satisfy (T, R). Since $\textsc{Used}(\mathcal{S}, B) = U$, there exists an $i \in \{0, \dots, p\}$ such that $\gamma_i|_B = f$. However, since γ_i satisfies the constraint (T, R), there exists a variable $x \in T \setminus B$ with $\gamma_i(x) \neq \alpha(x)$. Consider the iteration of the for-loop in Line 14 where this variable x is considered. Let $U' = \textsc{Used}(\mathcal{S}, B \cup \{x\})$. Again, it is easily seen that U' is considered in an iteration of the for-loop in Line 16. By our choice of x, $|\{f(x) \mid f \in U'\}| \geq 2$ and $\alpha(x), \gamma_i(x) \in \{f(x) \mid f \in U'\}$. Therefore, the condition in Line 18 is satisfied, and it follows again by induction that "YES" is returned in Line 19 for these choices of x and U'. □

We conclude this section with our main result.

Theorem 6.3.6. *For an instance $(X, k, \mathcal{C}, \alpha, \beta, \ell)$ of* CSP-Bound*, it can be decided in time $\mathcal{O}\big((r-1)^{\ell} \cdot k^{\ell(\ell+1)} \cdot \ell^{\ell} \cdot g(\mathcal{C}) \cdot poly(k, \ell, n)\big)$ whether there exists a reconfiguration sequence from α to β of length at most ℓ, where $r = \max_{(T,R) \in \mathcal{C}} |T|$ and $n = |X|$, and where $g(\mathcal{C})$ denotes the time to find a constraint in $\mathcal{C}$ that is not satisfied by a given value assignment, if such a constraint exists.*

Proof. Lemmas 6.3.3 and 6.3.4 show that Algorithm 9 yields the correct answer. It remains to bound the complexity. We will first bound the number of recursive calls, which is the main challenge. To do so, we define a *leaf* of the recursion tree as a recursive call $\textsc{CSPBoundRecurse}(B, U, L)$ that terminates by returning an answer in Line 5 or 7 (i.e. that does not make any further recursive calls). We will bound the number of leaves.

First, we note that there may be distinct leaves $\textsc{CSPBoundRecurse}(B, U, L)$ and $\textsc{CSPBoundRecurse}(B', U', L')$ with $B = B'$, $U = U'$ (and thus $L = L'$). This can happen when the variables have been added to B and B' in different orders. Because of this, for every leaf $\textsc{CSPBoundRecurse}(B, U, L)$, we define the tuple B^* to consist of all elements of B, in the order in which they have been added during the recursion. For every such leaf we have a VA-set U for B. We use this to define a *standardized VA-set* U^* from $[p]$ to $[k]$, where $p = |B|$, as follows. If $B^* = (x_1, \ldots, x_p)$, then every $f \in U$ is translated to a value assignment $f^* \in U^*$ with $f^*(i) = f(x_i)$, for all $i \in [p]$. Doing this for all $f \in U$ yields the VA-set U^* for $[p]$ (with $|U^*| = |U|$). We observe that two distinct leaves $\textsc{CSPBoundRecurse}(B_1, U_1, L_1)$ and $\textsc{CSPBoundRecurse}(B_2, U_2, L_2)$ cannot satisfy both $B_1^* = B_2^*$ and $U_1^* = U_2^*$, because these tuples and VA-sets completely determine all branching choices that are made during the computation (that is, they determine the next variable x chosen in Line 14 and the next VA-set U' chosen in Line 16). Thus, the number of leaves is bounded by the maximum number of possible combinations of B^* and U^*. For a given $p \in \{1, \ldots, \ell\}$, there are k^p possible value assignments from $[p]$ to $[k]$, and therefore $\mathcal{O}(k^{p\ell})$ possible VA-sets U^* from $[p]$ to $[k]$ with $|U^*| \leq \ell$. Since $|L(x)| \geq 2$ holds for every $x \in B$, Line 5 guarantees that $|B| \leq \ell + 1$ for every B that is considered. Consequently $p \leq \ell + 1$, and thus there are $\mathcal{O}(k^{\ell(\ell+1)})$ possible VA-sets U^* in total that can be considered in leaves. The total number of possible tuples B^* cannot easily be bounded by a function of $k + \ell + r$, but for every U^*, we can bound the total number of tuples B^* such that B^* and U^* can belong to a leaf together. Consider a (non-leaf) recursive call $\textsc{CSPBoundRecurse}(B, U, L)$. If $S \not\subseteq B$ then B is extended with the variable $x_i \in S \setminus B$ chosen in Line 9. This gives a unique choice for the next element in B^*, and this occurs $|S|$ times. On the other hand, if $S \subseteq B$, then there exists a critical constraint for U, B, and α. We let (T, R) be the constraint chosen in Line 12. Since this constraint is critical, $|T \cap B| \geq 1$ and there are at most $r - 1$ possible choices for the next variable x to add to B^* (Lines 13,14). The recursion depth is at most $\ell + 1$ (Line 5 and Lemma 6.3.2), therefore such a variable choice is made at most $\ell - |S| + 1$ times. This shows that for every U^*, there are at most $(r-1)^{\ell+1-|S|}$ possible tuples B^* that can occur together with U^* in a leaf. We conclude that the total number of leaves is bounded by $\mathcal{O}((r-1)^{\ell+1-|S|} \cdot k^{\ell(\ell+1)})$. Since every leaf has at most ℓ ancestors in the recursion tree, at most $\mathcal{O}(\ell \cdot (r-1)^{\ell+1-|S|} \cdot k^{\ell(\ell+1)}) \subseteq \mathcal{O}(\ell \cdot (r-1)^{\ell} \cdot k^{\ell(\ell+1)})$ recursive calls are made in total.

We now argue that for every recursive call, we spend at most $\mathcal{O}(\ell^\ell \cdot g(\mathcal{C}) \cdot \text{poly}(k, \ell, n))$ time in total. For Line 7 this follows from Lemma 6.3.1, noting that $|U| \leq \ell$ always holds and $p \leq \ell$ holds whenever Line 7 is visited. Lines 6 and 12 can be executed in time $g(\mathcal{C})$ by definition. We attribute the time spent in Lines 16–19 to generate new VA-sets U' to the resulting recursive call (This estimation is justified since only a small fraction of the newly generated VA-sets U' does not satisfy the condition in Line 18, so we lose only a constant factor). Using a proper implementation, these VA-sets can be generated and tested in time $\text{poly}(\ell, k)$ per VA-set. This yields the stated complexity bound. $\square$

6.3.2 Hardness results

In this section, we show that if we drop any of the three parameters ℓ, k, or r, then CSP-Bound is no longer in `FPT`. To do so, we give two `W[2]`-hardness results which hold for very restricted types of CSP instances. A CSP instance $(X, k, \mathcal{C})$ is called a Horn-SAT *instance* if $k = 2$ and every constraint in $\mathcal{C}$ can be formulated as a Boolean clause that uses at most one positive literal. (As is customary in Boolean satisfiability, we assume in this case that the variables can take on the values 0 and 1.) The Horn-SAT-Bound problem is the CSP-Bound problem restricted to Horn-SAT instances. The even more restricted problem where all clauses have three variables is called Horn-3SAT-Bound.

In both proofs, we give reductions from the `W[2]`-hard Hitting Set problem. Recall that a Hitting Set instance $(\mathcal{U}, \mathcal{F}, p)$ consists of a finite universe $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$, and a positive integer p. The question is whether there exists a subset $U \subseteq \mathcal{U}$ of size at most p such that for every set $F \in \mathcal{F}$ we have $F \cap U \neq \emptyset$. This problem is `W[2]`-hard when parameterized by p [46].

Theorem 6.3.7. Horn-SAT-Bound *is* `W[2]`*-hard when parameterized by* ℓ.

Proof. Given an instance $(\mathcal{U}, \mathcal{F}, p)$ of Hitting Set, we create a variable x_u for each element $u \in \mathcal{U}$ and two additional variables, y_1 and y_2, for a total of $|\mathcal{U}| + 2$ variables. For each set $\{u_1, u_2, \ldots u_t\} \in \mathcal{F}$, we create a Horn clause $(y_1 \vee \overline{y_2} \vee \overline{x_{u_1}}, \overline{x_{u_2}}, \ldots \overline{x_{u_t}})$. Finally, we add an additional clause $(y_2 \vee \overline{y_1})$. These clauses constitute a Horn formula $\mathcal{H}$ with $|\mathcal{F}| + 1$ clauses. Let α be the satisfying assignment for $\mathcal{H}$ that sets all its variables to 1, and β be the satisfying assignment for $\mathcal{H}$ that sets $y_1 = y_2 = 0$ and all other variables to 1.

We show that $\mathcal{F}$ has a hitting set of size at most p if and only if there is a reconfiguration sequence of length at most $2p + 2$ between α and β. Before we can set y_2 to 0, y_1 has to be set to 0. Before y_1 can be set to 0, some of the x variables (i.e. variables corresponding to elements of the universe $\mathcal{U}$) have to be set to 0 to satisfy all the clauses corresponding to the sets. If $\mathcal{F}$ has a hitting set of size at most p, we first set the (at most p) variables corresponding to the elements of the hitting set to 0 one by one, then set y_1 to 0, followed by setting y_2 to 0, and then revert back the (at most p) variables from 0 to 1, resulting in a reconfiguration sequence of length at most $2p + 2$.

To show the converse, suppose there is a reconfiguration sequence of length at most $2p + 2$ from α to β. At most $2p$ of these steps will involve x variables. Consider the first time the value of y_1 is changed from 1 to 0. In the previous assignment, and hence in the resulting assignment, y_2 must be 1 to satisfy the clause $(y_2 \vee \overline{y_1})$. Hence, in the resulting assignment all clauses corresponding to the sets are satisfied by x variables set to 0. Clearly, the set of these variables intersects every clause, and hence the corresponding elements of $\mathcal{U}$ form a hitting set of $\mathcal{F}$. Combining the fact that at most $2p$ reconfiguration steps will involve x variables with the fact that such variables have to be set back to 1 in β, it follows that $\mathcal{F}$ has a hitting set of size at most p. □

Theorem 6.3.7 implies (in particular) that for CSP-Bound, there is no FPT algorithm when parameterized only by $k + \ell$, unless FPT = W[2]. Next, we consider the above-guarantee parameterization of CSP-Bound. Given two valid value assignments α and β for X and $[k]$, we let $S = \{x \in X \mid \alpha(x) \neq \beta(x)\}$. Clearly, the length of any reconfiguration sequence from α to β is at least $|S|$. Hence, in the above-guarantee parameterization of the problem, instead of allowing the running time to depend on the full length ℓ of a reconfiguration sequence, we let $\bar{\ell} = \ell - |S|$ and allow the running time to depend on $\bar{\ell}$ only. However, the next theorem implies that no FPT algorithm for CSP-Bound exists, when parameterized by $\bar{\ell} + k + r$, unless FPT = W[2].

Theorem 6.3.8. *Horn-3SAT-Bound is* W[2]*-hard when parameterized by $\bar{\ell} = \ell - |S|$, where $S = \{x \in X \mid \alpha(x) \neq \beta(x)\}$.*

Proof. Starting from a Hitting Set instance $(\mathcal{U}, \mathcal{F}, p)$, we first create a variable x_u for each $u \in \mathcal{U}$. We let $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ and $\{u_1, u_2, \ldots, u_r\}$ be a set in $\mathcal{F}$. For each such set in $\mathcal{F}$, we create r new variables $y_1, y_2, \ldots, y_r$ and the clauses $(y_1 \vee \overline{x_{u_1}} \vee \overline{y_2})$, $(y_2 \vee \overline{x_{u_2}} \vee \overline{y_3})$, …, $(y_r \vee \overline{x_{u_r}} \vee \overline{y_1})$. We let α be the satisfying assignment for the formula with all variables set to 1, and let β be the satisfying assignment with all the x_u, $u \in \mathcal{U}$, variables set to 1 and the rest set to 0.

Consider the clauses corresponding to a set $\{u_1, u_2, \ldots, u_r\}$ in $\mathcal{F}$, with variables $y_1, \ldots, y_r$. None of the y variables can be set to 0 before we set at least one x variable to 0. Moreover, after setting any x variable to 0, we can in fact set all y variables to 0, provided this is done in the proper order. Combining the previous observations with the fact that

$|S| = \sum_{i=1}^{m} |F_i|$, we show that $\mathcal{F}$ has a hitting set of size at most p if and only there is a reconfiguration sequence of length at most $\sum_{i=1}^{m} |F_i| + 2p$ from α to β.

Consider the clauses corresponding to a set $\{u_1, u_2, \ldots u_r\}$ in $\mathcal{F}$. There exist $(y_1 \vee \overline{x_{u_1}} \vee \overline{y_2})$, $(y_2 \vee \overline{x_{u_2}} \vee \overline{y_3})$, $\ldots$, $(y_{r-1} \vee \overline{x_{u_{r-1}}} \vee \overline{y_r})$, $(y_r \vee \overline{x_{u_r}} \vee \overline{y_1})$, and none of the y variables can be set to 0 before we set at least one x variable to 0. Moreover, after setting any x variable to 0, we can set all y variables to 0 in the following order: Assume we set x_{u_r} to 0 first. Then, y_r can be set to 0 followed by y_{r-1}, y_{r-2}, and up to y_1. Hence, if $\mathcal{F}$ has a hitting set of size at most p, we can set (one by one) the x variables corresponding to the elements of the hitting set to 0, then set all y variables to 0, and finally set back the x variables to 1. It is not hard to see that this corresponds to a reconfiguration sequence of length $\sum_{i=1}^{m} |F_i| + 2p$ from α to β.

For the converse, assume that there is a reconfiguration sequence of length at most $\sum_{i=1}^{m} |F_i| + 2p$ from α to β. At most $2p$ of these steps will involve x variables. Since all x variables are set to 1 in β, the value of each x variable must change at least twice. Moreover, the value of at least one x variable is required to change for setting all y variables in the clauses corresponding to each set in $\mathcal{F}$. Putting it all together, we know that the number of x variables whose values will change is at most p and the corresponding elements of $\mathcal{U}$ must form a hitting set of $\mathcal{F}$ of size at most p. □

6.4 Classification under Schaefer's framework

In Theorem 4.2.2, we have seen that for a large number of graph vertex-subset problems the bounded reachability problem parameterized by ℓ is hard. In contrast, Theorem 6.3.6 states that the problem is easy (in the fixed-parameter tractable sense) for CSP. Intuitively, the main difference between those two theorems is that in the latter we do not impose any restrictions on how many variables can take the same value from the domain. In this section, we consider the CSP problem with such restrictions and a domain of size two. Namely, we will consider the Boolean SAT problem augmented with "weights" and show that the complexity landscape changes completely when weights come into play. Using a combination of known results from the literature combined with some of our earlier results, we will provide a classification of different instances of the problem under Schaefer's framework [125]. Schaefer's framework provides a way to classify Boolean formulas and was first used by Schaefer to show that for any class that can be defined using the framework, deciding whether a formula of that class has a satisfying assignment is either in `P` or `NP`-complete. Schaefer's framework has previously been used by Gopalan et al. [67], Schwerdtfeger [126], and Mouawad et al. [108] to provide similar characterizations for the reachability and bounded reachability variants of the Boolean SAT problem under classical complexity assumptions.

We start with the required notation and definitions. We use terminology originally introduced by Schaefer [125] and adapted to reconfiguration by Gopalan et al. [67] and Schwerdtfeger [126]. An *r-ary Boolean logical relation* (or *relation* for short) R is defined as a subset of $\{0,1\}^r$, where $r \geq 1$. Each $i \in \{1,\ldots,r\}$ can be interpreted as a variable of R such that R specifies exactly which assignments of values to the variables are to be considered satisfying. Every element $\mathbf{v} \in R$ is an *r-bit vector* and we denote such vectors using bold lowercase letters.

For any r-ary relation R and positive integer $r' \leq r$, we define an r'-ary *restriction* of R to be any r'-ary relation R' that can be obtained from R by substitution with constants and identification of variables. More precisely, let $X : \{1,\ldots,r\} \to \{1,\ldots,r'\} \cup \{c_0, c_1\}$ be a mapping from the variables of R to the variables of R' and the constants 0 and 1. Any such X defines a mapping $f_X : \{0,1\}^{r'} \to \{0,1\}^r$ as follows. For $\mathbf{v} \in \{0,1\}^{r'}$, let $f_X(\mathbf{v})$ be the r-bit vector whose i^{th} bit is 0 if $X(i) = c_0$, 1 if $X(i) = c_1$, and equal to the $X(i)^{th}$ bit of $\mathbf{v}$ otherwise. We say that an r'-ary relation R' is a *restriction of R with respect to* $X : \{1,\ldots,r\} \to \{1,\ldots,r'\} \cup \{c_0, c_1\}$ if $\mathbf{v} \in R' \Leftrightarrow f_X(\mathbf{v}) \in R$.

A Boolean formula ϕ over a set $\{x_1,\ldots,x_n\}$ of variables defines a relation R_ϕ as follows. For any n-bit vector $\mathbf{v} \in \{0,1\}^n$, we interpret $\mathbf{v}$ as the assignment to the variables of ϕ where x_i is set to be equal to the i^{th} bit of $\mathbf{v}$. We then say that $\mathbf{v} \in R_\phi$ if and only if $\mathbf{v}$ is a satisfying assignment. The *Hamming weight* of $\mathbf{v}$, $hw(\mathbf{v})$, is equal to the number of bits set to 1 in $\mathbf{v}$. The *Hamming distance* between two n-bit vectors $\mathbf{u}$ and $\mathbf{v}$, $hd(\mathbf{u},\mathbf{v})$, is equal to the absolute value of the difference of their Hamming weights, i.e. $hd(\mathbf{u},\mathbf{v}) = |hw(\mathbf{u}) - hw(\mathbf{v})|$.

A *CNF formula* is a Boolean formula of the form $C_1 \wedge \ldots \wedge C_m$, where each C_i, $1 \leq i \leq m$, is a *clause* consisting of a finite disjunction of *literals* (variables or negated variables). A *rCNF formula*, $r \geq 1$, is a CNF formula where each clause has at most r literals. A CNF formula is *Horn* (*dual Horn*) if each clause has at most one positive (negative) literal. A CNF formula is *monotone* (*anti-monotone*) if each clause has only positive (negative) literals.

For a finite set of relations $\mathcal{S}$, a *CNF($\mathcal{S}$) formula* over a set of n variables $\{x_1,\ldots,x_n\}$ is a finite collection $\{C_1,\ldots,C_m\}$ of clauses. Each C_i, $1 \leq i \leq m$, is defined by a tuple (R_i, X_i), where R_i is an r_i-ary relation in $\mathcal{S}$ and $X_i : \{1,\ldots,r_i\} \to \{1,\ldots,n\} \cup \{c_0, c_1\}$ is a function. Each X_i defines a mapping $f_{X_i} : \{0,1\}^n \to \{0,1\}^{r_i}$ and we say that an assignment $\mathbf{v}$ to the variables satisfies ϕ if and only if for all $i \in \{1,\ldots,m\}$, $f_{X_i}(\mathbf{v}) \in R_i$. For any variable x_j, we say that x_j *appears in* clause C_i if $X_i(q) = j$ for some $q \in \{1,\ldots,r_i\}$ and for any assignment $\mathbf{v}$ to the variables of ϕ, we say that $f_{X_i}(\mathbf{v})$ is the assignment induced by $\mathbf{v}$ on R_i.

For example, to represent the class 3CNF in Schaefer's framework, we specify $\mathcal{S}$ as follows. Let $R^0 = \{0,1\}^3 \backslash \{000\}$, $R^1 = \{0,1\}^3 \backslash \{100\}$, $R^2 = \{0,1\}^3 \backslash \{110\}$, $R^3 = \{0,1\}^3 \backslash \{111\}$, and $\mathcal{S} = \{R^0, R^1, R^2, R^3\}$. Since R^i can be used to represent all 3-clauses with exactly i negative literals (regardless of the positions in which they appear in a clause), clearly CNF($\mathcal{S}$) is exactly the class of 3CNF formulas.

Below we define some classes of relations used in the literature and relevant to our work. Note that componentwise bijunctive, OR-free, and NAND-free were first defined by Gopalan et al. [67]. Schwerdtfeger [126] later modified them slightly and defined safely component-wise bijunctive, safely OR-free, and safely NAND-free. We reuse the names componentwise bijunctive, OR-free, and NAND-free for Schwerdtfeger's safely componentwise bijunctive, safely OR-free, and safely NAND-free respectively.

Definition 6.4.1. *For a k-ary relation R:*

- *R is* bijunctive *if it is the set of satisfying assignments of a 2CNF formula.*
- *R is* Horn *(*dual Horn*) if it is the set of satisfying assignments of a Horn (dual Horn) formula.*
- *R is* affine *if it is the set of satisfying assignments of a formula $x_{i_1} \oplus \ldots \oplus x_{i_h} \oplus c$, with $i_1, \ldots, i_h \in \{1, \ldots, k\}$ and $c \in \{0,1\}$. Here $\oplus$ denote the exclusive OR operation which evaluates to* 1 *when exactly one of the values it operates on is* 1 *and evaluates to* 0 *otherwise.*
- *R is* componentwise bijunctive *if every connected component of the reconfiguration graph of R and of the reconfiguration graph of every restriction R' of R induces a bijunctive relation.*
- *R is* OR-free *(*NAND-free*) if there does not exist a restriction R' of R such that $R' = \{01, 10, 11\}$ ($R' = \{01, 10, 00\}$).*
- *R is* Horn-free *if there does not exist a restriction R' of R such that $R' = \{0,1\}^3 \setminus \{011\}$, or equivalently, R' is the set of all satisfying assignments of the clause $(x \vee \overline{y} \vee \overline{z})$ for some three variables x, y, and z.*
- *R is* dual-Horn-free *if there does not exist a restriction R' of R such that $R' = \{0,1\}^3 \setminus \{100\}$, or equivalently, R' is the set of all satisfying assignments of the clause $(\overline{x} \vee y \vee z)$ for some three variables x, y, and z.*

Definition 6.4.2 ([67, 108])**.** *A set $\mathcal{S}$ of relations is* tight *if at least one of the following three conditions holds:*

(1) Every relation in $\mathcal{S}$ is component-wise bijunctive.

(2) Every relation in $\mathcal{S}$ is OR-free.

(3) Every relation in $\mathcal{S}$ is NAND-free.

A set $\mathcal{S}$ of relations is navigable *if at least one of the following three conditions holds (note that if $\mathcal{S}$ is navigable then it is also tight):*

(1) Every relation in $\mathcal{S}$ is component-wise bijunctive.

(1) Every relation in $\mathcal{S}$ is OR-free and Horn-free.

(2) Every relation in $\mathcal{S}$ is NAND-free and dual-Horn-free.

Note that under Schaefer's framework, we can assume without loss of generality that all relations have the same constant arity r. In the SAT($\mathcal{S}$) problem, given a CNF($\mathcal{S}$) formula ϕ, the goal is to determine if ϕ is satisfiable. Using his framework, Schaefer [125] showed that SAT($\mathcal{S}$) is in P if every relation in $\mathcal{S}$ is bijunctive, Horn, dual Horn, or affine, and is NP-complete otherwise. Since Schaefer's original paper, a myriad of problems about Boolean formulas have been analyzed, and similar divisions into equivalence classes obtained [31]. We will also consider the following weighted variants of the problem. In the MIN-ONES-SAT($\mathcal{S}$) problem, given a CNF($\mathcal{S}$) formula ϕ and integer k, the goal is to determine if ϕ has a satisfying assignment with at most k variables set to 1. In the MAX-ONES-SAT($\mathcal{S}$) problem, given a CNF($\mathcal{S}$) formula ϕ and integer k, the goal is to determine if ϕ has a satisfying assignment with at least k variables set to 1.

Given a CNF($\mathcal{S}$) formula ϕ, the node set of the reconfiguration graph $R_{\text{SAT}}(\phi, 0, n)$ consists of all satisfying assignments of ϕ whose Hamming weight is between 0 and n and two nodes are adjacent whenever the Hamming distance between the two corresponding assignments is exactly one. In other words, there exists a corresponding node in $V(R_{\text{SAT}}(\phi, 0, n))$ for every $\mathbf{v} \in R_\phi$ such that $0 \leq hw(\mathbf{v}) \leq n$; $\mathbf{u}, \mathbf{v} \in V(R_{\text{SAT}}(\phi, 0, n))$ share an edge whenever $hd(\mathbf{u}, \mathbf{v}) = 1$. A reconfiguration sequence in $R_{\text{SAT}}(\phi, 0, n)$ corresponds to a sequence of *flips* of variables from 1 to 0 or 0 to 1 and is therefore also called a *flip sequence*. Flips that change the value of a variable from 1 to 0 (0 to 1) are called *negative flips* (*positive flips*). We will denote a flip sequence in which all positive flips precede all negative flips by a *positive canonical sequence*. A *negative canonical sequence* is defined similarly. We call a path in $R_{\text{SAT}}(\phi, 0, n)$ *monotonically increasing* if the Hamming weights of the nodes on the path increase monotonically, and define a *monotonically decreasing* path similarly. We represent flip sequences as edit sequences, where vertex markers are replaced by variable markers and addition/removal markers are replaced by positive/negative flips in the obvious way. For a tuple $t = (x_i, \ldots, x_j)$ of variables, $1 \leq i \leq j \leq n$, and an n-bit vector $\mathbf{v}$, we use $\mathbf{v}^t$ to denote the bit-vector restricted to $x_i, \ldots, x_j$.

$\mathcal{S}$-REACH
Input: A CNF($\mathcal{S}$) formula ϕ and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$.
Question: Is there a path from $\mathbf{v_s}$ to $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$?

$\mathcal{S}$-BOUND
Input: A CNF($\mathcal{S}$) formula ϕ, integer ℓ, and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$.
Question: Is there a path of length at most ℓ from $\mathbf{v_s}$ to $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$?

Theorem 6.4.3 is due to Gopalan et al. [67] and Theorem 6.4.4 is due to Mouawad et al. [108]. We refer the reader to the thesis of Pathak for a detailed exposition of the latter result [118].

Theorem 6.4.3 ([67]). *$\mathcal{S}$-REACH is in* `P` *if $\mathcal{S}$ is tight and* `PSPACE`*-complete otherwise.*

Theorem 6.4.4 ([108]). *$\mathcal{S}$-BOUND is in* `P` *if $\mathcal{S}$ is navigable,* `NP`*-complete if $\mathcal{S}$ is tight but not navigable, and* `PSPACE`*-complete otherwise.*

The additional reachability and bounded reachability problems we consider are:

- $\mathcal{S}$-MIN-ONES-REACH: Given a CNF($\mathcal{S}$) formula ϕ, integer k, and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of weight at most k, determine if there is a path from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, k)$.
- $\mathcal{S}$-MAX-ONES-REACH: Given a CNF($\mathcal{S}$) formula ϕ, integer k, and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of Hamming weight at least k, determine if there is a path from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, k, n)$.
- $\mathcal{S}$-MIN-ONES-BOUND: Given a CNF($\mathcal{S}$) formula ϕ, integers k and ℓ, and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of Hamming weight at most k, determine if there is a path of length at most ℓ from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, k)$.

- $\mathcal{S}$-MAX-ONES-BOUND: Given a CNF($\mathcal{S}$) formula ϕ, integers k and ℓ, and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of Hamming weight at least k, determine if there is a path of length at most ℓ from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, k, n)$.

We present four lemmas which, combined with Theorems 6.4.3 and 6.4.4 above and Theorems 4.2.2, 5.1.2, 6.3.6, and 6.3.8, will be enough to prove most of our results. The following lemmas have appeared (in a slightly different form) in the work of Gopalan et al. [67] and Mouawad et al. [108].

Lemma 6.4.5. *For $r \geq 3$ and R an r-ary relation, if R is OR-free then it is dual-Horn-free. Similarly, if R is NAND-free then it is Horn-free.*

Proof. Assume that R is OR-free but not dual-Horn-free. Then there exists a restriction R' of R such that $R' = \{0,1\}^3 \setminus \{100\}$. It is easy to see that, from R', one can obtain $R'' = \{01, 10, 11\}$ by setting one of the three variables in R' to 0, resulting in a contradiction. A similar proof shows that NAND-free relations are Horn-free. □

Lemma 6.4.6. *Let ϕ be a CNF($\mathcal{S}$) formula, $\mathbf{v_s}$ and $\mathbf{v_t}$ be two satisfying assignments of ϕ, and $\sigma = f_1 \ldots f_\ell$ be a flip sequence from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$, where $f_i \in \{x_1^+, \ldots, x_n^+\} \cup \{x_1^-, \ldots, x_n^-\}$, $1 \leq i \leq \ell$.*

(1) If every relation in $\mathcal{S}$ is OR-free and there exists $i \in \{1, \ldots, \ell - 1\}$ such that $f_i = x^+$ is a positive flip and $f_{i+1} = y^-$ is a negative flip ($x \neq y$), then $\sigma' = f_1 \ldots f_{i-1} f_{i+1} f_i \ldots f_q$ is also a flip sequence from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$.

(2) If every relation in $\mathcal{S}$ is NAND-free and there exists $i \in \{1, \ldots, \ell - 1\}$ such that $f_i = x^-$ is a negative flip and $f_{i+1} = y^+$ is a positive flip ($x \neq y$), then $\sigma' = f_1 \ldots f_{i-1} f_{i+1} f_i \ldots f_q$ is also a flip sequence from $\mathbf{v_s}$ and $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$.

Proof. We will prove only case (1) as the other case follows by reversing the roles of positive and negative flips. Let $\mathbf{u}$ be the n-bit vector right before applying f_i, $\mathbf{v}$ be the n-bit vector after applying f_i but before applying f_{i+1}, and $\mathbf{w}$ be the one right after applying f_{i+1}. Thus it is clear that $\mathbf{u}^{(x,y)} = 01, \mathbf{v}^{(x,y)} = 11$, and $\mathbf{w}^{(x,y)} = 10$. Also, notice that since no other variables are flipped between $\mathbf{u}, \mathbf{v}$, and $\mathbf{w}$, the values of all variables other than x and y remain the same in $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$. Let t be the n-bit vector whose value is the same as $\mathbf{u}, \mathbf{v}$, and $\mathbf{w}$ on all variables except x and y and $t^{(x,y)} = 00$. If t is not a satisfying assignment of ϕ, then there exists a clause in ϕ which contains variables x and y and is no longer satisfied if x and y are both set to 0 but is satisfied for the remaining three value combinations for

x and y. Hence, the substitution described above gives us the relation $\{10, 01, 11\}$ on x and y in this clause, which is precisely the OR relation. Since R is OR-free, t must be a satisfying assignment of ϕ and thus we can swap the flips f_{i+1} and f_i.

Using similar arguments we can show that case (2) holds, as otherwise it would imply the existence of a NAND relation. □

Lemma 6.4.7. *For ϕ a CNF($\mathcal{S}$) formula and $\mathbf{v_s}$ and $\mathbf{v_t}$ two satisfying assignments of ϕ, if every relation in $\mathcal{S}$ is OR-free (NAND-free), then any reconfiguration sequence σ of length ℓ from $\mathbf{v_s}$ and $\mathbf{v_t}$ can be converted into a negative (positive) canonical sequence of length at most ℓ.*

Proof. If σ is not a negative (positive) canonical sequence, it must have a negative (positive) flip followed by a positive (negative) flip. If both flips act on the same variable, we cancel them out; otherwise, we swap them using the proof of Lemma 6.4.6. Doing this repeatedly gives us the required canonical sequence. Note that the order among the flips of the same sign is preserved since we never swap two flips of the same sign. □

Lemma 6.4.8. *If every relation in $\mathcal{S}$ is OR-free and ϕ is a CNF($\mathcal{S}$) formula, then the diameter of each connected component of $R_{\text{SAT}}(\phi, 0, n)$ is at most $2n$ and the diameter of each connected component of $R_{\text{SAT}}(\phi, 0, k)$ is at most $2k$. If every relation in $\mathcal{S}$ is NAND-free and ϕ is a CNF($\mathcal{S}$) formula, then the diameter of each connected component of $R_{\text{SAT}}(\phi, 0, n)$ is at most $2n$ and the diameter of each connected component of $R_{\text{SAT}}(\phi, k, n)$ is at most $2(n - k)$.*

Proof. We call a node $\mathbf{v}$ in a connected component of the reconfiguration graph *locally minimal (maximal)*, if there exists no $\mathbf{u}$ in the same connected component such that $hw(\mathbf{u}) \leq hw(\mathbf{v})$ ($hw(\mathbf{u}) \geq hw(\mathbf{v})$). When every relation in $\mathcal{S}$ is OR-free (NAND-free), it follows from Lemma 6.4.7 that there is exactly one locally minimal (maximal) assignment in each connected component of the reconfiguration graph. Moreover, there is a monotonically decreasing (increasing) path from every other node in the connected component to the locally minimal (maximal) assignment. Hence, the statement of the lemma follows. □

6.4.1 Classical complexity results

Lemma 6.4.9. *$\mathcal{S}$-MIN-ONES-REACH is in* `P` *if every relation in $\mathcal{S}$ is OR-free and it is* `PSPACE`*-complete otherwise.*

Proof. If some relation in $\mathcal{S}$ is not OR-free, then the problem is PSPACE-complete due to the PSPACE-completeness of the VC-Reach problem. That is, we can reduce VC-Reach to $\mathcal{S}$-MIN-ONES-REACH where ϕ is a 2CNF monotone formula.

Given a CNF($\mathcal{S}$) formula ϕ and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of weight at most k, if every relation in $\mathcal{S}$ is OR-free, then $\mathcal{S}$ is tight and we can use Gopalan et al.'s algorithm (Theorem 6.4.3) to find a reconfiguration sequence from $\mathbf{v_s}$ to $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$ (if one exists). To make sure that the maximum weight constraint is not violated, i.e. to show that $R_{\text{SAT}}(\phi, 0, k)$ also contains a reconfiguration sequence form $\mathbf{v_s}$ to $\mathbf{v_t}$, it is enough to note that since every relation is OR-free, we can convert any reconfiguration sequence into a negative canonical one (Lemma 6.4.7). □

Lemma 6.4.10. *$\mathcal{S}$-MAX-ONES-REACH is in* P *if every relation in $\mathcal{S}$ is NAND-free and it is* PSPACE*-complete otherwise.*

Proof. If there exists a relation in $\mathcal{S}$ that is not NAND-free, then the problem is PSPACE-complete due to the PSPACE-completeness of the IS-Reach problem. That is, we can reduce IS-Reach to $\mathcal{S}$-MAX-ONES-REACH where ϕ is a 2CNF anti-monotone formula.

If every relation in $\mathcal{S}$ is NAND-free, then $\mathcal{S}$ is tight and we can use Gopalan et al.'s (Theorem 6.4.3) algorithm to find a reconfiguration sequence (in $R_{\text{SAT}}(\phi, 0, n)$ if one exists). Since every relation is NAND-free, we can convert any reconfiguration sequence into a positive canonical sequence (Lemma 6.4.7) and hence the minimum weight constraint is not violated. □

Lemma 6.4.11. *$\mathcal{S}$-MIN-ONES-BOUND is in* P *if every relation in $\mathcal{S}$ is both OR-free and Horn-free, it is* NP*-complete if all relations in $\mathcal{S}$ are OR-free but some are not Horn-free, and it is* PSPACE*-complete otherwise.*

Proof. If some relation in $\mathcal{S}$ is not OR-free, then the problem is PSPACE-complete (VC-Bound). When all relations are OR-free but some are not Horn-free the problem becomes NP-hard due to Theorem 6.3.8. NP-completeness follows from the fact that the diameter of the reconfiguration graph is polynomial (Lemma 6.4.8).

If every relation in $\mathcal{S}$ is OR-free and Horn-free, then we can use the algorithm of Mouawad et al. (Theorem 6.4.4) to solve the problem. Because of OR-freeness, we can convert any reconfiguration sequence of length ℓ into a negative canonical sequence of length at most ℓ that does not violate the maximum weight constraint (Lemma 6.4.7). □

Lemma 6.4.12. *$\mathcal{S}$-MAX-ONES-BOUND is in* P *if every relation in $\mathcal{S}$ is both NAND-free and dual-Horn-free, it is* NP*-complete if all relations in $\mathcal{S}$ are NAND-free but some are not dual-Horn-free, and it is* PSPACE*-complete otherwise.*

Proof. If some relation in $\mathcal{S}$ is not NAND-free, then the problem is `PSPACE`-complete (IS-Bound). When all relations are NAND-free but some are not dual-Horn-free then the problem is `NP`-complete again due to Theorem 6.3.8 combined with the fact that the diameter of the reconfiguration graph is polynomial (Lemma 6.4.8).

If every relation in $\mathcal{S}$ is OR-free and Horn-free, then we can use the algorithm of Mouawad et al. (Theorem 6.4.4) to solve the problem. Because of NAND-freeness, we have a positive canonical sequence and the minimum weight constraint is never violated (Lemma 6.4.7). □

6.4.2 Parameterized complexity results

Lemma 6.4.13. *$\mathcal{S}$-MIN-ONES-REACH parameterized by k is in* `P` *(and hence* `FPT`*) if every relation in $\mathcal{S}$ is OR-free and it is* `FPT` *(but* `PSPACE`*-complete) otherwise.*

Proof. The fact that the problem is `FPT` follows from Theorem 5.1.2 combined with the fact that under Schaefer's framework all relations have constant arity. □

Lemma 6.4.14. *$\mathcal{S}$-MAX-ONES-REACH parameterized by k is in* `P` *(and hence* `FPT`*) if every relation in $\mathcal{S}$ is NAND-free and it is* `W[1]`*-hard otherwise.*

Proof. As IS-Reach parameterized by k is `W[1]`-hard (Theorem 4.2.2), it follows that $\mathcal{S}$-MAX-ONES-REACH parameterized by k is `W[1]`-hard whenever some relation in $\mathcal{S}$ is not NAND-free. □

Lemma 6.4.15. *$\mathcal{S}$-MIN-ONES-BOUND parameterized by ℓ is in* `P` *(and hence* `FPT`*) if every relation in $\mathcal{S}$ is both OR-free and Horn-free, it is* `FPT` *(but not in* `P`*) if every relation is OR-free but some are not Horn-free, and it is* `W[1]`*-hard otherwise.*

Proof. The `W[1]`-hardness of $\mathcal{S}$-MIN-ONES-BOUND parameterized by ℓ when some relation is not OR-free follows from Theorem 4.2.2, i.e. VC-Bound parameterized by ℓ is `W[1]`-hard.

Otherwise, given a CNF($\mathcal{S}$) formula ϕ and two satisfying assignments $\mathbf{v_s}$ and $\mathbf{v_t}$ of weight at most k, if every relation is OR-free but some relation is not Horn-free, then we can solve the problem using our fixed-parameter tractable algorithm for CSP-Bound (Theorem 6.3.6) and find a reconfiguration sequence from $\mathbf{v_s}$ to $\mathbf{v_t}$ in $R_{\text{SAT}}(\phi, 0, n)$ (if one exists). To make sure that the maximum weight constraint is not violated, i.e. to show that $R_{\text{SAT}}(\phi, 0, k)$ also contains a reconfiguration sequence from $\mathbf{v_s}$ to $\mathbf{v_t}$, it is enough to note that since every relation is OR-free, we can convert any reconfiguration sequence into a negative canonical one (Lemma 6.4.7). □

Lemma 6.4.16. *$\mathcal{S}$-MAX-ONES-BOUND parameterized by ℓ is in* `P` *if every relation in $\mathcal{S}$ is both NAND-free and dual-Horn-free, it is* `FPT` *if every relation is NAND-free but some are not dual-Horn-free, and it is* `W[1]`*-hard otherwise.*

Proof. The fact that $\mathcal{S}$-MAX-ONES-BOUND parameterized by ℓ is `W[1]`-hard when some relation is not NAND-free follows from Theorem 4.2.2, i.e. IS-BOUND parameterized by ℓ is `W[1]`-hard.

When every relation is NAND-free but some relation is not Horn-free, we can solve the problem using our fixed-parameter tractable algorithm for CSP-BOUND (Theorem 6.3.6) and then transform any reconfiguration sequence into a positive canonical one (Lemma 6.4.7). □

Chapter 7

Conclusion and directions for future work

Reconfiguration, as a framework, is a relatively new area and several interesting questions remain unanswered. We summarize what we believe to be some of the most interesting open problems below.

In Chapter 3, we showed that the DS-Reach problem is PSPACE-complete on planar, split, bipartite, bounded degree, and bounded treewidth graphs, and solvable in polynomial time on paths and trees. Referring back to Figure 2.2, we note that the number of remaining "familiar" graph classes where one can hope for polynomial-time algorithms is not too large. One particularly interesting class is that of series-parallel graphs, i.e. graphs of treewidth at most two. Since DS-Reach is PSPACE-complete on graphs of treewidth t, for some $t > 1$, and solvable in polynomial time when $t = 1$, can we find the exact value of t for which the classical complexity status of the problem changes from "easy" to "hard"? The problem is open for $t = 2$ and the same question holds for the VC-Reach (or IS-Reach) problem (Chapter 4). Similarly, it is not hard to see that DS-Reach is solvable in polynomial time whenever the input graph has a dominating set of size one (e.g. threshold graphs and cliques). Can we find an integer constant c such that DS-Reach is PSPACE-complete on graphs having a dominating set of size c? Or equivalently, can we find the boundary separating hard instances from easy instances by considering the other side of the density spectrum, i.e. the "dense side"? While studying structural properties of the reconfiguration graph (Section 3.2), we demonstrated infinite families of planar, bounded treewidth, and b-partite graphs, $b > 2$, for which $R_{DS}(G, 0, \Gamma(G) + 1)$ is not connected. Interestingly, all of our proofs break if we consider $R_{DS}(G, 0, \Gamma(G) + 2)$. Hence, it remains to be seen whether $R_{DS}(G, 0, \Gamma(G) + 2)$ is connected for all graphs. Finally, as the class of biclique-free graphs is the largest class on which the Dominating Set problem is known to be fixed-parameter tractable parameterized by solution size, establishing fixed-parameter tractability results beyond biclique-free graphs seems like a challenging task (Section 3.3.2). However, our work on reconfiguration problems has so far been "catching up" with known results in parameterized complexity. Hence, it would be very appealing if we can instead use insights from reconfiguration to find a "new" graph class on which both Dominating Set and its reconfiguration variants are fixed-parameter tractable.

Given the rather intriguing structure of nice reconfiguration sequences, we believe that they deserve a more careful study (Section 4.1). For instance, finding a biclique of size k in a graph is a notoriously hard problem whose parameterized complexity was only recently settled (after being open for more than two decades), the problem is W[1]-hard [99]. Can we, combining the fact that add-remove segments induce bicliques in a graph with the W[1]-hardness of VC-Bound parameterized by ℓ, find an alternative, maybe easier, proof of this result? Moreover, we believe that bicliques as induced subgraphs are replaced by bicliques as minors when we consider hereditary properties other than that of edgeless graphs. For example, if we consider Π as the collection of all forests, then, intuitively, every pair of added and removed vertices in an add-remove segment must belong to some cycle in the graph. Contracting some of these edges should again produce a biclique (as a minor). Can we relate the complexity of finding such structures to the complexity of the reconfiguration variants of a problem?

More generally, a fundamental problem is to try to establish a connection between the (parameterized) complexity of reconfiguration problems and the (parameterized) complexity of the underlying decision problem. We know that this relationship is more subtle then "P implies P", "NP-complete implies PSPACE-complete, "FPT implies FPT", and "W-hard implies W-hard". However, all of our results suggest a pattern relating the parameterized complexity of a problem and the parameterized complexity of its reachability variant (and the diameter of the reconfiguration graph): whenever a problem that is W-hard in general is fixed-parameter tractable parameterized by solution size on a graph class $\mathcal{C}$, then the reachability version of the problem (with the same parameter) is also fixed-parameter tractable on $\mathcal{C}$. We think that either proving or disproving this pattern would be quite interesting; IS-Reach on biclique-free graphs is the best contender we know of so far (Section 5.4).

The reconfiguration variants of the (s, t)-Cut problem, i.e. the security problem introduced in Chapter 1, have not been studied yet and we believe they might lead to some interesting results. First, are these problems in P, NP-complete, or PSPACE-complete? We believe they are PSPACE-complete, which would provide another example of a "natural" problem in P whose reconfiguration variants are PSPACE-complete. If this turns out to be the case, then are the problems fixed-parameter tractable? Note that the number of

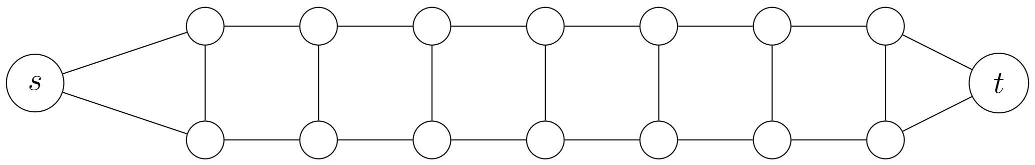

Figure 7.1: A graph illustrating why finding irrelevant vertices might be hard for cut problems.

minimal (s, t)-cuts of size at most k in a graph can be exponential in n, and therefore our "reconfiguration via enumeration" technique fails. Moreover, it is not known whether the problem admits a compact representation. Also, finding irrelevant vertices, at least using the same strategies we used in Sections 3.3.2 and 5.4, does not seem to apply. Consider a $2 \times n$ grid where s is connected to the first two vertices of the grid and t is connected to the last two (Figure 7.1). It is not hard to see that if we set our source solution to the neighbors of s and our target solution to the neighbors of t then any reconfiguration sequence between those two solutions must touch almost every vertex (a function of n, not k) in the graph. Several algorithmic techniques have been developed to deal with parameterized graph separation problems [103]. Can we adapt these techniques as well and add them to our toolkit for parameterized reconfiguration problems?

Appendix

Problem definitions in alphabetical order

Bounded Hitting Set (BHS)

Input: A finite universe $\mathcal{U}$, a constant c, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$ each of size at most c, and an integer k

Question: Is there $H \subseteq \mathcal{U}$ such that $|H| \leq k$ and for every set $F \in \mathcal{F}$ we have $F \cap U \neq \emptyset$?

BHS-Bound

Input: A finite universe $\mathcal{U}$, a constant c, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$ each of size at most c, integers k and ℓ, and two hitting sets H_s and H_t of $\mathcal{F}$ of size at most k

Question: Is there a path of length at most ℓ from H_s to H_t in $R_{\text{BHS}}(\mathcal{F}, 0, k)$?

BHS-Reach

Input: A finite universe $\mathcal{U}$, a constant c, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$ each of size at most c, an integer k, and two hitting sets H_s and H_t of $\mathcal{F}$ of size at most k

Question: Is there a path from H_s to H_t in $R_{\text{BHS}}(\mathcal{F}, 0, k)$?

Coloring (COL)

Input: A graph G with vertex set $V(G)$ and edge set $E(G)$ and an integer k.

Question: Does G admit a proper k-coloring?

Col-Bound
Input: A graph G, positive integers k and ℓ, and two k-colorings α and β of G.
Question: Is there a path of length at most ℓ from α to β in $R_{COL}(G, 0, k)$?

Col-Reach
Input: A graph G, positive integer k, and two k-colorings α and β of G.
Question: Is there a path from α to β in $R_{COL}(G, 0, k)$?

Dominating Set (DS)
Input: A graph G and a positive integer k
Question: Is there $D \subseteq V(G)$ such that $|D| \leq k$ and $N[D] = V(G)$?

DS-Bound
Input: A graph G, positive integers k and ℓ, and two dominating sets D_s and D_t of G of size at most k
Question: Is there a path of length at most ℓ from D_s to D_t in $R_{DS}(G, 0, k)$?

DS-Reach
Input: A graph G, positive integer k, and two dominating sets D_s and D_t of G of size at most k
Question: Is there a path from D_s to D_t in $R_{DS}(G, 0, k)$?

Feedback Vertex Set (FVS)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \leq k$ and $G[V(G) \setminus S]$ is a forest?

FVS-Bound
Input: A graph G, positive integers k and ℓ, and two feedback vertex sets S_s and S_t of G of size at least k
Question: Is there a path of length at most ℓ from S_s to S_t in $R_{\text{FVS}}(G, 0, k)$?

FVS-Reach
Input: A graph G, positive integer k, and two feedback vertex sets S_s and S_t of G of size at least k
Question: Is there a path from S_s to S_t in $R_{\text{FVS}}(G, 0, k)$?

Independent Set (IS)
Input: A graph G and a positive integer k
Question: Is there $I \subseteq V(G)$ such that $|I| \geq k$ and $G[I]$ is edgeless?

IS-Bound
Input: A graph G, positive integers k and ℓ, and two independent sets I_s and I_t of G of size at least k
Question: Is there a path of length at most ℓ from I_s to I_t in $R_{\text{IS}}(G, k, n)$?

IS-Reach
Input: A graph G, positive integer k, and two independent sets I_s and I_t of G of size at least k
Question: Is there a path from I_s to I_t in $R_{\text{IS}}(G, k, n)$?

Induced Bipartite Subgraph (IBS)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \geq k$ and $G[S]$ is bipartite?

IBS-Bound
Input: A graph G, positive integers k and ℓ, and two sets S_s and S_t of size at least k such that $G[S_s]$ and $G[S_t]$ are bipartite
Question: Is there a path of length at most ℓ from S_s to S_t in $R_{\text{IBS}}(G, k, n)$?

IBS-Reach
Input: A graph G, positive integer k, and two sets S_s and S_t of size at least k such that $G[S_s]$ and $G[S_t]$ are bipartite
Question: Is there a path from S_s to S_t in $R_{\text{IBS}}(G, k, n)$?

Induced Forest (IF)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \geq k$ and $G[S]$ is a forest?

IF-Bound
Input: A graph G, positive integers k and ℓ, and two sets S_s and S_t of size at least k such that $G[S_s]$ and $G[S_t]$ of G are forests
Question: Is there a path of length at most ℓ from S_s to S_t in $R_{\text{IF}}(G, k, n)$?

IF-Reach
Input: A graph G, positive integer k, and two sets S_s and S_t of size at least k such that $G[S_s]$ and $G[S_t]$ of G are forests
Question: Is there a path from S_s to S_t in $R_{\text{IF}}(G, k, n)$?

Multicolored Independent Set (MIS)
Input: A graph G, a positive integer ℓ, and a vertex-coloring $c : V(G) \to \{c_1, \ldots, c_\ell\}$ for G
Question: Does G have an independent set of size ℓ including exactly one vertex of each color?

Odd Cycle Transversal (OCT)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \leq k$ and $G[V(G) \setminus S]$ is bipartite?

OCT-Bound
Input: A graph G, positive integers k and ℓ, and two odd cycle transversals S_s and S_t of G of size at least k
Question: Is there a path of length at most ℓ from S_s to S_t in $R_{\text{OCT}}(G, 0, k)$?

OCT-Reach
Input: A graph G, positive integer k, and two odd cycle transversals S_s and S_t of G of size at least k
Question: Is there a path from S_s to S_t in $R_{\text{OCT}}(G, 0, k)$?

Red-Blue Dominating Set (RBDS)
Input: A bipartite graph G with bipartition (R, B) and a positive integer k
Question: Is there $D \subseteq R$ such that $|D| \leq k$ and $N[D] = B$?

RBDS-Bound
Input: A bipartite graph G with bipartition (R, B), positive integers k and ℓ, and two red-blue dominating sets D_s and D_t of G of size at most k
Question: Is there a path of length at most ℓ from D_s to D_t in $R_{RBDS}(G, 0, k)$?

RBDS-Reach
Input: A bipartite graph G with bipartition (R, B), positive integer k, and two red-blue dominating sets D_s and D_t of G of size at most k
Question: Is there a path from D_s to D_t in $R_{RBDS}(G, 0, k)$?

Unbounded Hitting Set (UHS)
Input: A finite universe $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$, and an integer k
Question: Is there $H \subseteq \mathcal{U}$ such that $|H| \leq k$ and for every set $F \in \mathcal{F}$ we have $F \cap U \neq \emptyset$?

UHS-Bound
Input: A finite universe $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$, integers k and ℓ, and two hitting sets H_s and H_t of $\mathcal{F}$ of size at most k
Question: Is there a path of length at most ℓ from H_s to H_t in $R_{\text{UHS}}(\mathcal{F}, 0, k)$?

UHS-Reach
Input: A finite universe $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$, an integer k, and two hitting sets H_s and H_t of $\mathcal{F}$ of size at most k
Question: Is there a path from H_s to H_t in $R_{\text{UHS}}(\mathcal{F}, 0, k)$?

Vertex Cover (VC)
Input: A graph G and a positive integer k
Question: Is there $S \subseteq V(G)$ such that $|S| \leq k$ and $G[V(G) \setminus S]$ is edgeless?

VC-Bound
Input: A graph G, positive integers k and ℓ, and two vertex covers S_s and S_t of G of size at most k
Question: Is there a path of length at most ℓ from S_s to S_t in $R_{VC}(G, 0, k)$?

Vertex Cover Local Search (VC-Local-Search)
Input: A graph G, a vertex cover S of G, and integer $\ell \geq 1$
Question: Does G have a vertex cover S' such that $|S'| < |S|$ and $|S' \Delta S| \leq \ell$?

VC-Reach
Input: A graph G, positive integer k, and two vertex covers S_s and S_t of G of size at most k
Question: Is there a path from S_s to S_t in $R_{VC}(G, 0, k)$?

Vertex Cover Walk (VC-Walk)
Input: A graph G, a vertex cover S of G, and an unlabeled edit sequence σ of length $\ell \geq 1$
Question: Can we apply σ to G and S?

References

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the* 6^{th} *Workshop on Algorithm Engineering and Experiments*, pages 62–69, 2004.

[2] D. Achlioptas, A. Coja-Oghlan, and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Structures and Algorithms*, 38(3):251–268, 2011.

[3] R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102, 2002.

[4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Theory, algorithms, and applications. Prentice Hall Inc., Upper Saddle River, NJ, USA, 1993.

[5] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings of the* 17^{th} *Scandinavian Workshop on Algorithm Theory*, pages 97–110. Springer Berlin Heidelberg, 2000.

[6] N. Alon and S. Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

[7] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[8] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer monographs in mathematics. Springer, 2008.

[9] V. Barbéra and B. Jaumard. Design of an efficient channel block retuning. *Mobile Networks and Applications*, 6(6):501–510, 2001.

[10] T. C. Biedl, E. D. Demaine, M. L. Demaine, R. Fleischer, L. Jacobsen, and J. I. Munro. The complexity of Clickomania. *Computing Research Repository*, cs.CC/0107, 2001.

[11] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998.

[12] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Computer Journal*, 51(3):255–269, 2008.

[13] M. Bonamy and N. Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013.

[14] M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma. On the diameter of reconfiguration graphs for vertex colourings. *Electronic Notes in Discrete Mathematics*, 38:161–166, 2011.

[15] 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.

[16] P. Bonsma. The complexity of rerouting shortest paths. In *Proceedings of the* 37^{th} *International Symposium on Mathematical Foundations of Computer Science*, pages 222–233, 2012.

[17] P. Bonsma. Rerouting shortest paths in planar graphs. In *Proceedings of the* 32^{nd} *Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 337–349, 2012.

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

[19] P. Bonsma, A. E. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proceedings of the* 9^{th} *International Symposium on Parameterized and Exact Computation*, pages 110–121, 2014.

[20] R. Borndrfer, A. Eisenbltter, M. Grtschel, and A. Martin. Frequency assignment in cellular phone networks. In *International Symposium on Mathematical Programming*, pages 24–29, 1997.

[21] P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.

[22] P. Bose, D. Jansens, A. van Renssen, M. Saumell, and S. Verdonschot. Making triangulations 4-connected using flips. *Computational Geometry*, 47(2):187–197, 2014.

[23] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[24] L. Cereceda. *Mixing graph colourings.* PhD thesis, London School of Economics, 2007.

[25] L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(56):913–919, 2008.

[26] L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European Journal of Combinatorics*, 30(7):1593–1606, 2009.

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

[28] B. Chor, M. R. Fellows, and D. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proceedings of the* 30th *International Conference on Graph-Theoretic Concepts in Computer Science*, pages 257–269, 2004.

[29] S. Cleary and K. S. John. Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 2009.

[30] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[31] N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of Boolean constraint satisfaction problems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[32] W. H. Cunningham. The optimal multiterminal cut problem. In *Center for Discrete Mathematics and Theoretical Computer Science Series*, 1991.

[33] P. Damaschke and L. Molokov. The union of minimal hitting sets: Parameterized combinatorial bounds and counting. *Journal of Discrete Algorithms*, 7(4):391–401, 2009.

[34] A. Dawar. Finite model theory on tame classes of structures. In *Proceedings of the* 32^{nd} *International Symposium on Mathematical Foundations of Computer Science*, volume 4708 of *Lecture Notes in Computer Science*, pages 2–12. Springer Berlin Heidelberg, 2007.

[35] A. Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010. Workshop on Logic, Language, Information and Computation.

[36] A. Dawar and S. Kreutzer. Domination problems in nowhere-dense classes of graphs. *CoRR*, 2009. arXiv:0907.42837.

[37] E. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: decomposition, approximation, and coloring. In *Proceedings of the* 46^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 637–646, 2005.

[38] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *Proceedings of the* 26^{th} *International Symposium on Mathematical Foundations of Computer Science*, volume 2136, pages 18–33, 2001.

[39] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

[40] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Computer Journal*, 51(3):292–302, 2008.

[41] R. Diestel. *Graph theory.* Springer-Verlag, Electronic Edition, 2005.

[42] R. G. Downey, V. Estivill-Castro, M. R. Fellows, E. Prieto, and F. A. Rosamond. Cutting up is hard to do: The parameterized complexity of k-cut and related problems. In *Electronic Notes in Theoretical Computer Science*, pages 205–218. Elsevier Science Publishers, 2003.

[43] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[44] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theoretical Computer Science*, 141(1–2):109–131, 1995.

[45] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness III: some structural aspects of the W-hierarchy. *Complexity theory*, pages 191–225, 1995.

[46] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1997.

[47] N. Dyn, I. Goren, and S. Rippa. Transforming triangulations in polygonal domains. *Computer Aided Geometry Design*, 10(6):531–536, 1993.

[48] J. Ellis, H. Fan, and M. Fellows. The dominating set problem is fixed parameter tractable for graphs of bounded genus. In *Proceedings of the* 19^{th} *Scandinavian Workshop on Algorithm Theory*, volume 2368 of *Lecture Notes in Computer Science*, pages 180–189. Springer Berlin Heidelberg, 2002.

[49] D. Eppstein. On the NP-completeness of cryptarithms. *SIGACT News*, 18(3):38–40, 1987.

[50] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[51] P. Erdos and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35:85–90, 1960.

[52] C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of Brooks theorem. In *Proceedings of the* 39^{th} *International Symposium on Mathematical Foundations of Computer Science*, volume 8635 of *Lecture Notes in Computer Science*, pages 287–298. Springer Berlin Heidelberg, 2014.

[53] U. Feige and K. Talwar. Approximating the bandwidth of caterpillars. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 3624 of *Lecture Notes in Computer Science*, pages 62–73. Springer Berlin Heidelberg, 2005.

[54] M. R. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Combinatorial Algorithms*, volume 5874 of *Lecture Notes in Computer Science*, pages 2–10. Springer Berlin Heidelberg, 2009.

[55] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. Rosamond, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012.

[56] M. R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[57] M. R. Fellows, F. A. Rosamond, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. Local search: is brute-force avoidable? In *Proceedings of the* 21^{st} *International Joint Conference on Artifical Intelligence*, pages 486–491, 2009.

[58] H. Fernau, T. Hagerup, N. Nishimura, P. Ragde, and K. Reinhardt. On the parameterized complexity of the generalized rush hour puzzle. In *Proceedings of the* 15^{th} *Canadian Conference on Computational Geometry*, pages 6–9, 2003.

[59] S. Fiorini, N. Hardy, B. Reed, and A. Vetta. Planar graph bipartization in linear time. *Discrete Applied Mathematics*, 156(7):1175–1180, 2008.

[60] G. W. Flake and E. B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1–2):895–911, 2002.

[61] J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM Journal on Computing*, 31(1):113–145, 2002.

[62] J. Flum and M. Grohe. *Parameterized complexity theory.* Springer-Verlag, Berlin, 2006.

[63] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990.

[64] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[65] S. Gaspers, E. J. Kim, S. Ordyniak, S. Saurabh, and S. Szeider. Don't be strict in local search! *CoRR*, 2012. arXiv:1208.1688.

[66] S. Gharibian and J. Sikora. Ground state connectivity of local Hamiltonians. *CoRR*, 2014. arXiv:1409.3182.

[67] 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.

[68] M. Grohe. Logic, graphs, and algorithms. *Electronic Colloquium on Computational Complexity*, 14(091), 2007.

[69] M. Grohe, S. Kreutzer, and S. Siebertz. Characterisations of nowhere dense graphs. In *Proceedings of the* 33^{rd} *Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 21–40, 2013.

[70] M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the* 46^{th} *Annual ACM Symposium on Theory of Computing*, pages 89–98, 2014.

[71] J. Guo, J. Gramm, F. Hffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

[72] G. Gutin and A. P. Punnen, editors. *The traveling salesman problem and its variations*. Combinatorial optimization. Kluwer Academic, Dordrecht, London, 2002.

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

[74] P. Hall. On representatives of subsets. In *Classic Papers in Combinatorics*, pages 58–62. Birkhuser Boston, 1987.

[75] J. Han. Frequency reassignment problem in mobile communication networks. *Computers and Operations Research*, 34(10):2939–2948, 2007.

[76] F. Harary. *Graph theory*. Addison-Wesley, Reading, MA, USA, 1991.

[77] 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.

[78] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete and Computational Geometry*, 22(3):333–346, 1999.

[79] R. Impagliazzo and R. Paturi. The complexity of k-SAT. In *Proceedings of the* 14^{th} *Annual IEEE Conference on Computational Complexity*, pages 237–240, 1999.

[80] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

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

[82] T. Ito, M. Kamiński, and E. D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

[83] T. Ito, M. Kamiski, and H. Ono. Fixed-parameter tractability of token jumping on planar graphs. In *Algorithms and Computation*, Lecture Notes in Computer Science, pages 208–219. Springer International Publishing, 2014.

[84] T. Ito, M. Kamiski, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka. On the parameterized complexity for token jumping on graphs. In *Proceedings of the* 11^{th} *Annual Conference on Theory and Applications of Models of Computation*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer International Publishing, 2014.

[85] T. Ito, K. Kawamura, H. Ono, and X. Zhou. Reconfiguration of list L(2, 1)-labelings in a graph. In *Proceedings of the* 23^{rd} *Annual International Symposium on Algorithms and Computation*, pages 34–43, 2012.

[86] R. Jacob. *Standortplanung mit Blick auf Online-Strategien.* PhD thesis, Universität Würzburg, 1997.

[87] M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *CoRR*, 2014. arXiv:1403.6347.

[88] W. W. Johnson and W. E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879.

[89] C. C. Kai, L. K. Fai, and L. Chu-Sing. *Rapid Prototyping: Principles and Applications in Manufacturing.* World Scientific Publishing Co., Inc., 2003.

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

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

[92] G. Kendall, A. J. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, pages 13–34, 2008.

[93] S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

[94] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[95] D. E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[96] C. Lawson. Software for c_1 surface interpolation. *Mathematical Software III*, pages 161–194, 1977.

[97] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[98] D. R. Lick and A. T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22:1082–1096, 1970.

[99] B. Lin. The parameterized complexity of k-biclique. *CoRR*, 2014. arXiv:1406.3700.

[100] A. Lubiw and V. Pathak. Flip distance between two triangulations of a point-set is NP-complete. *CoRR*, 2012. arXiv:1205.2425.

[101] M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.

[102] O. Marcotte and P. Hansen. The height and length of colour switching. In *Graph Colouring and Applications*, pages 101–110. American Mathematical Society, Providence, 1999.

[103] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

[104] K. Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:95–115, 1927.

[105] S. Mishra, V. Raman, S. Saurabh, S. Sikdar, and C. Subramanian. The complexity of Konig subgraph problems andabove-guarantee vertex cover. *Algorithmica*, 61(4):857–881, 2011.

[106] N. Misra, N. Narayanaswamy, V. Raman, and B. S. Shankar. Solving min ones 2-SAT as fast as vertex cover. In *Proceedings of the* 35th *International Symposium on Mathematical Foundations of Computer Science*, volume 6281, pages 549–555, 2010.

[107] B. Mohar. Kempe equivalence of colorings. In *Graph Theory in Paris*, Trends in Mathematics, pages 287–297. Birkhauser Basel, 2007.

[108] A. E. Mouawad, N. Nishimura, V. Pathak, and V. Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. *CoRR*, 2014. arXiv:1404.3801.

[109] A. E. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the parameterized complexity of reconfiguration problems. In *Proceedings of the* 8th *International Symposium on Parameterized and Exact Computation*, 2013.

[110] A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. *CoRR*, 2014. arXiv:1405.2447.

[111] R. A. Murphey, P. M. Pardalos, and M. G. Resende. Frequency assignment problems. In *Handbook of Combinatorial Optimization*, pages 295–377. Kluwer Academic Publishers, 1999.

[112] J. Nesetril and P. O. de Mendez. Structural properties of sparse graphs. In *Building Bridges*, volume 19 of *Bolyai Society Mathematical Studies*, pages 369–426. Springer Berlin Heidelberg, 2008.

[113] J. Nesetril and P. O. de Mendez. First order properties on nowhere dense structures. *Journal of Symbolic Logic*, 75(3):868–887, 2010.

[114] J. Nesetril and P. O. de Mendez. From sparse graphs to nowhere dense structures: Decompositions, independence, dualities and limits, 2010. European Congress of Mathematics.

[115] R. Niedermeier. *Invitation to fixed-parameter algorithms.* Oxford University Press, Oxford, 2006.

[116] R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the* 27th *International Symposium on Theoretical Aspects of Computer Science*, volume 5, pages 17–32, 2010.

[117] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, USA, 1994.

[118] V. Pathak. *Reconfiguring Triangulations*. PhD thesis, University of Waterloo, 2014.

[119] G. Philip, V. Raman, and S. Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Proceedings of the* 17th *Annual European Symposium on Algorithms*, volume 5757 of *Lecture Notes in Computer Science*, pages 694–705. Springer Berlin Heidelberg, 2009.

[120] S. Plotkin, S. Rao, and W. D. Smith. Shallow excluded minors and improved graph decompositions. In *Proceedings of the* 5th *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 462–470, 1994.

[121] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[122] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, Nov. 2004.

[123] S. Saurabh. Private communications, 2014.

[124] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[125] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the* 10th *Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[126] K. W. Schwerdtfeger. A computational trichotomy for connectivity of Boolean satisfiability. *CoRR*, 2013. arXiv:1312.4524.

[127] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[128] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of the* 18th *Annual ACM Symposium on Theory of Computing*, pages 122–135, 1986.

[129] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance. *Open problems in communication and computation*, 1987.

[130] J. A. Telle and Y. Villanger. FPT algorithms for domination in biclique-free graphs. In *Proceedings of the* 20th *Annual European Conference on Algorithms*, pages 802–812, 2012.

[131] J. van den Heuvel. The complexity of change. *Surveys in Combinatorics 2013*, 409:127–160, 2013.

[132] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

[133] M. Wrochna. Homomorphism reconfiguration via homotopy. *CoRR*, 2014. arXiv:1408.2812.

[134] M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. *CoRR*, 2014. arXiv:1405.0847.