HAL
open science

Reducing the vertex cover number via edge contractions

Paloma Thomé de Lima, Vinicius F. dos Santos, Ignasi Sau, Uéverton dos Santos Souza, Prafullkumar Tale

To cite this version:

Paloma Thomé de Lima, Vinicius F. dos Santos, Ignasi Sau, Uéverton dos Santos Souza, Prafullkumar Tale. Reducing the vertex cover number via edge contractions. Journal of Computer and System Sciences, 2023, 136, pp.63-87. 10.1016/j.jcss.2023.03.003 . lirmm-04107607

HAL Id: lirmm-04107607

https://hal-lirmm.ccsd.cnrs.fr/lirmm-04107607v1

Submitted on 26 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reducing the Vertex Cover Number via Edge Contractions

Paloma T. Lima
Computer Science Department, IT University of Copenhagen, Denmark

Vinicius F. dos Santos
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Uéverton S. Souza
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil
Institute of Informatics, University of Warsaw, Warsaw, Poland

Prafullkumar Tale
CISPA Helmholtz Center for Information Security, SIC, Saarbrücken, Germany

Abstract

Given a graph G on n vertices and two integers k and d, the Contraction(vc) problem asks whether one can contract at most k edges to reduce the vertex cover number of G by at least d. Recently, Lima et al. [JCSS 2021] proved that Contraction(vc) admits an XP algorithm running in time $f(d) \cdot n^{\mathcal{O}(d)}$. They asked whether this problem is FPT under this parameterization. In this article, we prove that: (i) Contraction(vc) is W[1]-hard parameterized by $k + d$. Moreover, unless the ETH fails, the problem does not admit an algorithm running in time $f(k + d) \cdot n^{o(k+d)}$ for any function f. This answers negatively the open question stated in Lima et al. [JCSS 2021]. (ii) Contraction(vc) is NP-hard even when $k = d$. (iii) Contraction(vc) can be solved in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$. This improves the algorithm of Lima et al. [JCSS 2021], and shows that when $k = d$, Contraction(vc) is FPT parameterized by d (or by k).

2012 ACM Subject Classification Theory of computation → Param. complex. and exact algorithms

Keywords and phrases Blocker problems, edge contraction, vertex cover, parameterized complexity.

Related Version This article is permanently available at `https://arxiv.org/abs/2202.03322`. An extended abstract of this paper will appear in the *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS), LIPIcs, Vienna, Austria, August 2022*.

Funding *Vinicius F. dos Santos*: Grant APQ-01707-21 Minas Gerais Research Support Foundation (FAPEMIG) and Grant 312069/2021-9 National Council for Scientific and Technological Development (CNPq).
Ignasi Sau: CAPES-PRINT Institutional Internationalization Program, process 88887.371209/2019-00, and projects DEMOGRAPH (ANR-16-CE40-0028), ESIGMA (ANR-17-CE23-0010), ELIT (ANR-20-CE48-0008-01), and UTMA (ANR-20-CE92-0027).
Uéverton S. Souza: This research has received funding from Rio de Janeiro Research Support Foundation (FAPERJ) under grant agreement E-26/201.344/2021, National Council for Scientific and Technological Development (CNPq) under grant agreement 309832/2020-9, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement CUTACOMBS (No. 714704).
Prafullkumar Tale: This research is a part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement SYSTEMATICGRAPH (No. 725978).

Acknowledgements The last author would like to thank Roohani Sharma for insightful discussions.

1 Introduction

Graph modification problems have been extensively studied in theoretical computer science, in particular for their vast expressive power and their applicability in a number of scenarios. Such problems can be generically defined as follows. For a fixed graph class $\mathcal{F}$ and a fixed set $\mathcal{M}$ of allowed graph modification operations, such as vertex deletion, edge deletion, edge addition, edge editing or edge contraction, the $\mathcal{F}$-$\mathcal{M}$-Modification problem takes as input a graph G and a positive integer k, and the goal is to decide whether at most k operations from $\mathcal{M}$ can be applied to G so that the resulting graph belongs to the class $\mathcal{F}$. For most natural graph classes $\mathcal{F}$ and modification operations $\mathcal{M}$, the $\mathcal{F}$-$\mathcal{M}$-Modification problem is NP-hard [33, 42, 43]. To cope up with this hardness, these problems have been examined via the lens of parameterized complexity [8, 13]. With an appropriate choice of $\mathcal{F}$ and the allowed modification operations $\mathcal{M}$, $\mathcal{F}$-$\mathcal{M}$-Modification can encapsulate well-studied problems like Vertex Cover, Feedback Vertex Set (FVS), Odd Cycle Transversal (OCT), Chordal Completion, or Cluster Editing, to name just a few.

The most natural and well-studied modification operations are, probably in this order, vertex deletion, edge deletion, and edge addition. In recent years, the *edge contraction* operation has begun to attract significant scientific attention. (When contracting an edge uv in a graph G, we delete u and v from G, add a new vertex and make it adjacent to vertices that were adjacent to u or v.) In parameterized complexity, $\mathcal{F}$-Contraction problems, i.e., $\mathcal{F}$-$\mathcal{M}$-Modification problems where the only modification operation in $\mathcal{M}$ is edge contraction, are usually studied with the number of edges allowed to contract, k, as the parameter. A series of recent papers studied the parameterized complexity of $\mathcal{F}$-Contraction for various graph classes $\mathcal{F}$ such as paths and trees [26], generalizations and restrictions of trees [1, 2], cactus graphs [32], bipartite graphs [24, 27], planar graphs [23], grids [40], cliques [9], split graphs [3], chordal graphs [35], bicliques [37], or degree-constrained graph classes [6, 22, 41].

For all the $\mathcal{F}$-$\mathcal{M}$-Modification problems mentioned above, a typical definition of the problem contains a description of the target graph class $\mathcal{F}$. For example, Vertex Cover, FVS, and OCT are $\mathcal{F}$-$\mathcal{M}$-Modification problems where $\mathcal{F}$ is the collection of edgeless graphs, forests, and bipartite graphs, respectively, and $\mathcal{M}$ contains only vertex deletion. Recently, a different formulation of these graph modification problems, called *blocker problems*, has been considered. In this formulation, the target graph class is defined in a *parametric way* from the input graph. To make the statement of such problems precise, consider an invariant $\pi : \mathcal{G} \mapsto \mathbb{N}$, where $\mathcal{G}$ is the collection of all graphs. For a fixed invariant π, a typical input of a blocker problem consists of a graph G, a budget k, and a threshold value d, and the question is whether G can be converted into a graph G' using at most k allowed modifications such that $\pi(G') \leq \pi(G) - d$. This is the same as determining whether (G, k, d) is a Yes-instance of $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification where $\mathcal{F}^{\pi}_{G,d} = \{G' \in \mathcal{G} \mid \pi(G') \leq \pi(G) - d\}$.

Consider the following examples of this formulation. For the invariant $\pi(G) = |E(G)|$, threshold $d = |E(G)|$, and vertex deletion as the modification operation in $\mathcal{M}$, $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification is the same as Vertex Cover. Setting the threshold d to a fixed integer p leads to Partial Vertex Cover. In a typical definition of this problem, the input is a graph G and two integers k, p, and the objective is to decide whether there is a set of vertices of size at most k that has at least p edges incident on it. Consider another example when $\pi(G) = \mathtt{vc}(G)$, the size of a minimum vertex cover of G, the threshold value $d = \mathtt{vc}(G) - 1$, and the allowed modification operation is edge contraction. To reduce the size of a minimum vertex cover from $\mathtt{vc}(G)$ to 1 by k edge contractions, we need to find a connected vertex

cover of size $k+1$. Hence, in this case $\mathcal{F}^{\pi}_{G,d}$-$\mathcal{M}$-Modification is the same as Connected Vertex Cover. In all these cases, we can think of the set of vertices or edges involved in the modifications as 'blocking' the invariant π, that is, preventing π from being smaller.

Blocker problems have been investigated for numerous graph invariants, such as the chromatic number, the independence number, the matching number, the diameter, the domination number, the total domination number, and the clique number of a graph [4, 5, 7, 11, 16, 30, 36, 39, 42] with 'vertex deletion' or 'edge deletion' as the allowed graph modification operation. Blocker problems with the edge contraction operation have already been studied with respect to the chromatic number, clique number, independence number [16, 39], the domination number [19, 21], total domination number [20], and the semitotal domination number [18].

This article is strongly motivated by the results in [34]. They proved, among other results, that it is coNP-hard to test whether we can reduce the size of a minimum feedback vertex set or of a minimum odd cycle transversal of a graph by one, i.e., $d = 1$, by performing one edge contraction, i.e., $k = 1$. This is consistent with earlier results, as blocker problems are generally very hard, and become polynomial-time solvable only when restricted to specific graph classes. However, the notable exception is the case when the invariant in question is the size of a minimum vertex cover of the input graph. We formally define the problem before mentioning existing results and our contribution (where G/F denotes the graph obtained from G by contracting the edges in F).

Contraction(vc) **Input:** An undirected graph G and two non-negative integers k and d. **Question:** Does there exist a set $F \subseteq E(G)$ such that $\lvert F\rvert \leq k$ and $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$?

Our results. A simple reduction, briefly mentioned in [34], shows that the above problem is NP-hard for *some* k in $\{d, d+1, \ldots, 2d\}$. In our first result, we enhance our understanding of the classical complexity of the problem and prove that the problem is NP-hard even when $k = d$. As any edge contraction can decrease $\mathtt{vc}(G)$ by at most one, if $k < d$ then the input instance is a trivial No-instance. To state our first result, we introduce the notation of $\mathsf{rank}(G)$, which is the number of vertices of G minus its number of connected components (or equivalently, the number of edges of a set of spanning trees of each of the connected components of G). Note that it is sufficient to consider values of k that are at most $\mathsf{rank}(G)$, as otherwise it is possible to transform G to an edgeless graph with at most k contractions, and therefore in this case G is a yes-instance for Contraction(vc) if and only if $\mathtt{vc}(G) \geq d$.

▶ **Theorem 1.** *To decide whether an instance (G, k, d) of* Contraction(vc) *is a* Yes-*instance is*

- coNP-hard *if $k = \mathsf{rank}(G)$,*
- coNP-hard *if $k < \mathsf{rank}(G)$ and $2d \leq k$, and*
- NP-hard *if $k < \mathsf{rank}(G)$ and $k = d + \frac{\ell-1}{\ell+3} \cdot d$ for any integer $\ell \geq 1$ such that k is an integer.*

As one needs to contract at least d edges to reduce the size of a minimum vertex cover by d, the above theorem, for $\ell = 1$, implies that the problem is para-NP-hard when parameterized by the 'excess over the lower bound', i.e., by $k - d$. Since we can assume that $d \leq k$, d is a 'stronger' parameter than k. One of the main results of [34] is an XP algorithm for Contraction(vc) with running time $f(d) \cdot n^{\mathcal{O}(d)}$. Here, and in the rest of the article, we denote by n the number of vertices of the input graph. The authors explicitly asked whether the problem admits an FPT algorithm parameterized by d. As our next result, we answer

this question in the negative by proving that such an algorithm is highly unlikely, even when parameterized by the larger parameter $d + k$ (or equivalently, just k, as discussed above).

▶ **Theorem 2.** Contraction(vc) *is* W[1]-hard *parameterized by* $k + d$. *Moreover, unless the* ETH *fails*[1], *it does not admit an algorithm running in time* $f(k + d) \cdot n^{o(k+d)}$ *for any computable function* $f : \mathbb{N} \mapsto \mathbb{N}$. *The result holds even if we assume that the input* (G, k, d) *is such that* $k < \mathsf{rank}(G)$ *and* $d \leq k < 2d$, *and* G *is a bipartite graph with a bipartition* $\langle X, Y \rangle$ *such that* X *is a minimum vertex cover of* G.

For the XP algorithm in [34], the authors did not explicitly mention an upper bound on the corresponding function f, but it is expected to be quite large since the algorithm uses Courcelle's theorem [12] as a subroutine. Our next result improves this algorithm by providing a concrete upper bound on the running time, and by distinguishing in a precise way the contribution of k and d.

▶ **Theorem 3.** *There exists an algorithm that solves* Contraction(vc) *in time* $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$. *Moreover, for an input* (G, k, d), *the algorithm runs in time* $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$ *unless* $k < \mathsf{rank}(G)$ *and* $d \leq k < 2d$.

Note that the above result implies, in particular, that the problem is FPT parameterized by d when $k - d$ is a constant.

Our methods. A central tool in both our negative and positive results is Lemma 13, which allows us to reformulate the problem as follows. As discussed later, by applying appropriate FPT reductions to the input graph G, it is possible to assume that we have at hand a minimum vertex cover X of G. We say that a set of edges F is a *solution* of (G, k, d) if $|F| \leq k$ and $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$. Lemma 13 implies that there exists such a solution (i.e., an edge set) if and only if there exist vertex subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that the pair $\langle X_s, Y_s \rangle$, which we call a *solution pair*, satisfies the technical conditions mentioned in its statement (and which we prefer to omit here). This reformulation allows us to convert the problem of finding a subset F of edges to the problem of modifying the given minimum vertex cover X to obtain another vertex cover $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ such that $|X_{\mathsf{el}}| \leq |X| + (k - d)$ and $\mathsf{rank}(X_{\mathsf{el}}) \geq k$. Here, we define $\mathsf{rank}(X_{\mathsf{el}}) := \mathsf{rank}(G[X_{\mathsf{el}}])$. See Figure 2 for an illustration.

In our hardness reductions, another simple, yet critical, tool is Lemma 14, which states that if there is a vertex which is adjacent to a pendant vertex, then there is a solution pair that does not contain this vertex. We present overviews of the reductions in Section 3 and Section 4 to demonstrate the usefulness of these two lemmas in the respective hardness results. The reduction that we use to prove the third item in the statement of Theorem 1 (which is the most interesting case) is from a variant of Multicolored Independent Set, while the one in the proof of Theorem 2 is from Edge Induced Forest, a problem that we define and that we prove to be W[1]-hard in Theorem 19, by a parameter preserving reduction from, again, Multicolored Independent Set. It is worth mentioning that the W[1]-hardness in Theorem 19 holds even if we assume that the input graph G is a bipartite graph with a bipartition $\langle X, Y \rangle$ such that X is a minimum vertex cover of G, and such that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$. This case is the crux of the difficulty of the problem. This becomes clear in the XP algorithm of Theorem 3 that we proceed to discuss.

[1] The Exponential Time Hypothesis (ETH) is a conjecture stating that, roughly speaking, N-variable 3-SAT cannot be solved in time $2^{o(N)}$. We refer the reader to [14, Chapter 14] for the formal definition and related literature.

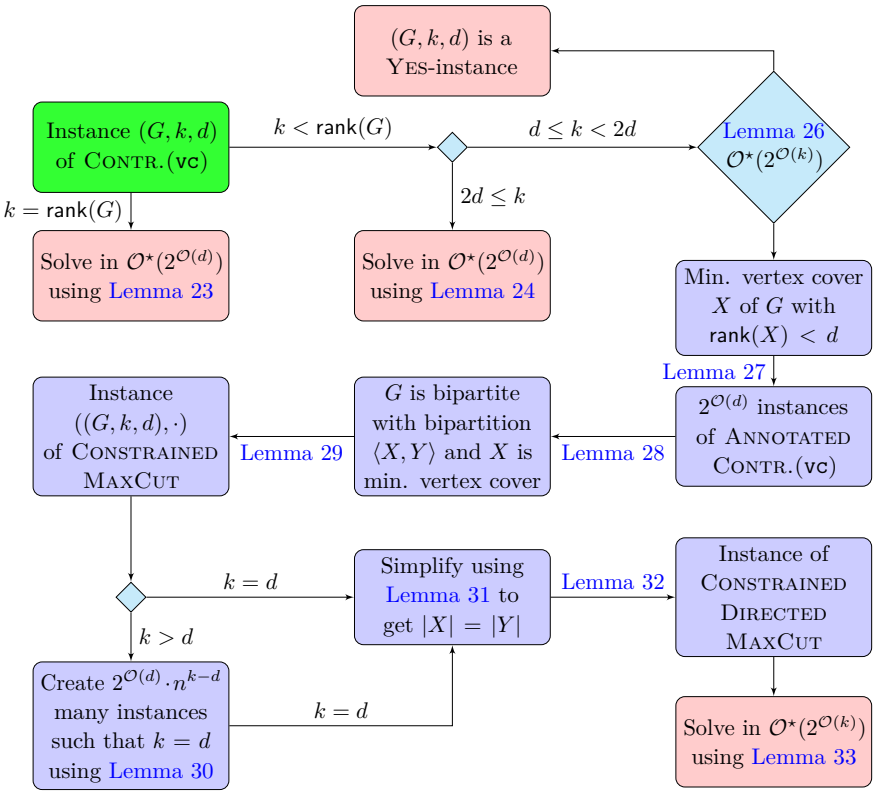

Figure 1 Diagram of the algorithm for CONTRACTION(vc) given by Theorem 3. Recall that we can assume that $d \leq k \leq \mathsf{rank}(G)$, hence the case distinction considered in the beginning is exhaustive. Note also that, in the case where $d \leq k < 2d$, it holds that $\mathcal{O}^\star(2^{\mathcal{O}(k)}) = \mathcal{O}^\star(2^{\mathcal{O}(d)})$.

The algorithm for CONTRACTION(vc), which is our main technical contribution, is provided in Section 5. A diagram of this algorithm is shown in Figure 1. By a standard Knapsack-type dynamic programming table, which is also mentioned in [34], we can assume that the input graph G is connected. We distinguish three cases depending on the relation between k, d, and $\mathsf{rank}(G)$. The first two cases are easy, and can be solved in time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$, by essentially running an FPT algorithm to determine whether $\mathsf{vc}(G) < d$; see Lemma 23 and Lemma 24. We now present an overview of the algorithm for the third case, namely when its input (G,k,d) is with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$ (cf. Lemma 25). Inspired by Lemma 13, we introduce an annotated version of the problem called ANNOTATED CONTRACTION(vc). We first argue (cf. Lemma 26) that there is an algorithm that runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, and either correctly concludes that (G,k,d) is a YES-instance of CONTRACTION(vc) or finds a minimum vertex cover X of G such that $\mathsf{rank}(X) < d$. Using this vertex cover, we can construct $2^{\mathcal{O}(d)}$ many instances of ANNOTATED CONTRACTION(vc) such that (G,k,d) is a YES-instance of CONTRACTION(vc) if and only if at least one of these newly created instances is a YES-instance of ANNOTATED CONTRACTION(vc) (cf. Lemma 27). Hence, it suffices to design an algorithm to solve ANNOTATED CONTRACTION(vc). We show that we can apply a simple reduction rule (cf. Lemma 28) that allows us to assume that the input graph G of ANNOTATED CONTRACTION(vc) is bipartite with bipartition $\langle X,Y\rangle$ such that X is a minimum vertex cover of G, as mentioned above.

A solution of an instance of ANNOTATED CONTRACTION(vc) is a solution pair $\langle X_s, Y_s\rangle$

as stated in Lemma 13. We find convenient to present an algorithm that finds a partition $\langle V_L, V_R\rangle$ of $V(G)$ instead of a solution pair $\langle X_s, Y_s\rangle$. To formalize this, we introduce the problem called Constrained MaxCut and we show it to be equivalent to Annotated Contraction(vc) (cf. Lemma 29). We partition the input instances of Constrained MaxCut into the following two types: (1) $k = d$, and (2) $k > d$. For the instances of the second type, we construct $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances of the first type such that the input instance is a Yes-instance if and only if at least one of these newly created instances is a Yes-instance (cf. Lemma 30). We remark that this is the only step in the whole algorithm where an n^{k-d}-factor appears (note that this is unavoidable by Theorem 19).

Finally, to handle the instances of the first type (i.e., with $k = d$), we apply a simplification based on the existence of a matching saturating X (cf. Lemma 31), we introduce a directed variation of the problem called Constrained Directed MaxCut, and we prove it to be equivalent to its undirected version (cf. Lemma 32). We then present a dynamic programming algorithm, with running time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, that critically uses the fact that $k = d$ (cf. Lemma 33), in particular to "merge" appropriately some directed cycles in order to obtain a directed *acyclic* graph, whose topological ordering gives a natural way to process the vertices of the input graph in a dynamic programming fashion. At the end of Subsubsection 5.3.4 we present an overview of this algorithm.

Organization. In Section 2 we present some notations, known results, preliminary results about Contraction(vc), and Lemma 13 and Lemma 14. In Section 3 we present a reduction from a special case of Multicolored Independent Set to Contraction(vc). This, along with other preliminary results, proves Theorem 1. In Section 4 we present two parameter preserving reductions, one from Multicolored Independent Set to Edge Induced Forest, and another one from Edge Induced Forest to Contraction(vc). These two reductions, along with known results about Multicolored Independent Set, prove Theorem 2. Section 5 is the most technical part of the paper, and contains the description of the algorithm to solve Contraction(vc) mentioned in Theorem 3. We conclude the article in Section 6 with some open problems.

2 Preliminaries

For a positive integer q, we denote the set $\{1, 2, \ldots, q\}$ by $[q]$. We use $\mathbb{N}$ to denote the collection of all non-negative integers.

2.1 Graph theory

We use standard graph-theoretic notation, and we refer the reader to [15] for any undefined notation. For an undirected graph G, sets $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively. For a directed graph D, sets $V(D)$ and $A(D)$ denote its set of vertices and arcs, respectively. We denote an edge with two endpoints u, v as uv. To avoid confusion with edges, we denote an arc with tail u and head v as (u, v). Unless otherwise specified, we use n to denote the number of vertices in the input (di)graph G of the problem under consideration. Two vertices u, v in $V(G)$ are *adjacent* if there is an edge uv in G. The *open neighborhood* of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v. The *closed neighborhood* of a vertex v, denoted by $N_G[v]$, is the set $N_G(v) \cup \{v\}$. We say that a vertex u is a *pendant vertex* if $|N_G(v)| = 1$. The *in-neighbourhood* of v in a digraph D is the set of vertices u such that (u, v) is an arc in $A(D)$. We say that (u, v) is an *in-coming* arc of v. Similarly, the *out-neighbourhood* of v is the set of vertices u such that (v, u) is an arc in $A(D)$.

We say that (v, u) is an *out-going* arc of v. We denote the out-neighbors of v by $N_{\mathsf{out}}(v)$. We omit the subscript in the notation for neighborhood if the graph under consideration is clear.

For a subset S of $V(G)$, we define $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. For a subset F of edges, we denote by $V(F)$ the collection of endpoints of edges in F. For a subset S of $V(G)$ (resp. a subset F of $E(G)$), we denote the graph obtained by deleting S (resp. deleting F) from G by $G - S$ (resp. by $G - F$). We denote the subgraph of G induced on the set S by $G[S]$. For two subsets S_1, S_2 of $V(G)$, $E(S_1, S_2)$ denotes the set of edges with one endpoint in S_1 and another one in S_2. With a slight abuse of notation, we use $E(S_1)$ to denote the set $E(S_1, S_1)$. Similarly, we define these notations for digraphs. Namely, $A(S_1, S_2)$ denotes the set of arcs with tail in S_1 and head in S_2.

A graph is *connected* if there is a path between every pair of distinct vertices. A subset $S \subseteq V(G)$ is said to be a *connected set* if $G[S]$ is connected. A *spanning tree* of a connected graph is a connected acyclic subgraph that includes all the vertices of the graph. A *spanning forest* of a graph is a collection of spanning trees of its connected components. The *rank* of a graph G, denoted by $\mathsf{rank}(G)$, is the number of edges of a spanning forest of G with the maximum number of edges. The *rank* of a set $X \subseteq V(G)$ of vertices, denoted by $\mathsf{rank}(X)$, is the rank of $G[X]$. The *rank* of a set $F \subseteq V(G)$ of edges, denoted by $\mathsf{rank}(F)$, is the rank of $V(F)$. Note that an edge contraction decreases the rank of a graph G by exactly one.

A set of vertices Y is said to be an *independent set* if no two vertices in Y are adjacent. We use the following observation.

▶ **Observation 4.** *Consider two independent sets X, Y in a graph G such that there is no isolated vertex in $G[X \cup Y]$. If $\mathsf{rank}(E(X, Y)) \leq k$, then $|X|, |Y| \leq k$.*

A graph is *bipartite* if its vertex set can be partitioned into two independent sets. For a graph G, a set $X \subseteq V(G)$ is said to be a *vertex cover* if $V(G) \setminus X$ is an independent set. A set of vertices Y is said to be a *clique* if any two vertices in Y are adjacent. A set of edges M is called a *matching* if no two edges in M share an endpoint. We say that a matching M *saturates* a set $X \subseteq V(G)$ if $X \subseteq V(M)$.

A vertex cover X is a *minimum vertex cover* if for any other vertex cover Y of G, we have $|X| \leq |Y|$. We denote by $\mathtt{vc}(G)$ the size of a minimum vertex cover of a graph G. As a vertex cover needs to contain at least one vertex from each edge in a matching, $\mathtt{vc}(G)$ is at least the size of a maximum matching. Consider a minimum vertex cover X. For any $X' \subseteq X$, we have $|X'| \leq |N(X')|$ as otherwise $Y = (X \setminus X') \cup N(X')$ is another vertex cover of G and $|Y| < |X|$, contradicting the fact that X is a minimum vertex cover. As $|X'| \leq |N(X')|$ for every $X' \subseteq X$, Hall's theorem [25] in bipartite graphs implies that there exists a matching saturating a minimum vertex cover X in G. Such a matching can be found in polynomial time [28]. For a graph G, a set $X \subseteq V(G)$ is said to be an *odd cycle transversal* if $G - X$ is a bipartite graph. An odd cycle transversal X is a *minimum odd cycle transversal* if for any other odd cycle transversal Y of G, we have $|X| \leq |Y|$. We denote by $\mathtt{oct}(G)$ the size of a minimum odd cycle transversal of a graph G. We need the following algorithmic results regarding vertex covers and odd cycle transversals.

▶ **Proposition 5** ([10]). *There is an algorithm that takes as input a graph G and an integer ℓ, runs in time $1.2738^{\ell} \cdot n^{\mathcal{O}(1)}$, and correctly determines whether $\mathtt{vc}(G) \leq \ell$.*

▶ **Proposition 6** (Corollary 10 in [38]). *There is an algorithm that takes as input a graph G and an integer ℓ, runs in time $2.6181^{\ell} \cdot n^{\mathcal{O}(1)}$ and determines whether $\mathtt{oct}(G) \leq \ell$.*

▶ **Proposition 7** (Corollary 15 in [38]). *There is an algorithm that takes as input a graph G, runs in time $1.6181^{\mathtt{oct}(G)} \cdot n^{\mathcal{O}(1)}$ and computes a minimum vertex cover of G.*

The algorithm in Proposition 5 can be easily modified to compute $\mathtt{vc}(G)$ if $\mathtt{vc}(G) \leq \ell$. For a graph G, we denote by $\mathtt{bc}(G)$ the minimum number of edges in G that need to be contracted to make it a bipartite graph. Note that for a set $F \subseteq E(G)$, if one can obtain a bipartite graph by contracting all edges in F, then one can obtain a bipartite graph by deleting one endpoint of every edge in F. Hence, we have the following observation.

▶ **Observation 8.** *For a graph G, $\mathtt{oct}(G) \leq \mathtt{bc}(G)$.*

Consider a (not necessarily proper) 2-coloring $\psi : V(G) \mapsto \{1, 2\}$. Heggernes et al. [27] defined a notion of *cost* of a 2-coloring ψ of a graph as $\sum_{X \in M_\psi}(|X| - 1)$, where M_ψ is the set of monochromatic components of ψ. Let (V_1, V_2) be the partition of $V(G)$ such that every vertex in V_1 and V_2 has color 1 and 2, respectively. It is easy to see that cost of ψ is equal to $\mathsf{rank}(V_1) + \mathsf{rank}(V_2)$. We restate [27, Lemma 1] as the following observation.

▶ **Observation 9.** *For a graph G, $\mathtt{bc}(G) \leq \ell$ if and only if there exists a partition (V_L, V_R) of $V(G)$ such that $\mathsf{rank}(V_L) + \mathsf{rank}(V_R) \leq \ell$.*

2.2 Edge contraction

The *contraction* of an edge uv in a graph G deletes vertices u and v from G, and adds a new vertex which is adjacent to all vertices that were adjacent to either u or v. This process does not introduce self-loops or parallel edges. The resulting graph is denoted by G/e. For a graph G and edge $e = uv$, we formally define G/e in the following way: $V(G/e) = (V(G) \cup \{w\}) \setminus \{u, v\}$ and $E(G/e) = \{xy \mid x, y \in V(G) \setminus \{u, v\}, xy \in E(G)\} \cup \{wx \mid x \in N_G(u) \cup N_G(v)\}$. Here, w is a new vertex. An edge contraction reduces the number of vertices in a graph by exactly one. Several edges might disappear because of one edge contraction. For a subset of edges F in G, graph G/F denotes the graph obtained from G by contracting all the edges in F.

We now formally define a contraction of a graph G to another graph H.

▶ **Definition 10** (Graph contraction)**.** *A graph G is said to be* contractible *to a graph H if there is a function $\psi : V(G) \to V(H)$ such that the following properties hold.*

1. *For any vertex h in $V(H)$, the set $W(h) := \{v \in V(G) \mid \psi(v) = h\}$ is not empty and the graph $G[W(h)]$ is connected.*
2. *For any two vertices h, h' in $V(H)$, edge hh' is present in H if and only if $E(W(h), W(h'))$ is not empty.*

We say that graph G is *contractible to H via function ψ*. For a vertex h in H, the set $W(h)$, also denoted by $\psi^{-1}(h)$, is called a *witness set* associated with or corresponding to h. For a fixed ψ, we define the *H-witness structure* of G, denoted by $\mathcal{W}$, as the collection of all witness sets. Formally, $\mathcal{W} = \{W(h) \mid h \in V(H)\}$. Note that a witness structure $\mathcal{W}$ is a partition of the vertices in G. If a witness set contains more than one vertex, then we call it a *big* witness set, otherwise we call it a *small* witness set.

2.3 Contraction($\mathtt{vc}$)

In this subsection, we present a couple of observations regarding an instance (G, k, d) of the Contraction($\mathtt{vc}$) problem. Later, we present a lemma that helps us to characterize the problem as finding a vertex cover with special properties.

▶ **Observation 11.** *Consider an instance (G, k, d) of* Contraction($\mathtt{vc}$) *such that $k = \mathsf{rank}(G)$. Then, (G, k, d) is a* Yes*-instance if and only if $d \leq \mathtt{vc}(G)$.*

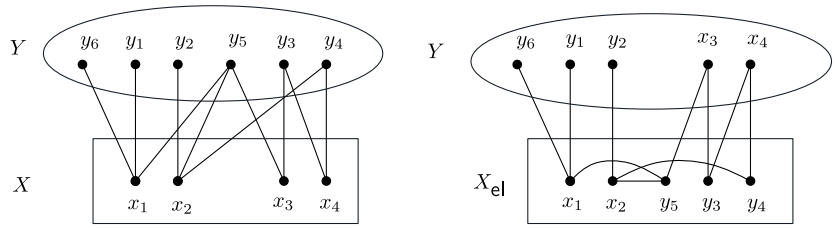

Figure 2 We can reduce the size of a minimum vertex cover of G by two by contracting the three edges in $F = \{x_1y_5, x_2y_5, x_2y_4\}$, i.e., $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$ for $d = 2$. Lemma 13 implies that there exists a solution pair $\langle X_s = \{x_3, x_4\}, Y_s = \{y_3, y_4, y_5\}\rangle$ such that $\mathsf{rank}((X \setminus X_s) \cup Y_s) = \mathsf{rank}(X_{\mathsf{el}}) \geq 3 = |F|$ and $|Y_s| - |X_s| \leq 1 = |F| - d$.

Proof. Suppose that (G, k, d) is a YES-instance. Let $F \subseteq E(G)$ be a collection of at most k edges in G such that $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. As $\mathtt{vc}(G/F) \geq 0$, we have $d \leq \mathtt{vc}(G)$.

Suppose now that $d \leq \mathtt{vc}(G)$. Let F be the collection of edges in a spanning forest of G. Note that the graph G/F does not contain any edge and hence $\mathtt{vc}(G/F) = 0 \leq \mathtt{vc}(G) - d$. As $k = \mathsf{rank}(G)$, we have $|F| = k$. Hence, F is a solution of (G, k, d). ◀

▶ **Observation 12.** *Consider an instance (G, k, d) of* CONTRACTION($\mathtt{vc}$) *such that G is a connected graph, $k < \mathsf{rank}(G)$, and $2d \leq k$. Then, (G, k, d) is a* YES*-instance if and only if $d < \mathtt{vc}(G)$.*

Proof. Suppose that (G, k, d) is a YES-instance. Let $F \subseteq E(G)$ be a collection of at most k edges in G such that $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. As $|F| \leq k$ and $k < \mathsf{rank}(G)$, the graph G/F contains at least one edge. Hence, $\mathtt{vc}(G/F) \geq 1$. This implies $1 + d \leq \mathtt{vc}(G)$.

Suppose now that $d < \mathtt{vc}(G)$. Let X be a minimum vertex cover of G. Consider an algorithm that contracts a path between two vertices in X that are distance at most two. The existence of such vertices is guaranteed by the fact that G is a connected graph. Let G' be the resulting graph. Note that $\mathtt{vc}(G') = \mathtt{vc}(G) - 1$. It is easy to verify that if (G, k, d) is a YES-instance, then $(G', k - 2, d - 1)$ is a YES-instance. As $d < \mathtt{vc}(G)$ and $k \geq 2d$, the subroutine can repeat the process d times to get an equivalent instance (G', k', d') such that $k' \geq 0$ and $d' = 0$. As (G', k', d') is a trivial YES-instance, (G, k, d) is a YES-instance. ◀

Suppose that (G, k, d) is a YES-instance of CONTRACTION($\mathtt{vc}$). We say that a set $F \subseteq E(G)$ is a *solution* of (G, k, d) if $|F| \leq k$ and $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. Fix a minimum vertex cover X of G. As X is a vertex cover, for every edge in F, *at least one* of its endpoints is in X. We argue that one can construct an *enlarged vertex cover* X_{el} of G such that for every edge in F, *both* of its endpoints are in X_{el}. Also, X_{el} is not much larger than X. In order to construct X_{el} from X, one needs to remove and add some vertices to X. We denote the removed and added vertices by X_s and Y_s, respectively, and call $\langle X_s, Y_s\rangle$ a *solution pair*. See Figure 2 for an illustration. The following lemma relates a solution (a set of edges) to a solution pair (a tuple of disjoint vertex sets).

▶ **Lemma 13.** *Consider a connected graph G, a minimum vertex cover X of G, and two non-negative integers k, d such that $k < \mathsf{rank}(G)$. For a proper subset F of edges of a spanning forest of G of size k, $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$ if and only if there exists subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $X_{\mathsf{el}} := (X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$, and (iii) $|Y_s| - |X_s| \leq k - d$, i.e., $|X_{\mathsf{el}}| \leq |X| + k - d$.*

Proof. ($\Rightarrow$) Consider the collection $\mathcal{F}$ of subsets of $E(G)$ of size k such that for every $F \in \mathcal{F}$, $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. For $F \in \mathcal{F}$, suppose G is contracted to G/F via function ψ. Let X' be a minimum vertex cover of G/F and $Y' = V(G/F) \setminus X'$. We say that $\langle X', Y' \rangle$ is a partition corresponding to F. We define a function $\mathsf{cost} : \mathcal{F} \mapsto \mathbb{N}$ as follows. For $F \in \mathcal{F}$, $\mathsf{cost}(F)$ is the minimum number of vertices in Y' that are associated with big witness sets over all partitions $\langle X', Y' \rangle$ corresponding to F. Formally, $\mathsf{cost}(F) := \min_{\langle X', Y' \rangle} |\{y \in Y' \mid |\psi^{-1}(y)| > 1\}|$, where $\langle X', Y' \rangle$ ranges over all partitions corresponding to F.

We assume, for the sake of contradiction, that there is no set in $\mathcal{F}$ whose cost is zero. Let $F \in \mathcal{F}$ be a set of edges of minimum cost over all sets in $\mathcal{F}$. By our assumption, $\mathsf{cost}(F) > 0$. This implies that there is a partition $\langle X', Y' \rangle$ of $V(G/F)$ and a vertex $y' \in Y'$ such that $|\psi^{-1}(y')| > 1$. Recall that F is a proper subset of edges in a spanning forest of G. Hence, $|F| < \mathsf{rank}(G)$ and there is at least one edge present in G/F. This implies that a minimum vertex cover X' of G/F is not empty. As G is a connected graph, so is G/F. Hence, there is a vertex $x' \in X'$ such that $x'y' \in E(G/F)$.

Consider a G/F-witness structure $\mathcal{W}$ of G. Let $W(x'), W(y')$ be the witness sets corresponding to x' and y', respectively. Recall that $W(x')$ and $W(y')$ are connected sets in G. As $x'y' \in E(G/F)$, there exists an edge e in $E(G)$ with one of its endpoints in $W(x')$ and the other in $W(y')$. Hence, $W(x') \cup W(y')$ is a connected set in G. We claim that there is a spanning tree of $G[W(x') \cup W(y')]$ that has a leaf in $W(y')$.

Let T_x and T_y be spanning trees of $G[W(x')]$ and $G[W(y')]$, respectively. Without loss of generality, we can assume that $E(T_x) \cup E(T_y) \subseteq F$. As $|W(y')| > 1$, T_y contains at least one edge and hence at least two leaves. Consider the tree T_{xy} such that $E(T_{xy}) = E(T_x) \cup \{e\} \cup E(T_y)$, where e is the edge mentioned in the previous paragraph. Note that T_{xy} is a spanning tree of $G[W(x') \cup W(y')]$. This spanning tree has a leaf, say y_1, in $W(y')$, as e is incident on at most one leaf of T_y.

Consider the partition $\mathcal{W}_1$ obtained from $\mathcal{W}$ by removing $W(x'), W(y')$ and adding W_{x°, W_{y°. Here $W_{x^\circ} = (W(x') \cup W(y')) \setminus \{y_1\}$ and $W_{y^\circ} = \{y_1\}$. Formally, $\mathcal{W}_1 = (\mathcal{W} \cup \{W_{x^\circ}, W_{y^\circ}\}) \setminus \{W(x'), W(y')\}$. Let $F_1 = (F \cup E(T_{xy})) \setminus (E(T_x) \cup E(T_y))$. It is easy to verify that $\mathcal{W}_1$ is a G/F_1-witness structure of G. As F_1 is obtained from F by removing an edge incident on y_1 and adding edge e (which was not in F), we have $|F_1| = |F|$. Also, note that $\mathsf{cost}(F_1) < \mathsf{cost}(F)$ as the witness set corresponding to y' no longer contributes to the cost.

We argue that F_1 is in $\mathcal{F}$. Let x° and y° be the two vertices corresponding to witness sets W_{x° and W_{y°, respectively. Note that the graph obtained from G/F by deleting vertices x and y is the same as the graph obtained from G/F_1 by deleting vertices x° and y°. As $W(x) \subseteq W_{x^\circ}$, x° covers all the edges in G/F that were covered by x. Also, as y was not in a vertex cover, it did not cover any edge in G/F. This implies $\mathtt{vc}(G/F) = \mathtt{vc}(G/F_1)$. Thus, $\mathtt{vc}(G/F_1) \leq \mathtt{vc}(G) - d$, and F_1 is in $\mathcal{F}$. But this contradicts the fact that F is a set of edges with minimum cost. Hence, our assumption was wrong and there exists a set of edges in F whose cost is zero.

Consider a set $F \in \mathcal{F}$ such that $\mathsf{cost}(F) = 0$. Let $\langle X', Y' \rangle$ be the partition of $V(G/F')$ such that X' is a vertex cover of G and for every $y' \in Y'$, $|\psi^{-1}(\{y'\})| = 1$. Let $X_{\mathsf{el}} = \bigcup_{x' \in X'} \psi^{-1}(\{x'\})$. Alternately, X_{el} is the subset of vertices in $V(G)$ such that $\psi(X_{\mathsf{el}}) = X'$. Define $X_s := X \setminus X_{\mathsf{el}}$ and $Y_s := X_{\mathsf{el}} \cap Y$. We argue that $\langle X_s, Y_s \rangle$ is a solution pair. As $|\psi^{-1}(\{y'\})| = 1$ for every $y' \in Y'$, $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ is a vertex cover of G. Recall that F is a subset of edges in a spanning forest of G. As $V(F) \subseteq X_{\mathsf{el}}$, the rank of $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ is at least $|F| = k$. Note that the set X' can be obtained from X_{el} by contracting the edges in F. As F is a subset of edges of a forest, we get $|X_{\mathsf{el}}| \leq |X'| + |F| \leq |X| - d + |F| = |X| + k - d$. Hence, $\langle X_s, Y_s \rangle$ satisfies all three properties.

($\Leftarrow$) Suppose that there is a solution pair $\langle X_s, Y_s\rangle$ such that $X_s \subseteq X$ and $Y_s \subseteq Y$ with the following properties: (i) $X_{el} := (X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) = \mathsf{rank}(X_{\mathsf{el}}) \geq k$, and (iii) $|Y_s| - |X_s| \leq k - d$, i.e., $|X_{el}| \leq |X| + k - d$. Let F be the edge set of a spanning forest of $G[X_{el}]$ of size k. Such a set of edges exists as the second property ensures that $\mathsf{rank}(X_{\mathsf{el}}) \geq k$. As X_{el} is a vertex cover of G, the set $V(G[X_{\mathsf{el}}]/F)$ is a vertex cover of G/F. Since F is the edge set of a spanning forest of $G[X_{\mathsf{el}}]$, we have $\mathtt{vc}(G/F) \leq |V(G[X_{\mathsf{el}}]/F)| = |X_{\mathsf{el}}| - |F| \leq |X| + (|F| - d) - |F|$. The last inequality is implied by the third property and the fact that $|F| = k$. This implies $\mathtt{vc}(G/F) \leq |X| - d$ and as X is a minimum vertex cover of G, we have $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. ◀

In the following lemma, we argue that there exists a solution pair $\langle X_s, Y_s\rangle$ such that X_s does not contain any vertex in X which is adjacent to a pendant vertex. For example, in Figure 2, there exists a solution pair $\langle X_s, Y_s\rangle$ such that $x_1 \notin X_s$.

▶ **Lemma 14.** *Consider a connected graph G, a minimum vertex cover X of G, and two integers ℓ and d. Suppose that there exists a vertex x° in X which is adjacent to a pendant vertex. Suppose that there are subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq \ell$, and (iii) $|Y_s| - |X_s| \leq \ell - d$. Then, there are subsets $X'_s \subseteq X$ and $Y'_s \subseteq V(G) \setminus X$ that satisfy these three conditions and $x^\circ \notin X'_s$.*

Proof. If $x^\circ \notin X_s$ then the lemma is vacuously true. Consider the case where $x^\circ \in X_s$. Let y° be a pendant vertex in G which is adjacent to x°. As $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, y° is in it. More specifically, $y^\circ \in Y_s$. Define $X'_s := X_s \setminus \{x^\circ\}$ and $Y'_s := Y_s \setminus \{y^\circ\}$.

As $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, and y° is a pendant vertex, it follows that $(X \setminus (X_s \cup \{x^\circ\}) \cup (Y_s \setminus \{y^\circ\})$ is also a vertex cover of G. As y° is not adjacent to any vertex in $(X \setminus X_s) \cup Y_s$, we have $\mathsf{rank}((X \setminus X_s) \cup (Y_s \setminus \{y^\circ\})) = \mathsf{rank}((X \setminus X_s) \cup Y_s) \geq \ell$. Removing a vertex from X_S, which is the same as adding a vertex in $(X \setminus X_s) \cup (Y_s \setminus \{y^\circ\}))$, cannot decrease its rank. This implies $\mathsf{rank}((X \setminus X'_s) \cup Y'_s)) \geq \ell$. Note that $|X'_s| = |X_s| - 1$ and $|Y'_s| = |Y_s| - 1$. Hence, $|Y'_s| - |X'_s| \leq \ell - d$. As $\langle X'_s, Y'_s\rangle$ satisfies all the three properties, and $x^\circ \notin X'_s$, we get a solution pair with the desired properties. ◀

2.4 Parameterized complexity

An instance of a parameterized problem Π consists of an input I, which is an input of the non-parameterized version of the problem, and an integer k, which is called the *parameter*. A problem Π is said to be *fixed-parameter tractable*, or FPT, if given an instance (I, k) of Π, we can decide whether (I, k) is a YES-instance of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$. Here, $f : \mathbb{N} \mapsto \mathbb{N}$ is some computable function depending only on k. Parameterized complexity theory provides tools to rule out the existence of FPT algorithms under plausible complexity-theoretic assumptions. For this, a hierarchy of parameterized complexity classes $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \cdots \subseteq \mathsf{XP}$ was introduced, and it was conjectured that the inclusions are proper. The most common way to show that it is unlikely that a parameterized problem admits an FPT algorithm is to show that it is $\mathsf{W}[1]$ or $\mathsf{W}[2]$-hard. It is possible to use reductions analogous to the polynomial-time reductions employed in classical complexity. Here, the concept of $\mathsf{W}[1]$-hardness replaces the one of NP-hardness, and we need not only to construct an equivalent instance FPT time, but also to ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original instance. These types of reductions are called *parameter preserving reductions*. For a detailed introduction to parameterized complexity and related terminologies, we refer the reader to the recent books by Cygan et al. [14] and Fomin et al. [17].

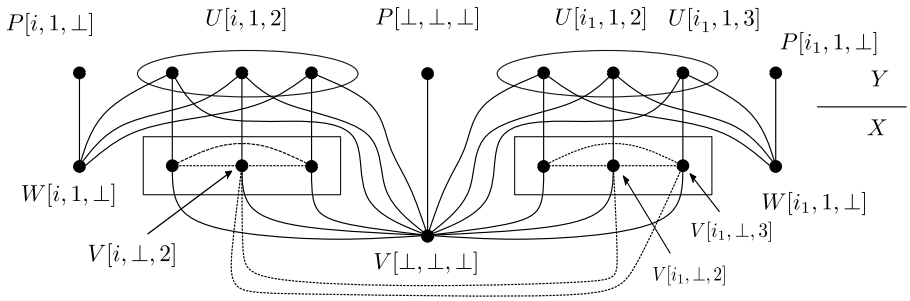

Figure 3 Reduction from $(3 \times q)$-Multicolored Clique to Contraction(vc) for $\ell = 1$. Every vertex below the horizontal line (accompanied by X, Y) on right is in X which is a minimum vertex cover of G'. Dashed edges shows edges in G.

In the Multicolored Independent Set problem, the input is a graph G, an integer q, and a partition $\langle V_1, V_2, \ldots, V_q \rangle$ of $V(G)$. The objective is to determine whether there exists a multicolored independent set in G. We say that an independent set in G is *multicolored* if it contains one vertex from V_i for every $i \in [q]$. Note that it is safe to assume that each V_i is a clique in G. We will need the following result.

▶ **Proposition 15** (cf. Theorem 14.21 in [14]). Multicolored Independent Set *parameterized by the size of the solution q is* W[1]-hard. *Moreover, unless the* ETH *fails, it does not admit an algorithm running in time $f(q) \cdot n^{o(q)}$ for any computable function $f : \mathbb{N} \mapsto \mathbb{N}$.*

A *reduction rule* is a polynomial-time algorithm that takes as input an instance of a problem and outputs another, usually reduced, instance. A reduction rule said to be *applicable* on an instance if the output instance and input instance are different. A reduction rule is *safe* if the input instance is a Yes-instance if and only if the output instance is a Yes-instance.

3 NP-hardness results

In this section we prove Theorem 1. The first and the second item in the statement of Theorem 1 follow directly from Observation 11 and Observation 12, respectively. Hence, we focus on the third case in this section. Recall that in the Multicolored Independent Set problem, the input is a graph G, an integer q, and a partition $\langle V_1, V_2, \ldots, V_q \rangle$ of $V(G)$. We consider a special case of this problem and call it $(3 \times q)$-Multicolored Independent Set. In this problem, the input is the same as that of Multicolored Independent Set, but it comes with a guarantee that every color class has exactly three vertices. The NP-hardness of this problem follows from the standard reduction from 3-SAT to Independent Set (see, for example, [31, Theorem 8.8]). This reduction ensures that each color class is a clique of size two or three. For every color class V_i that contains two vertices, we add a new vertex to V_i and make it adjacent to every vertex in the graph.

The reduction: The reduction takes as input an instance $(G, q, \langle V_1, V_2, \ldots, V_q \rangle)$ of $(3 \times q)$-Multicolored Independent Set, a positive integer ℓ, and returns an instance (G', k, d) of Contraction(vc) such that $k = d + \frac{\ell-1}{\ell+3} \cdot d$. For notational convenience, rename the three vertices in the ith color class of G as $V[i, \perp, 1], V[i, \perp, 2]$, and $V[i, \perp, 3]$ for every $i \in [q]$. We use a similar notation to refer to new vertices added to G in order to construct G'. We use i and j as the running variables in the set $[q]$ and $[\ell]$, respectively. See Figure 3 for an

illustration for the case when $\ell = 1$. The reduction adds the following vertices to a copy of G to construct G':

- $W[i, j, \perp]$ and $P[i, j, \perp]$ for every $i \in [q]$ and every $j \in [\ell]$,
- $U[i, j, 1], U[i, j, 2]$, and $U[i, j, 3]$, for every $i \in [q]$ and every $j \in [\ell]$, and
- two vertices denoted by $V[\perp, \perp, \perp]$ and $P[\perp, \perp, \perp]$.

It adds the following edges:

- For every $i \in [q]$ and $j \in [\ell]$, it adds the four edges incident on $W[i, j, \perp]$ whose other endpoints are $P[i, j, \perp]$, $U[i, j, 1]$, $U[i, j, 2]$, and $U[i, j, 3]$.
- For every $i \in [q]$ and $j \in [\ell]$, it adds three matching edges whose endpoints are $\{V[i, \perp, 1], U[i, j, 1]\}$, $\{V[i, \perp, 2], U[i, j, 2]\}$, and $\{V[i, \perp, 3], U[i, j, 3]\}$.
- For every $i \in [q]$ and $j \in [\ell]$, it adds three edges incident on $V[\perp, \perp, \perp]$ whose other endpoints are $V[i, \perp, 1]$, $V[i, \perp, 2]$, and $V[i, \perp, 3]$. It adds three more edges incident on $V[\perp, \perp, \perp]$ whose other endpoints are $U[i, j, 1]$, $U[i, j, 2]$, and $U[i, j, 3]$.
- It adds an edge with endpoints $V[\perp, \perp, \perp]$ and $P[\perp, \perp, \perp]$.

This completes the construction of G'. The reduction sets $d = (\ell+3)\cdot q$ and $k = d+(\ell-1)\cdot q$, and returns (G', k, d) as the instance of Contraction(vc).

We define sets V, U, W, and P in the natural way, i.e., V is the collection of all the vertices that have representation $V[i, j, \perp]$ for some $i \in [q]$ and $j \in [\ell]$. We define the other sets similarly. Note that $V[\perp, \perp, \perp] \notin V$ and $P[\perp, \perp, \perp] \notin P$. By the construction, every vertex in $\{P[\perp, \perp, \perp]\} \cup P$ is a pendant vertex.

For the sake of simplicity, we start by presenting an overview of the correctness of the reduction for the case where $\ell = 1$, i.e., $k = d$. The formal proof is provided after the overview. By Lemma 13, there is a solution F of (G', k, d) if and only if there exists a solution pair $\langle X_s, Y_s\rangle$ such that (i) $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ is a vertex cover of G', (ii) $\mathsf{rank}(X_{\mathsf{el}}) \geq |F| = k$, and (iii) $|X_{\mathsf{el}}| \leq |X| + |F| - d \leq |X| + k - d = |X|$. The reduction ensures that the size of X is $k + 1$. With this, the second and the third conditions force X_{el} to be a connected set of the same size as that of X. For the example in Figure 3, Lemma 14 implies that $V[\perp, \perp, \perp]$, $W[i, 1, \perp]$ and $W[i_1, 1, \perp]$ are in X_{el}. Hence, to provide connectivity between $V[\perp, \perp, \perp]$ and $W[i, 1, \perp]$, at least one of the vertices in $\{U[i, 1, 1], U[i, 1, 2], U[i, 1, 3]\}$ needs to be in X_{el}. However, as $|X_{\mathsf{el}}| = |X|$, at least one vertex in $\{V[i, \perp, 1], V[i, \perp, 2], V[i, \perp, 3]\}$ needs to be out of X_{el}, i.e., in X_s. As this is true for every color class, X_s includes at least one vertex from it. The first condition enforces X_s to be an independent set. This implies that X_s can include at most one vertex from each color class. Moreover, if X_s includes $V[i, \perp, 2]$ then it cannot include $V[i_1, \perp, 2]$ or $V[i_1, \perp, 3]$. These are precisely the conditions we want for encoding an instance of Multicolored Independent Set. This concludes the overview of the reduction.

We formalize the above ideas in Lemma 17 and Lemma 18. Before that, in the next lemma we argue about the size of a minimum vertex cover of G'.

▶ **Lemma 16.** *The set $X := V \cup W \cup \{V[\perp, \perp, \perp]\}$ is a minimum vertex cover of G'.*

Proof. By the construction of G', it is easy to verify that X is a vertex cover of G'. To prove that it is a minimum vertex cover, we show that there is a matching M of size $|X|$ in G'. Initialize $M = \emptyset$. For every vertex in $\{V[\perp, \perp, \perp]\} \cup W$, include the edge in M incident on its pendant neighbor. For every $i \in [q]$, include the three edges whose endpoints are $\{V[i, \perp, 1], U[i, j, 1]\}$, $\{V[i, \perp, 2], U[i, j, 2]\}$, and $\{V[i, \perp, 3], U[i, j, 3]\}$. It is easy to verify that M is a matching of size $|X|$. This implies that X is a minimum vertex cover of G'. ◀

▶ **Lemma 17.** *If* $(G, q, \langle V_1, V_2, \ldots, V_q\rangle)$ *is a* Yes*-instance of* $(3 \times q)$-Multicolored Independent Set*, then* (G', k, d) *is a* Yes*-instance of* Contraction(vc).

Proof. Suppose that Q is a multicolored independent set in G. Let $\{V[i, \bot, z_i]\} = Q \cap V_i$ for $z_i \in \{1, 2, 3\}$. By Lemma 16, $X := V \cup W \cup \{V[\bot, \bot, \bot]\}$ is a minimum vertex cover of G'. Define

$$X_s := \{V[i, \bot, z_i] \mid i \in [q]\},\ Y_s := \{U[i, j, z_i] \mid i \in [q] \ \wedge\ j \in [\ell]\} \text{ and } X_s := X \setminus X_{\mathsf{el}}.$$

It is easy to verify that X_{el} is a vertex cover of G'. As $G[X_{\mathsf{el}}]$ is a connected graph, $\mathsf{rank}(X_{\mathsf{el}}) = |X_{\mathsf{el}}| - 1 = |X| + (\ell - 1) \cdot q - 1$. As $|X| = (\ell + 3) \cdot q + 1 = d + 1$, we get $\mathsf{rank}(X_{\mathsf{el}}) = d + 1 + (\ell - 1) \cdot q - 1 = k$. Also, $|Y_s| - |X_s| = (\ell - 1) \cdot q = k - d$. This implies that the pair $\langle X_s, Y_s\rangle$ satisfies all the three conditions mentioned in the statement of Lemma 13. As $k < \mathsf{rank}(G)$, Lemma 13 implies that there exists a set of edges F of size at most k in G' such that $\mathtt{vc}(G'/F) \leq \mathtt{vc}(G) - d$. Hence, (G', k, d) is a Yes-instance of Contraction(vc). ◀

▶ **Lemma 18.** *If* (G', k, d) *is a* Yes*-instance of* Contraction(vc) *then* $(G, q, \langle V_1, V_2, \ldots, V_q\rangle)$ *is a* Yes*-instance of* $(3 \times q)$-Multicolored Independent Set.

Proof. Suppose that F' is a solution of (G', k, d), i.e., $\mathtt{vc}(G'/F) \leq \mathtt{vc}(G) - d$ and $|F'| \leq k$. As $k < \mathsf{rank}(G)$, we can assume, without loss of generality, that $|F'| = k$. Lemma 13 implies that there exists a solution pair $\langle X_s, Y_s\rangle$ that satisfies the three conditions mentioned in its statement. Recall that every vertex in $\{V[\bot, \bot, \bot]\} \cup W$ is adjacent to some pendant vertex in G'. Lemma 14 implies that there exits a solution pair $\langle X_s, Y_s\rangle$ with the additional property that $X_s \cap (\{V[\bot, \bot, \bot]\} \cup W) = \emptyset$. This implies $(\{V[\bot, \bot, \bot]\} \cup W) \subseteq X_{\mathsf{el}} := (X \setminus X_s) \cup Y_s$.

We first argue that $q \leq |X_s|$. By the second condition in Lemma 13, $\mathsf{rank}(X_{\mathsf{el}}) \geq k$. By the third condition in Lemma 13, $|X_{\mathsf{el}}| \leq |X| + k - d$. As $|X| = d + 1$, it follows that $|X_{\mathsf{el}}| \leq k + 1$. Hence, the number of vertices in $G'[X_{\mathsf{el}}]$ is at most $k + 1$, whereas the number of edges in a spanning forest of $G'[X_{\mathsf{el}}]$ is at least k. This implies that $G'[X_{\mathsf{el}}]$ is connected. Fix integers $i \in [q]$ and $j \in [\ell]$. By the construction of G', every path between $V[\bot, \bot, \bot]$ to $W[i, j, \bot]$ contains at least one vertex in $\{U[i, j, 1], U[i, j, 2], U[i, j, 3]\}$. As $(\{V[\bot, \bot, \bot]\} \cup W) \subseteq X_{\mathsf{el}}$, and $G[X_{\mathsf{el}}]$ is connected, Y_s contains at least one vertex in the set. As this is true for every $i \in [q]$ and $j \in [\ell]$, we have $|Y_s| \geq q \cdot \ell$. By the third condition mentioned in Lemma 13, $|Y_s| - |X_s| \leq k - d$. Substituting $k - d = (\ell - 1) \cdot q$, we get $q \leq |X_s|$.

As $X_s \cap (\{V[\bot, \bot, \bot]\} \cup W) = \emptyset$ and $X := V \cup W \cup \{V[\bot, \bot, \bot]\}$, we have $X_s \subseteq V$. By the first condition mentioned in Lemma 13, $(X \setminus X_s) \cup Y_s$ is a vertex cover of G'. As each V_i is a clique in G', we have that $|X_s \cap V_i| \leq 1$ for every $i \in [q]$. This, along with the fact that $q \leq |X_s|$ imply that $|X_s \cap V_i| = 1$ for every $i \in [q]$. Recall that $G'[V]$ is isomorphic to G. As $G'[X_s]$ is an independent set in G', it follows that X_s is also an independent set in G. It is also evident that it is multicolored. This implies that $(G, q, \langle V_1, V_2, \ldots, V_q\rangle)$ is a Yes-instance of $(3 \times q)$-Multicolored Independent Set. ◀

As mentioned before, the first and the second point in the statement of the theorem follow directly from Observation 11 and Observation 12, respectively. Lemma 17 and Lemma 18 imply that the reduction is correct. By the description of the reduction, it outputs the constructed instance in polynomial time. Hence, the third point in the statement of Theorem 1 is correct, which concludes its proof.

4 W[1]-hardness results

In this section we prove Theorem 2. That is, we show that Contraction(vc) is W[1]-hard when parameterized by the solution size k plus the measure d. Moreover, unless the ETH fails, it does not admit an algorithm running in time $f(k+d) \cdot n^{o(k+d)}$ for any computable function $f : \mathbb{N} \mapsto \mathbb{N}$. To obtain these results, we introduce the Edge Induced Forest problem. We define this problem formally in Subsection 4.1, and present a parameter preserving reduction from Multicolored Independent Set to it. This reduction, along with known results about Multicolored Independent Set, imply the corresponding result for Edge Induced Forest. The proof is presented in Theorem 19. In Subsection 4.2, we present a parameter preserving reduction from Edge Induced Forest to Contraction(vc). This reduction, along with Theorem 19, imply the correctness of Theorem 2.

4.1 Edge Induced Forest is W[1]-hard

We define the following problem.

Edge Induced Forest **Input:** A graph G and an integer ℓ. **Question:** Does there exist a set F of at least ℓ edges in G such that $G[V(F)]$ is a forest?

We note that a similar problem called Induced Forest has already been studied. In this problem, the input is the same but the objective is to find a subset X of *vertices* of G of size at least ℓ such that $G[X]$ is a forest. The general result of Khot and Raman [29] implies that Induced Forest is W[1]-hard when parameterized by the size of the solution ℓ. As expected, we can prove a similar result for Edge Induced Forest.

▶ **Theorem 19.** Edge Induced Forest, *parameterized by the size of the solution ℓ, is* W[1]-hard. *Moreover, unless the* ETH *fails, it does not admit an algorithm running in time $f(\ell) \cdot n^{o(\ell)}$ for any computable function $f : \mathbb{N} \mapsto \mathbb{N}$.*

Proof. We present a simple parameter preserving reduction from Multicolored Independent Set. The reduction takes as input an instance $(G, q, \langle V_1, V_2, \ldots, V_q \rangle)$ of Multicolored Independent Set, and constructs another graph G' from G by adding a universal vertex α to G. Formally, it adds a vertex α to $V(G)$, and adds edge $u\alpha$ to $E(G)$ for every vertex u in $V(G) \setminus \{\alpha\}$ to obtain G'. It adds $q+1$ pendant vertices adjacent to α. Formally, for every $i \in [q+1]$, it adds a vertex x_i to $V(G')$, and an edge $x_i\alpha$ to $E(G')$. Let P be the collection of all the pendant vertices added in this step. It sets $\ell = 2 \cdot q + 1$, and returns the instance (G', ℓ) of Edge Induced Forest as the constructed instance. This completes the description of the reduction.

We now argue that the reduction is safe. In the forward direction, suppose that Q is a multicolored independent set in G. Define $F := \{x_i\alpha \mid \forall\, x_i \in P\} \cup \{u_i\alpha \mid u_i \in Q \cap V_i \;\forall\, i \in [q]\}$. It is easy to verify that F is a solution of (G', ℓ).

In the reverse direction, suppose that F is a solution of (G', ℓ), i.e., $G'[V(F)]$ is a forest and $|F| \geq \ell$. We first argue that $\alpha \in V(F)$. Assume, for the sake of contradiction, that α is not in $V(F)$. This implies that F contains $2 \cdot q + 1$ many edges in $G' - \{\alpha\}$. Note that every vertex in P is an isolated vertex in $G' - \{\alpha\}$. Hence, $V(F) \subseteq \bigcup_{i \in [q]} V_i$. As $|F| \leq V(F)$, it follows that there exists $i \in [q]$ such that $|V(F) \cap V_i| \geq 3$. However, as V_i is a clique in G', this contradicts the fact that $G'[V(F)]$ is a forest. Hence, $\alpha \in V(F)$.

As $\alpha \in V(F)$, it is safe to assume that F contains all the edges in $F_P := E_{G'}(\{\alpha\}, P)$. Since $|P| = q + 1$, F contains at least q many edges whose endpoints are in $V(G') \setminus P$.

The following two statements are direct consequences of the facts that α is an universal vertex, V_i is a clique in G' for any $i \in [q]$, and $G'[F]$ is a forest: (i) For any $i \in [q]$, we have $|V(F) \cap V_i| \leq 1$, and in particular $|F \cap E_{G'}(V_i)| = 0$. (ii) For any $i \neq j \in [q]$, $|F \cap E_{G'}(V_i, V_j)| = 0$. This implies that every edge in $F \setminus F_P$ has α as one of its endpoints. As there are at least q edges $F \setminus F_P$, $|V(F) \cap V_i| = 1$ for every $i \neq j \in [q]$.

We define a subset Q of $V(G)$ as $Q := \{u \in V(G) \mid u\alpha \in F \text{ and } u \in V_i \text{ for some } i \in [q]\}$. As for every $i \in [q]$, we have $|V(F) \cap V_i| = 1$, this implies $|Q \cap V_i| = 1$. We argue that Q is a multicolored independent set in G. Consider any two indices $i \neq j \in [q]$, and let u_i, u_j be the unique vertices in $Q \cap V_i$ and $Q \cap V_j$, respectively. If $u_iu_j \in E(G)$ then $u_iu_j \in E(G')$ as $E(G) \subseteq E(G')$. However, as $u_i\alpha, u_j\alpha \in F$, this contradicts the fact that $G'[F]$ is a forest. Hence, vertices u_i and u_j are not adjacent in G. Since i, j are arbitrary indices in $[q]$, this is true for any $i \neq j \in [q]$, and therefore Q is a multicolored independent set in G.

This implies that $(G, q, \langle V_1, V_2, \ldots, V_q\rangle)$ is a Yes-instance of Multicolored Independent Set if and only if (G', ℓ) is a Yes-instance of Edge Induced Forest. By the description of the reduction, it outputs the constructed instance in polynomial time. The W[1]-hardness of the problem follows from Proposition 15. It is also easy to see that if Edge Induced Forest admits an algorithm with running time $f(\ell) \cdot n^{o(\ell)}$ for some computable function $f : \mathbb{N} \mapsto \mathbb{N}$, then Multicolored Independent Set also admits an algorithm with running time $f(q) \cdot n^{o(q)}$, which contradicts Proposition 15. ◀

4.2 Contraction(vc) is W[1]-hard

In this subsection we present a parameter preserving reduction from Edge Induced Forest to Contraction(vc).

The reduction: The reduction takes as input an instance (G, ℓ) of Edge Induced Forest and returns an instance (G', k, d) of Contraction(vc). It constructs a graph G' from G as follows.

- It initializes $V(G') = E(G') = \emptyset$.
- For every vertex u in $V(G)$, it adds two vertices z_u, p_u to $V(G')$ and the edge z_up_u to $E(G')$.
- For every edge uv in $E(G)$, it adds the vertex set $\{y^a_{uv}, y^b_{uv}, y^c_{uv}, w^1_{uv}, w^2_{uv}, p^1_{uv}, p^2_{uv}\}$ to $V(G')$. It adds edge $\{z_uy^c_{uv}, z_vy^c_{uv}\}$ to $E(G')$. These edges encode adjacency relations in G. It also adds edges $\{y^a_{uv}y^b_{uv}, y^a_{uv}y^c_{uv}, y^b_{uv}w^1_{uv}, y^b_{uv}w^2_{uv}, w^1_{uv}p^1_{uv}, w^2_{uv}p^2_{uv}\}$ to $E(G')$. These edges are part of a gadget which is private to edge uv.

This completes the construction of G'. The reduction sets $k = 4 \cdot \ell$, $d = 3 \cdot \ell$, and returns (G', k, d) as the constructed instance. This completes the description of the reduction. Note that, indeed, $k < \mathsf{rank}(G')$ and $d \leq k < 2d$ (more precisely, $k - d = \frac{d}{3}$). See Figure 4 for an illustration.

Before proving the correctness of the reduction, we first note some properties of the graph G'. We define the following sets:

- $Z := \{z_u \in V(G') \mid u \in V(G)\}$,
- $Y^{abc} := Y^a \cup Y^b \cup Y^c$ where $Y^a := \{y^a_{uv} \in V(G') \mid uv \in E(G)\}$, $Y^b := \{y^b_{uv} \in V(G') \mid uv \in E(G)\}$, and $Y^c := \{y^c_{uv} \in V(G') \mid uv \in E(G)\}$,
- $W := \{w^1_{uv}, w^2_{uv} \in V(G') \mid uv \in E(G)\}$, and
- $P := \{p_u \in V(G') \mid u \in V(G)\} \cup \{p^1_{uv}, p^2_{uv} \mid uv \in E(G)\}$.

Note that $\langle Z, Y^{abc}, W, P\rangle$ is a partition of $V(G')$, each vertex in P is a pendant vertex, and each vertex in $Z \cup W$ is adjacent to a pendant vertex in P. Moreover, $\mathsf{rank}(G') > k$. Note that $X := Z \cup W \cup Y^a$ is an independent set in G'. In the next lemma, we argue that it

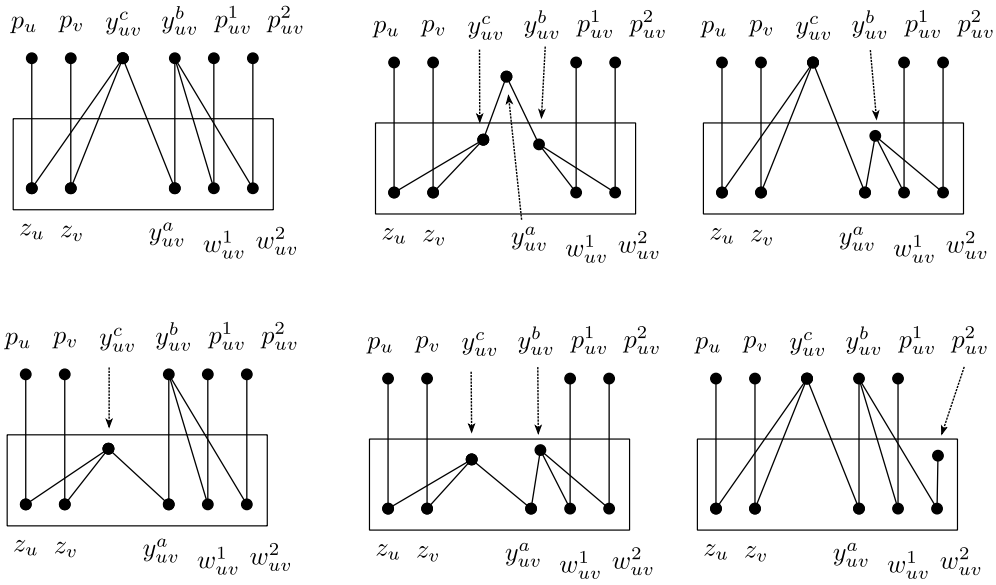

Figure 4 The top-left figure illustrates an encoding of edge uv in G while reducing from an instance of Edge Induced Forest to an instance of Contraction(vc). The remaining five figures correspond to the partition of Y_s mentioned in the proof of Lemma 22.

is also a minimum vertex cover of G', which implies that, as claimed in the statement of Theorem 2, G' is a bipartite graph with a bipartition $\langle X, Y\rangle$ such that X is a minimum vertex cover of G'.

▶ **Lemma 20.** *The set $X = Z \cup W \cup Y^a$ is a minimum vertex cover of G'.*

Proof. By the construction of G', it follows that X is a vertex cover of G'. To prove that it is a minimum vertex cover, we show that there is a matching of size $|X|$ in G. Consider the following set of edges $M := \{z_u p_u \mid u \in V(G)\} \cup \{w^1_{uv}p^1_{uv}, w^2_{uv}p^2_{uv} \mid uv \in E(G)\} \cup \{y^a_{uv}y^c_{uv} \mid uv \in E(G)\}$. It is easy to verify that M is a matching in G' of size $|X|$. Hence, any vertex cover has size at least $|X|$. This implies that X is a minimum vertex cover of G'. ◀

▶ **Lemma 21.** *If (G, ℓ) is a* Yes*-instance of* Edge Induced Forest*, then (G', k, d) is a* Yes*-instance of* Contraction(vc)*.*

Proof. Let F be a solution of (G, ℓ) i.e., $G[V(F)]$ is a forest and $|F| \geq \ell$. We assume, without loss of generality, that $|F| = \ell$. By Lemma 20, the set $X := Z \cup W \cup Y^a$ is a minimum vertex cover of G'. Note that X is also an independent set in G'. We denote the independent set $V(G') \setminus X$ by Y.

We construct a solution pair $\langle X_s, Y_s\rangle$ using F. Define $X_s := \{y^a_{uv} \in Y^a \mid uv \in F\}$, and $Y_s := Y^b_s \cup Y^c_s$ where $Y^b_s := \{y^b_{uv} \in Y^b \mid uv \in F\}$, and $Y^c_s := \{y^c_{uv} \in Y^c \mid uv \in F\}$. It is easy to verify that the set X_{el} obtained from X by removing the vertices in X_s and adding the vertices in Y_s, is another vertex cover. Formally, $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$. Note that $X \setminus X_s = (Z \cup W \cup Y^a) \setminus X_s = Z \cup W \cup (Y^a \setminus X_s)$, as by definition $X_s \subseteq Y^a$. As every vertex $y^a_{uv} \in Y^a$ is adjacent to only y^c_{uv} and y^b_{uv} in Y, $(X \setminus X_s) \cup Y_s = Z \cup W \cup (Y^a \setminus X_s) \cup Y^b_s \cup Y^c_s$ is a vertex cover of G' and its size is $|X| - \ell + 2 \cdot \ell = |X| + k - d$. By the construction of G', one

can obtain graph $G'[Z \cup Y_s^c]$ by subdividing every edge in $G[F]$. Hence, $G'[Z \cup Y_s^c]$ is a forest with $2 \cdot \ell$ edges and some isolated vertices. Also, $G'[Y_s^b \cup W]$ is a forest with at least $2 \cdot \ell$ edges (two edges corresponding to each vertex in Y_s^b). As the vertices in $Z \cup Y^c$ and $Y^b \cup W$ are not adjacent, $G'[Z \cup Y_s^c \cup Y_s^b \cup W]$ is a forest with 4ℓ edges. This implies that $\mathsf{rank}(X_{\mathsf{el}}) = 4 \cdot \ell$. Hence, $\langle X_s, Y_s \rangle$ satisfies the three conditions mentioned in Lemma 13. As $k < \mathsf{rank}(G)$, Lemma 13 implies that there is a subset F' of $E(G')$ such that $\mathtt{vc}(G'/F') \leq \mathtt{vc}(G) - d$. Hence, (G', k, d) is a Yes-instance of Contraction($\mathtt{vc}$). ◀

We first present a brief overview of the proof of the correctness in the backward direction. By Lemma 13, there is a solution F of (G', k, d) if and only if there exists a solution pair $\langle X_s, Y_s \rangle$ such that (i) $X_{\mathsf{el}} = (X \setminus X_s) \cup Y_s$ is a vertex cover of G', (ii) $\mathsf{rank}(X_{\mathsf{el}}) \geq |F| = k = 4 \cdot \ell$, and (iii) $|Y_s| - |X_s| \leq k - d = \ell$. Note that as X and $Y = V(G) \setminus X$ are independent sets in G', every edge in $E(X_{\mathsf{el}})$ is incident on exactly one vertex in Y_s. We can interpret the second condition as a *value function* and the third condition as a *cost function*. In other words, our objective is to find sets X_s, Y_s such that their cost, i.e., $|Y_s| - |X_s|$, is at most ℓ whereas their value, i.e., the rank of edges in $E(X_{\mathsf{el}})$ that are incident on Y_s, is at least $4 \cdot \ell$. Lemma 14 implies that the vertices of the form z_u, w_{uv}^1, and w_{uv}^2 are in X_{el}. The first condition implies that only the five configurations shown in Figure 4 are possible (the top-left is *not* a configuration). Starting from top-middle and moving row-wise, the individual value and cost of these configurations are $(4, 1)$, $(3, 1)$, $(3, 1)$, $(6, 2)$, and $(1, 1)$, respectively. To meet both the value and budget constraints, every vertex in X_s, Y_s needs to be the of first type. This implies there are ℓ vertices in X_s that are of the form y_{uv}^a, and Y_s contains the corresponding vertices of the form y_{uv}^b and y_{uv}^c. We argue that the edges corresponding to vertices in Y_{uv}^c form a solution of (G, ℓ) and formalize these ideas in the next lemma.

▶ **Lemma 22.** *If (G', k, d) is a* Yes*-instance of* Contraction($\mathtt{vc}$)*, then (G, ℓ) is a* Yes*-instance of* Edge Induced Forest*.*

Proof. Suppose that F' is a solution of (G', k, d), i.e., $\mathtt{vc}(G'/F') \leq \mathtt{vc}(G') - d$ and $|F'| \leq k$. As $k < \mathsf{rank}(G)$, we can assume, without loss of generality, that $|F'| = k$. Lemma 13 implies that there exists a solution pair $\langle X_s, Y_s \rangle$ that satisfies the three conditions mentioned in its statement. Recall that every vertex in $Z \cup W$ is adjacent to some pendant vertex in G'. Lemma 14 implies that there exists a solution pair $\langle X_s, Y_s \rangle$ with the additional property that $X_s \cap (Z \cup W) = \emptyset$. As $X_s \subseteq X = Z \cup W \cup Y^a$, this implies that $X_s \subseteq Y^a$.

We argue that $|X_s| = \ell$. We partition the vertices in Y_s into the following five sets.

- $Y[1,1,1] := \{y_{uv}^b, y_{uv}^c \in Y_s \mid (y_{uv}^a \in X_s) \ \wedge \ (y_{uv}^b \in Y_s) \ \wedge \ (y_{uv}^c \in Y_s)\}$.
- $Y[0,1,0] := \{y_{uv}^b \in Y_s \mid (y_{uv}^a \notin X_s) \ \wedge \ (y_{uv}^b \in Y_s) \ \wedge \ (y_{uv}^c \notin Y_s)\}$.
- $Y[0,0,1] := \{y_{uv}^c \in Y_s \mid (y_{uv}^a \notin X_s) \ \wedge \ (y_{uv}^b \notin Y_s) \ \wedge \ (y_{uv}^c \in Y_s)\}$.
- $Y[0,1,1] := \{y_{uv}^b, y_{uv}^c \in Y_s \mid (y_{uv}^a \notin X_s) \ \wedge \ (y_{uv}^b \in Y_s) \ \wedge \ (y_{uv}^c \in Y_s)\}$.
- $Y[0,0,0] := Y_s \cap P$.

We can define the sets $Y[1,0,0]$, $Y[1,0,1]$, and $Y[1,1,0]$ in a similar way. Note that $y_{uv}^a \in X_s$ implies that $y_{uv}^b, y_{uv}^c \in Y_s$. Hence, we do not need to consider the sets $Y[1,0,0]$, $Y[1,0,1]$, and $Y[1,1,0]$. This also implies $2 \cdot |X_s| = |Y[1,1,1]|$. Hence, to argue that $|X_s| = \ell$, it is sufficient to prove that $|Y[1,1,1]| = 2 \cdot \ell$. As the solution pair $\langle X_s, Y_s \rangle$ satisfies the third condition, i.e., $|Y_s| - |X_s| \leq k - d$, we have

$$|Y[1,1,1]| + |Y[0,1,0]| + |Y[0,0,1]| + |Y[0,1,1]| + |Y[0,0,0]| - |X_S| \leq \ell.$$

Substituting $|Y[1,1,1,]| = 2 \cdot |X_s|$, and multiplying by two, we get the following relation.

$$|Y[1,1,1]| + 2 \cdot |Y[0,1,0]| + 2 \cdot |Y[0,0,1]| + 2 \cdot |Y[0,1,1]| + 2 \cdot |Y[0,0,0]| \leq 2 \cdot \ell. \tag{1}$$

Define $X_{\mathsf{el}} := (X \setminus X_s) \cup Y_s$. By the definition, $|E(X_{\mathsf{el}})| \geq \mathsf{rank}(X_{\mathsf{el}})$. As the solution pair $\langle X_s, Y_s \rangle$ satisfies the second condition, i.e., $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq 4 \cdot \ell$, we have $|E(X_{\mathsf{el}})| \geq \mathsf{rank}(X_{\mathsf{el}}) \geq 4 \cdot \ell$. As X, Y both are independent sets in G', every edge in $E(X_{\mathsf{el}})$ has one of its endpoints in $X \setminus X_s$ and the other one in Y_s. It is easy to verify (see Figure 4) that the number of edges incident on each vertex in $Y[1,1,1]$, $Y[0,1,0]$, $Y[0,0,1]$, $Y[0,1,1]$, $Y[0,0,0]$ is 2, 3, 3, 3, and 1, respectively. Substituting these values we get

$$2 \cdot |Y[1,1,1]| + 3 \cdot |Y[0,1,0]| + 3 \cdot |Y[0,0,1]| + 3 \cdot |Y[0,1,1]| + |Y[0,0,0]| \geq 4 \cdot \ell.$$

Dividing the inequality by two yields the following relation.

$$|Y[1,1,1]| + \frac{3}{2} \cdot |Y[0,1,0]| + \frac{3}{2} \cdot |Y[0,0,1]| + \frac{3}{2} \cdot |Y[0,1,1]| + \frac{1}{2} \cdot |Y[0,0,0]| \geq 2 \cdot \ell. \tag{2}$$

Equation (1) and (2) imply that the only feasible case is when $|Y[1,1,1]| = 2 \cdot \ell$ and all other sets have cardinality zero. This implies $|X_s| = \ell$. Also, $|E(X_{\mathsf{el}})| = 2 \cdot |Y[1,1,1]| = 4 \cdot \ell$. As $\mathsf{rank}(X_{\mathsf{el}}) \geq 4 \cdot \ell$, it follows that $G'[X_{\mathsf{el}}]$ is a forest with $4 \cdot \ell$ edges. It is easy to verify that the graph induced on $Z \cap N[Y_s \cap Y^c]$ is a forest with $2 \cdot \ell$ edges.

We now construct a solution of (G, ℓ) using the set X_s, more precisely $Y_s \cap Y^c$. Define $F := \{uv \in E(G) \mid y^c_{uv} \in Y_s\}$. As $|X_s| = \ell$, we have $|F| = \ell$. It remains to argue that $G[V(F)]$ is a forest. By the construction of G', one can obtain $G'[Z \cap N[Y \cap Y^c]]$ by subdividing every edge in $G[V(F)]$. As the former graph is a forest, we can conclude that $G[V(F)]$ is also a forest. Hence, F is a solution of (G, ℓ). This implies that if (G', k, d) is a Yes-instance of Contraction(vc) then (G, ℓ) is a Yes-instance of Edge Induced Forest. ◀

We are ready to present the proof of Theorem 2.

Proof of Theorem 2. Consider the reduction presented in this subsection. Lemma 21 and Lemma 22 imply that the reduction is safe. By the description of the reduction, it outputs the constructed instance in polynomial time. The W[1]-hardness of Contraction(vc) follows from Theorem 19. As $k = 4 \cdot \ell$ and $d = 3 \cdot \ell$, if Contraction(vc) admits an algorithm with running time $f(k+d) \cdot n^{o(k+d)}$, then Edge Induced Forest also admits an algorithm with running time $f(\ell) \cdot n^{o(\ell)}$, which contradicts Theorem 19. ◀

5 Algorithm for Contraction(vc)

In this section we prove Theorem 3. We present an algorithm that takes as input an instance (G, k, d) of Contraction(vc), and returns either Yes or No, whose high-level description is as follows (cf. Figure 1):

- If $k = \mathsf{rank}(G)$, then it uses the algorithm mentioned in Lemma 23.
- If $k < \mathsf{rank}(G)$ and $2d \leq k$, then it uses the algorithm mentioned in Lemma 24.
- If $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, then it uses the algorithm mentioned in Lemma 25.

Note that, since we can safely assume that $d \leq k \leq \mathsf{rank}(G)$, the above three cases are exhaustive. We handle each of these cases in the next three subsections (note that the first two are much easier than the last one). Subsection 5.4 contains the correctness proof of Theorem 3. Throughout this section, we assume that G is a connected graph. We justify this assumption in Subsection 5.4.

5.1 First case: $k = \mathsf{rank}(G)$

It is sufficient to prove the following lemma to handle this case.

▶ **Lemma 23.** *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction($\mathtt{vc}$) *with a guarantee that $k = \mathsf{rank}(G)$, runs in time $1.2738^d \cdot n^{\mathcal{O}(1)}$, and correctly determines whether it is a* Yes*-instance.*

Proof. Consider an algorithm that for input (G, k, d), runs the algorithm mentioned in Proposition 5 as a subroutine with G and $d-1$ as its input. If the subroutine concludes that $\mathtt{vc}(G) \leq d-1$, then the algorithm returns No, otherwise it returns Yes. This concludes the description of the algorithm. Its correctness and running time follow from Observation 9 and Proposition 5, respectively. ◀

5.2 Second case: $k < \mathsf{rank}(G)$ and $2d \leq k$

As in the previous subsection, it is sufficient to prove the following lemma.

▶ **Lemma 24.** *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction($\mathtt{vc}$) *with guarantees that $k < \mathsf{rank}(G)$ and $2d \leq k$, runs in time $1.2738^d \cdot n^{\mathcal{O}(1)}$, and correctly determines whether it is a* Yes*-instance.*

Proof. Consider an algorithm that for input (G, k, d), runs the algorithm mentioned in Proposition 5 as a subroutine with G and d as its input. It considers the following three cases depending on the value of $\mathtt{vc}(G)$. *Case (i)* ($\mathtt{vc}(G) < d$): It concludes that (G, k, d) is a No-instance. *Case (ii)* ($\mathtt{vc}(G) = d$): It concludes that (G, k, d) is a No-instance. *Case (iii)* ($\mathtt{vc}(G) > d$): It concludes that (G, k, d) is a Yes-instance. This completes the description of the algorithm.

We now argue the correctness of the algorithm. As the vertex cover number of any graph is a non-negative integer, if $\mathtt{vc}(G) < d$ then the input is a No-instance. Note that to eliminate all edges in a connected graph by contracting edges, one needs to contract all the edges in a spanning tree. Hence, if $\mathtt{vc}(G) = d$ then the only feasible solution of (G, k, d) is a spanning tree of G. However, as $k < \mathsf{rank}(G)$, the algorithm correctly concludes that it is a No-instance. For the third case, consider a subroutine that finds two vertices in a minimum vertex cover that are at distance at most two and contracts a shortest path between these two vertices. The existence of such vertices is guaranteed by the fact that G is a connected graph. Note that this path is of length one or two. In each iteration of the process, k drops by at most two and $\mathtt{vc}(G)$ drops by one. As $2d \leq k$, if $\mathtt{vc}(G) > d$ then the subroutine can repeat the process d times. Hence, the algorithm correctly concludes that the input instance is a Yes-instance in the third step.

The running time of the algorithm follows from its description and Proposition 5. ◀

5.3 Third case: $k < \mathsf{rank}(G)$ and $d \leq k < 2d$

The objective of this subsection is to prove the following lemma.

▶ **Lemma 25.** *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction($\mathtt{vc}$) *with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$, and correctly determines whether it is a* Yes*-instance.*

We refer readers to Section 1, in particular Figure 1, for an overview of the algorithm presented in this subsection.

5.3.1 Simplifying an instance of Contraction(vc)

We prove the following lemma, which will allow us to assume henceforth that we are equipped with a minimum vertex cover of the input graph with small rank.

▶ **Lemma 26.** *There exists an algorithm that, given as input an instance (G, k, d) of* Contraction(vc) *with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, runs in time $2.6181^k \cdot n^{\mathcal{O}(1)}$, and either correctly concludes that (G, k, d) is a* Yes*-instance, or computes a minimum vertex cover X of G such that $\mathsf{rank}(\mathrm{X}) < d$.*

Proof. Consider an algorithm that, given (G, k, d) as input, runs the algorithm mentioned in Proposition 6 as a subroutine with G and k as its input. If $\mathtt{oct}(G) > k$, then it concludes that (G, k, d) is a Yes-instance. If $\mathtt{oct}(G) \leq k$, then it uses the algorithm mentioned in Proposition 7 to compute a minimum vertex cover X of G. If $\mathsf{rank}(X) \geq d$, then it concludes that (G, k, d) is a Yes-instance. Otherwise, it returns X as the desired vertex cover. This completes the description of the algorithm.

We argue the correctness of the algorithm. Consider the case where $\mathtt{oct}(G) > k$. Recall that we denote by $\mathtt{bc}(G)$ the minimum number of edges in G that need to be contracted to make it a bipartite graph. By Observation 8, $\mathtt{oct}(G) > k$ implies that $\mathtt{bc}(G) > k$. Hence, by Observation 9, for any partition (V_L, V_R) of $V(G)$, we have $\mathsf{rank}(V_L) + \mathsf{rank}(V_R) > k$. Consider a partition (V_L, V_R) of $V(G)$ such that V_L is a minimum vertex cover of G. As V_R is an independent set, $\mathsf{rank}(V_R) = 0$. This implies $\mathsf{rank}(V_L) > k$. Hence, we can reduce $\mathtt{vc}(G)$ by d by contracting d (which is at most k) edges whose both endpoints are in V_L. Consider the case when the algorithm finds a minimum vertex cover X of G such that $\mathsf{rank}(X) \geq d$. Once again, we can reduce $\mathtt{vc}(G)$ by d by contracting d edges of a spanning forest of $G[X]$. Hence, in both these cases, the algorithm correctly concludes that the input is a Yes-instance. Otherwise, the algorithm returns a minimum vertex cover X of G such that $\mathsf{rank}(X) < d$.

The running time of the algorithm follows from its description and Proposition 6. ◀

5.3.2 Reducing to Annotated Contraction(vc)

An input of the Annotated Contraction(vc) problem consists of an instance (G, k, d) of Contraction(vc), a minimum vertex cover X of G, and two disjoint subsets X_L, X_R of X. We are interested in a vertex cover X_{el} of G whose size is not much larger than that of X but has rank at least k. To construct X_{el} from X, we need to find a *solution pair* $\langle X_s, Y_s \rangle$ such that vertices in X_s are 'moved out' of X, and vertices in Y_s are 'moved in'. Given $\langle X_L, X_R \rangle$, we add a restriction on a possible solution pair $\langle X_s, Y_s \rangle$. Namely, we are interested in X_s that contains X_R and is disjoint from X_L. The following is the formal definition of the problem.

Annotated Contraction(vc)
Input: An instance (G, k, d) of Contraction(vc), a minimum vertex cover X of G, and a tuple $\langle X_L, X_R \rangle$ such that X_L, X_R are disjoint subsets of X.
Question: Do there exist sets $X_s \subseteq X$ and $Y_s \subseteq Y$ $(= V(G) \setminus X)$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$, (iii) $|Y_s| - |X_s| \leq k - d$, and (iv) $X_L \cap X_s = \emptyset$ and $X_R \subseteq X_s$?

The first three conditions correspond to the three conditions mentioned in Lemma 13. Given an instance (G, k, d) of Contraction(vc), using Lemma 13 we construct 'FPT-many' instances of Annotated Contraction(vc) such that the original instance is a Yes-instance if and only if at least one of the newly created instances is a Yes-instance. We remark that there is a small technical caveat while using Lemma 13. Consider an instance (G, k, d) of Contraction(vc), and let F be a solution. Lemma 13 implies that there are

subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq |F|$, and (iii) $|Y_s| - |X_s| \leq |F| - d$. However, the statement of Annotated Contraction(vc) specifies the integer k and not the actual size of a minimum solution F. For example, if there exists a solution F of size, say, $k/2$, then Lemma 13 ensures that $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k/2$, however $\mathsf{rank}((X \setminus X_s) \cup Y_s)$ can be smaller than k. To overcome this, we assume that $(G, k-1, d)$ is a No-instance of Contraction(vc). This implies that if there is a subset F of $E(G)$ of size *at most* k such that $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$, then F is of size *exactly* k. We summarize below all the assumptions on the input instance.

▶ **Guarantee 5.1.** *Consider an instance (G, k, d) of* Contraction(vc) *that satisfies the following conditions.*

- *G is a connected graph, $k < \mathsf{rank}(G)$, and $d \leq k$.*
- *A minimum vertex cover X of G is provided as an additional part of the input.*
- *$\mathsf{rank}(X) < d$.*
- *$(G, k-1, d)$ is a* No*-instance of* Contraction(vc).

Unless stated otherwise, we denote the independent set $V(G) \setminus X$ by Y.

Consider an instance (G, k, d) of Contraction(vc) with Guarantee 5.1. We construct $2^{\mathcal{O}(d)}$ many instances of Annotated Contraction(vc) such that (G, k, d) is a Yes-instance if and only if at least one of these newly created instances is a Yes-instance. Informally, let F be the set of edges in a spanning forest of $G[X]$. As $\mathsf{rank}(X) < d$, we have $|F| < d$. We iterate over all 'valid' partitions $\langle X_L, X_R \rangle$ of $V(F)$. We construct an instance of Annotated Contraction(vc) for each such a partition. We formalize this intuition and prove its correctness in the following lemma.

▶ **Lemma 27.** *Suppose that there is an algorithm that solves* Annotated Contraction(vc) *in time $f(n, k, d)$. Then, there exists an algorithm that given as input an instance (G, k, d) of* Contraction(vc) *with Guarantee 5.1, runs in time $3^d \cdot n^{\mathcal{O}(1)} \cdot f(n, k, d)$, and correctly determines whether it is a* Yes*-instance.*

Proof. Let $\mathcal{A}$ be an algorithm that, given an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Annotated Contraction(vc), runs in time $f(n, k, d)$, and correctly determines whether it is a Yes-instance. We describe an algorithm that solves Contraction(vc) using $\mathcal{A}$ as a subroutine.

The algorithm takes as input an instance (G, k, d) of Contraction(vc) and returns either Yes or No. By Guarantee 5.1, the input also consists of a minimum vertex cover X of rank less than d. Let F_x be the edge set of a spanning forest of $G[X]$. For every subset F'_x of F_x, the algorithm constructs multiple instances of Annotated Contraction(vc) as specified in the next paragraph. The algorithm uses Algorithm $\mathcal{A}$ to check if at least one of these newly created instances is a Yes-instance. If it is the case, then the algorithm returns Yes, otherwise it returns No.

Consider a subset F'_x of F_x. Let $\mathcal{P}$ be the collection of partitions $\langle X_{L,F'_x}, X_{R,F'_x} \rangle$ of $V(F_x \setminus F'_x)$ such that for every edge e in $F_x \setminus F'_x$, exactly one of its endpoints is in X_{L,F'_x} and the other one is in X_{R,F'_x}. For every partition $\langle X_{L,F'}, X_{R,F'} \rangle$ in $\mathcal{P}$, the algorithm does as follows: If X_{R,F'_x} is *not* an independent set in G, then the algorithm constructs a trivial No-instance. Otherwise, it adds $(G, k, d, X, \langle X_L, X_R \rangle)$ to the collection of instances of Annotated Contraction(vc). Here, $X_L = X_{L,F'} \cup V(F \setminus F')$ and $X_R = X_{R,F'}$. This completes the description of the algorithm.

We now argue the correctness of the algorithm. Suppose that (G, k, d) is a Yes-instance. Recall that, by Guarantee 5.1, $(G, k-1, d)$ is a No-instance. Hence, there exists a subset $F \subseteq E(G)$ of size exactly k such that $\mathsf{vc}(G/F) \leq \mathsf{vc}(G) - d$. By Lemma 13, there are subsets $X_s \subseteq X$ and $Y_s \subseteq V(G) \setminus X$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii)

$\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq |F|$, and (iii) $|Y_s| - |X_s| \leq |F| - d$. As $|F| = k$, there is a solution pair $\langle X_s, Y_s \rangle$ that satisfies the first three conditions. To see that the solution pair also satisfies the last condition mentioned in the problem statement, let F'_x be the subset of F_x such that $V(F \setminus F'_x) \cap (X \setminus X_s) = \emptyset$. As X_s is an independent set in G, for every edge e in $F_x \setminus F'_x$, exactly one of its endpoints is in X_s. As the algorithm constructs a new instance for every such a partition, at least one of the newly created instances is a Yes-instance.

The algorithm returns Yes only when Algorithm $\mathcal{A}$ returns Yes on one of the newly created instances. By the correctness of Algorithm $\mathcal{A}$, at least one of the newly created instances is a Yes-instance. Hence, there exists sets $X_s \subseteq X$ and $Y_s \subseteq Y$ such that (i) $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, (ii) $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$, and (iii) $|Y_s| - |X_s| \leq k - d$. By Lemma 13, there exists a subset $F \subseteq E(G)$ of size k such that $\mathtt{vc}(G/F) \leq \mathtt{vc}(G) - d$. Hence, (G, k, d) is a Yes-instance. This concludes the proof of correctness of the algorithm.

For every $i \in \{0, 1, 2, \ldots, d\}$, the algorithm iterates over all subsets of edges of size i. It can construct the partition by guessing the right endpoint of the remaining $d - i$ edges. For every partition, it creates an instance and executes Algorithm $\mathcal{A}$. Hence, the total running time of the algorithm is $\mathcal{O}(\sum_{i=0}^{d} \binom{d}{i} \cdot 2^{d-i} \cdot (f(n, d, k) + n^2)) = \mathcal{O}(3^d \cdot (f(n, k, d) + n^2))$. This concludes the proof of the lemma. ◀

As mentioned in the overview of the introduction, to solve an instance of Annotated Contraction($\mathtt{vc}$), we reduce it to an equivalent instance of the Constrained Max-Cut problem. To present such a reduction, it is convenient to work with an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Annotated Contraction($\mathtt{vc}$) where X is an independent set. We present a reduction rule that eliminates edges with both endpoints in X. The reduction rule states that it is safe to contract edges with both endpoints in X_L, and that it is safe to delete edges with one endpoint in X_L and another endpoint in X_R. Recall that if there is an edge with both endpoints in X_R, then the input is a trivial No-instance. Note that $\langle X_L, X_R \rangle$ is not a partition of X. However, as we only need the following reduction rule for the instances obtained by the algorithm mentioned in Lemma 27, we can assume that $X \setminus (X_L \cup X_R)$ is an independent set in G.

▶ **Reduction Rule 5.1.** *Consider an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of* Annotated Contraction($\mathtt{vc}$)*. Let $F_1 = E(X_L, X_R)$ and F_2 be the set of all edges in a spanning forest of $G[X_L]$.*

- *Delete the edges in F_1.*
- *Contract the edges in F_2 and reduce both k and d by $|F_2|$.*

Return the instance $((G', k', d'), X', \langle X'_L, X_R \rangle)$ where $G' = (G - F_1)/F_2$, $k' = k - |F_2|$, $d' = d - |F_2|$, $X' = V(G[X]/F_2)$, and $X'_L = V(G[X_L]/F_2)$.

▶ **Lemma 28.** *Reduction Rule 5.1 is safe. Therefore, it is safe to assume that we are given an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of* Annotated Contraction($\mathtt{vc}$) *such that X is an independent set and a minimum vertex cover of G.*

Proof. As $V(F_2) \subseteq X_L$, for any two subsets X_s, Y_s such that $X_L \cap X_s = X_L \cap Y_s = \emptyset$, we have $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$ if and only if $\mathsf{rank}((X' \setminus X_s) \cup Y_s) \geq k'$. Here, $X' = V(G[X]/F_2)$. Also, by the construction, $k - d = k' - d'$.

($\Rightarrow$) Suppose that $((G, k, d), X, \langle X_L, X_R \rangle)$ is a Yes-instance. Then, there exists a solution pair $\langle X_s, Y_s \rangle$ that satisfies the four conditions mentioned in the definition of the problem. We argue that $\langle X_s, Y_s \rangle$ is also a solution of $(G', k', d', X', \langle X'_L, X_R \rangle)$. As $(X \setminus X_s) \cup Y_s$ is a vertex cover of G, it is also a vertex cover of $G - F_1$. As $(X' \setminus X_s) \cup Y_s$ is obtained from $(X \setminus X_s) \cup Y_s$ by contracting the edges in F_2, whose both endpoints are in $X \setminus X_s$, the set

$(X' \setminus X_s) \cup Y_s$ is a vertex cover of $(G - F_1)/F_2$. As $V(F_2) \subseteq X_L$ and $X_L \cap X_s = X_L \cap Y_s = \emptyset$, $\mathsf{rank}((X \setminus X_s) \cup Y_s) \geq k$ implies that $\mathsf{rank}((X' \setminus X_s) \cup Y_s) \geq k - |F_2| = k'$. Also, by the construction, $k - d = k' - d'$. Hence, $|Y_s| - |X_s| \leq k' - d'$. It is easy to verify that $X'_L \cap X_s = \emptyset$ and $X_R \subseteq X_s$. Hence, $\langle X_s, Y_s \rangle$ satisfies all the four conditions with respect to instance $((G', k', d'), X', \langle X'_L, X_R \rangle)$. This implies that $((G', k', d'), X', \langle X'_L, X_R \rangle)$ is a Yes-instance.

($\Leftarrow$) Suppose that $((G', k', d'), X', \langle X'_L, X_R \rangle)$ is a Yes-instance. Then, there exists a solution pair $\langle X'_s, Y'_s \rangle$ that satisfies the four conditions mentioned in the definition of the problem. Any edge in $G - F_1$ which is not present in G' is incident on some vertex in $V(F_2)$. By the first condition, the set $(X' \setminus X'_s) \cup Y'_s$ is a vertex cover of G'. As $(X' \setminus X_s) \cup Y_s$ is obtained from $(X \setminus X_s) \cup Y_s$ by contracting the edges in F_2 whose both endpoints are in $X \setminus X_s$, $(X \setminus X'_s) \cup Y'_s$ is a vertex cover of $G - F_1$. This implies that if $\mathsf{rank}((X' \setminus X'_s) \cup Y'_s) \geq k'$, then $\mathsf{rank}((X \setminus X'_s) \cup Y'_s) \geq k' + |F_2| = k$. For every edge in F_1, one of its endpoints is incident on $X_L \subseteq X$. Hence, $(X \setminus X'_s) \cup Y'_s$ is a vertex cover of G and its rank is at least k. As $k - d = k' - d'$, we have $|Y'_s| - |X'_s| \leq k - d$. It is easy to verify that $X_L \cap X'_s$ and $X_R \subseteq X'_s$. Hence, $\langle X'_s, Y'_s \rangle$ satisfies all the four conditions with respect to the instance $((G, k, d), X, \langle X_L, X_R \rangle)$. This implies that $((G, k, d), X, \langle X_L, X_R \rangle)$ is a Yes-instance. ◀

5.3.3 Reducing to Constrained MaxCut

We find the following reformulation of Annotated Contraction(vc) convenient to present an algorithm to solve it.

> Constrained MaxCut
> **Input:** An instance (G, k, d) of Contraction(vc), a minimum vertex cover X of G, and a tuple $\langle X_L, X_R \rangle$ such that X_L, X_R are disjoint subsets of X.
> **Question:** Does there exist a partition $\langle V_L, V_R \rangle$ of $V(G)$ such that (i) $E(V_L \cap Y, V_R \cap X) = \emptyset$, (ii) $\mathsf{rank}(E(V_L \cap X, V_R \cap Y)) \geq k$, (iii) $|V_R \cap Y| - |V_R \cap X| \leq k - d$, and (iv) $X_L \subseteq V_L$ and $X_R \subseteq V_R$?

Note that in Annotated Contraction(vc) we are seeking for a pair of subsets, whereas in Constrained MaxCut we are looking for a partition of $V(G)$. Such a formulation allows us to handle vertices that we have decided to keep out of a solution pair. Note that the input instances for both of these problems are the same. Hence, due to Lemma 28, it is safe to assume that X is a minimum vertex cover and an independent set in G. In the next lemma we show that both problems are in fact equivalent.

▶ **Lemma 29.** *An instance $((G, k, d), X, \langle X_L, X_R \rangle)$ is a* Yes*-instance of* Annotated Contraction(vc) *if and only if it is a* Yes*-instance of* Constrained MaxCut.

Proof. ($\Rightarrow$) Let $\langle X_s, Y_s \rangle$ be a solution pair of $((G, k, d), X, \langle X_L, X_R \rangle)$ for the Annotated Contraction(vc) problem. Define $V_R = X_s \cup Y_s$ and $V_L = V(G) \setminus V_R$. It is easy to verify that $\langle V_L, V_R \rangle$ satisfies all the four conditions mentioned in the problem statement of Constrained MaxCut. This implies that $((G, k, d), X, \langle X_L, X_R \rangle)$ is a Yes-instance of Constrained MaxCut.

($\Leftarrow$) Let $\langle V_L, V_R \rangle$ be a desired partition of $V(G)$. Define $X_s = V_R \cap X$ and $Y_s = V_R \cap Y$. Once again, it is easy to verify that the solution pair $\langle X_s, Y_s \rangle$ satisfies all the four conditions mentioned in problem statement of Annotated Contraction(vc). This implies that $((G, k, d), X, \langle X_L, X_R \rangle)$ is a Yes-instance of Annotated Contraction(vc). ◀

Consider an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Constrained MaxCut. We consider the following two cases: (1) $k = d$, and (2) $d < k < 2d$. (Recall that we are in the case

where $k < 2d$.) The first case, as we will see, allows us to impose additional restrictions on the vertices that are in V_R. It also helps us to set up some conditions such that, if they are satisfied while running the algorithm, then it can terminate and safely conclude that the input is a YES-instance. In the second case, even for $k = d + 1$, we do not have these privileges. We deal with each of the two cases separately. Lemma 30 states that if an input instance is of the second type, then we can construct a collection of $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances of the first type such that the input instance is a YES-instance if and only if at least one of these newly created instances is a YES-instance. We remark that this is the only place, in the whole algorithm, where an n^{k-d}-factor appears in the running time. Recall that Theorem 2 implies that this factor is unavoidable. In the next subsection, we present an algorithm to solve the instances that are of the first type.

▶ **Lemma 30.** *Suppose that there is an algorithm that, given an instance* $((G, k, d), X, \langle X_L, X_R\rangle)$ *of* CONSTRAINED MAXCUT *with a guarantee that* $k = d$*, runs in time* $f(n, k, d)$ *and correctly determines whether it is a* YES*-instance. Then, there is an algorithm that solves* CONSTRAINED MAXCUT *in time* $f(n, k, d) \cdot 2^{\mathcal{O}(d)} \cdot n^{k-d+1}$.

Proof. Let $\mathcal{A}$ be an algorithm that, given an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of CONSTRAINED MAXCUT with the guarantee that $k = d$, runs in time $f(n, k, d)$, and correctly determines whether it is a YES-instance. We describe an algorithm that solves *any* instance of CONSTRAINED MAXCUT using $\mathcal{A}$ as a subroutine.

The algorithm takes as input an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of CONSTRAINED MAXCUT and either returns YES or NO. For every input instance, it constructs a collection $\mathcal{I} = \{(G^i, k^i, d^i), X^i, \langle X^i_L, X^i_R\rangle)\}$ of $2^{\mathcal{O}(d)} \cdot n^{k-d}$ many instances, as described below, such that $k^i = d^i$ for every $i \in [|\mathcal{I}|]$. It uses Algorithm $\mathcal{A}$ to check if at least one of these instances is a YES-instance. If it is the case then it returns YES, otherwise it returns NO.

The algorithm constructs the new instances as follows. First it guesses an integer q in $\{0, 1, \ldots, k - d\}$ such that if $((G, k, d), X, \langle X_L, X_R\rangle)$ is a YES-instance then q is the smallest integer for which $((G, d + q, d), X, \langle X_L, X_R\rangle)$ is a YES-instance. Note that the solution size, i.e., the number of edges allowed to be contracted in the second instance, is $d + q$. Let $\mathcal{Y}_q = \{Y^i \subseteq Y \mid |Y^i| = q\}$. For every set Y^i in the collection, the algorithm constructs at most $2^{\mathcal{O}(k)}$ many new instances. If $\mathsf{rank}(E(Y^i, N(Y^i))) \geq k$, then the algorithm constructs a trivial YES-instance. It constructs a graph G^i as follows: For every vertex $y \in Y^i$, it adds a vertex x and makes it adjacent to y. Alternately, every vertex y in Y^i is adjacent to a pendant vertex in G^i. Let X^i_p be the collection of all the pendant vertices added while constructing G^i from G. For every partition $\langle X^i_\ell, X^i_r\rangle$ of $N(Y^i)$, the algorithm constructs a new instance $((G^i, k^i, d^i), X^i, \langle X^i_L, X^i_R\rangle)$ where G^i is as described above, $k^i := d + q$, $d^i := d + q$, $X^i = X \cup X^i_p$, $X^i_L := X_L \cup X^i_\ell$, and $X^i_R := X_R \cup X^i_r \cup X^i_p$. This concludes the description of the algorithm.

We now argue the correctness of the algorithm. Suppose that $((G, k, d), X, \langle X_L, X_R\rangle)$ is a YES-instance. Hence, there exists an integer q in $\{0, 1, \ldots, k - d\}$ such that $((G, d + q, d), X, \langle X_L, X_R\rangle)$ is a YES-instance. We assume, without loss of generality, that q is the smallest such an integer. Consider the case when there exists a subset Y' of Y such that $|Y'| = q$ and $\mathsf{rank}(E(Y', N(Y'))) \geq k$. In this case, the algorithm constructs a trivial YES-instance, and thus correctly concludes that the input is a YES-instance. Now consider the case when for every subset Y' of Y of size q, $\mathsf{rank}(E(Y', N(Y'))) < k$. Suppose that $\langle V_L, V_R\rangle$ is a partition of $V(G)$ that satisfies all the conditions mentioned in the statement of the problem. By the third condition, $|V_R \cap Y| - |V_R \cap X| = q$. Fix a subset Y^i of $V_R \cap Y$ of size q. The third condition ensures that such a set exists. Consider a partition $\langle X^i_\ell, X^i_r\rangle$ of $N(Y^i)$ where $X^i_\ell = N(Y^i) \cap V_L$ and $X^i_r = N(Y^i) \cap V_R$. As the algorithm constructs an

instance for every subset Y' of Y of size q, and for every partition of $N(Y')$, it constructs an instance, say $((G^i, k^i, d^i), X^i, \langle X^i_L, X^i_R\rangle)$, corresponding to Y^i and a partition $\langle X^i_\ell, X^i_r\rangle$ of $N(Y^i)$. We argue that this is a YES-instance.

Let X^i_p be the set of pendant vertices added to G to construct G^i. Note that $|X^i_p| = q$. Hence, $V(G^i) = V(G) \cup X^i_p$. Consider a partition $\langle V^i_L, V^i_R\rangle$ of $V(G^i)$ where $V^i_L = V_L$ and $V^i_R = V_R \cup X^i_p$. It is easy to verify that this partition satisfies all the four conditions mentioned in the problem statement. This implies that $((G^i, k^i, d^i), X^i, \langle X^i_L, X^i_R\rangle)$ is a YES-instance. By the correctness of Algorithm $\mathcal{A}$, it correctly concludes that it is a YES-instance. Hence, in this case the algorithm returns YES on at least one of the newly created instances. Hence, if the input is a YES-instance, then the algorithm correctly concludes that it is a YES-instance.

We now argue that if the algorithm returns YES, then the input instance is indeed a YES-instance. Consider an input instance $((G, k, d), X, \langle X_L, X_R\rangle)$. By the description of the algorithm, it returns YES if and only if one of the newly created instances is a YES-instance. Suppose that one of these instances is a trivial YES-instance. The algorithm constructs such an instance only if it finds a subset Y' of Y which is of size at most $k - d$, and $\mathsf{rank}(N(Y'), Y') \geq k$. In this case, $\langle V_L = V(G) \setminus Y', V_R = Y'\rangle$ satisfies all the conditions mentioned in the problem statement. Hence, the input instance is a YES-instance. Otherwise, suppose that the algorithm concludes that a non-trivial instance $((G^i, k^i, d^i), X^i, \langle X^i_L, X^i_R\rangle)$ is a YES-instance using Algorithm $\mathcal{A}$. Suppose that $\langle V^i_L, V^i_R\rangle$ is a partition of $V(G^i)$ that satisfies all the four conditions in the problem statement. Let X^i_p be the collection of pendant vertices added while constructing G^i from G. Recall that $X^i_p \subseteq X^i_r$ and $|X^i_p| \leq k - d$. It is easy to verify that $\langle V_L = V^i_L, V_R = V^i_R \setminus X^i_p\rangle$ is the desired partition of $V(G)$ that satisfies all the four conditions mentioned in the problem statement. This implies that the input is a YES-instance. This concludes the proof of correctness of the algorithm.

It remains to argue about the running time of the algorithm. For the input instance $((G, k, d), X, \langle X_L, X_R\rangle)$, there are at most $(k - d) + 1$ choices for q. For each q, the size of $\mathcal{Y}_q$, the collection of subsets of Y of size exactly q, is at most n^q. If for a set Y' in $\mathcal{Y}_q$, $\mathsf{rank}(Y', N(Y')) \geq k$, then the algorithm creates only one instance. Otherwise, $\mathsf{rank}(Y', N(Y')) < k$. By Observation 4, the number of vertices in $N(Y')$ is bounded by k. In this case, the algorithm constructs $2^{\mathcal{O}(k)}$ many instances. The overall running time of the algorithm follows from the running time of Algorithm $\mathcal{A}$ and the fact that $k \leq 2d$. ◀

5.3.4 Simplifying an instance of Constrained MaxCut when $k = d$

As mentioned before, in this subsection we present an algorithm to solve an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of CONSTRAINED MAXCUT with a guarantee that $k = d$. We first present a reduction rule to simplify these instances under the presence of a matching saturating X, and prove its correctness using the fact that $k = d$.

▶ **Reduction Rule 5.2.** *Consider an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of* CONSTRAINED MAXCUT *such that $k = d$ and X is an independent set in G. Let M be a matching in G saturating X.*

- *If there exists $x \in X \setminus X_L$ such that $N(x) \setminus V(M) \neq \emptyset$, then add x to X_L.*
- *If there exists $x \in X_L$ such that $N(x) \setminus V(M) \neq \emptyset$, then delete all vertices in $N(x) \setminus V(M)$.*

Return instance $((G', k, d), X, \langle X'_L, X_R\rangle)$ where $G' = G - (N(x) \setminus V(M))$ and $X'_L = X_L \cup \{x\}$.

▶ **Lemma 31.** *Reduction Rule 5.2 is safe.*

Proof. Suppose that $((G, k, d), X, \langle X_L, X_R\rangle)$ is a YES-instance, and let $\langle V_L, V_R\rangle$ be the desired partition of $V(G)$ that satisfies all the conditions in the problem statement. By the

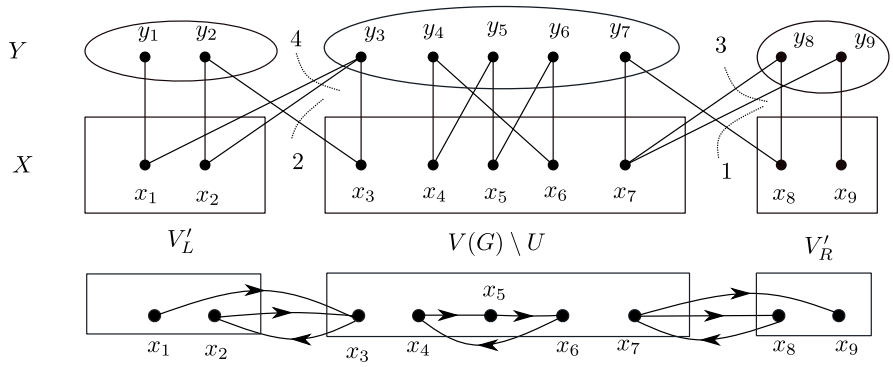

Figure 5 An example to illustration a reduction from Constrained MaxCut to Contrained Directed MaxCut. We do not show all the edges in G for the sake of clarity.

first condition, $N(X \cap V_R) \subseteq Y \cap V_R$. Let M' be the subset of edges of M that saturates all vertices in $X \cap V_R$. As $V(M') \cap Y \subseteq N(X \cap V_R)$, we have $(V(M') \cap Y) \subseteq (Y \cap V_R)$. As M is a matching, $|X \cap V_R| = |M'| = |V(M') \cap Y|$. However, as $k = d$, the third condition implies $|Y \cap V_R| - |X \cap V_R| = 0$. Hence, $(Y \cap V_R) \setminus (V(M') \cap Y)$ is an empty set.

Consider the first case. If $x \in V_R$ then, by the first condition, $N(x) \setminus M$ is in $Y \cap V_R$. However, this contradicts the fact that $(Y \cap V_R) \setminus (V(M') \cap Y)$ is an empty set. This implies that if $((G, k, d), X, \langle X_L, X_R\rangle)$ is a Yes-instance, then $((G', k, d), X, \langle X'_L, X_R\rangle)$ is also a Yes-instance. The reverse direction is vacuously true.

Consider the second case. If $N(x) \setminus V(M)$ is in V_R, this contradicts the fact that $(Y \cap V_R) \setminus (V(M') \cap Y)$ is an empty set. Hence, for any partition $\langle V_L, V_R\rangle$, $N(x) \setminus V(M)$ is in V_L. It is easy to see that $\langle V_L, V_R\rangle$ is a solution of $((G, k, d), X, \langle X_L, X_R\rangle)$ if and only if $\langle V_L \setminus (N(x) \setminus M), V_R\rangle$ is a solution of $((G', k, d), X, \langle X'_L, X_R\rangle)$. ◀

We now present an informal description of the algorithm to solve Constrained MaxCut with a guarantee that $k = d$. Consider an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Constrained MaxCut on which Reduction Rule 5.2 is not applicable. See Figure 5 for an illustration. Let $M = \{x_i y_i \mid i \in [9]\}$ be a matching saturating the vertices in X. Note that, in this case, $|X| = |Y| = |M|$. Consider a subset U of $V(G)$ which can be 'well-partitioned' into $\langle V'_L, V'_R\rangle$. Informally, this means that $\langle V'_L, V'_R\rangle$ can be extended to obtain a partition $\langle V_L, V_R\rangle$ of $V(G)$ such that it is a solution of $((G, k, d), X, \langle X_L, X_R\rangle)$. We can think of the vertices in U as 'processed vertices'. For example, consider $U = \{x_1, y_1, x_2, y_2, x_8, y_8, x_9, y_9\}$ in Figure 5, and let $\langle V'_L, V'_R\rangle$ be a 'well-partition' of U where $V'_L = \{x_1, y_1, x_2, y_2\}$ and $V'_R = \{x_8, y_8, x_9, y_9\}$. Our objective is to extend V'_L, V'_R to obtain V_L, V_R by processing more vertices, i.e., by adding them to either V'_L or V'_R.

As G is connected and X, Y are independent sets in G, at least one of the following four sets is non-empty: (1) $E(Y \setminus U, V'_R \cap X)$, (2) $E(X \setminus U, V'_L \cap Y)$, (3) $E(X \setminus U, V'_R \cap Y)$, and (4) $E(Y \setminus U, V'_L \cap X)$. As we are aiming for a partition $\langle V_L, V_R\rangle$ for which $E(V_L \cap Y, V_R \cap X) = \emptyset$, in the first case it is safe to move the endpoints of the edges in $E(Y \setminus U, V_R \cap X)$ that are in $Y \setminus U$ to V_R. For example, it is safe to move y_7 to V'_R. Similarly, in the second case, it is safe to move the endpoints of edges in $E(X \setminus U, V_L \cap Y)$ that are in $X \setminus U$ to V_L. For example, it is safe to move x_3 to V'_L. As $\langle V_L, V_R\rangle$ also needs to satisfy $|Y \cap V_R| = |X \cap V_R|$, such a move also forces other vertices that are adjacent to these vertices via edges in M to move. For

example, x_7 and y_3 are forced to move to V'_R and V'_L, respectively.

In the third case, if $\mathsf{rank}(E(X \setminus U, V_R \cap Y)) \geq k$ then $\langle V'_L \cup (V(G) \setminus U), V'_R \rangle$ is the desired partition. Otherwise, $\mathsf{rank}(E(X \setminus U, V_R \cap Y)) < k$. It is easy to see that in this case, the number of vertices in $X \setminus U$ that are incident on edges in $E(X \setminus U, V_R \cap Y)$ is at most k. We can guess how the desired partition $\langle V_L, V_R \rangle$ intersects with these endpoints and extend the set of processed vertices in $2^{\mathcal{O}(k)}$ many ways. Similarly, in the fourth case, if $\mathsf{rank}(E(Y \setminus U, V_L \cap X)) \geq k$, then $\langle V'_L, V'_R \cup (V(G) \setminus U) \rangle$ is a partition that satisfies all the desired conditions. Otherwise, we can extend to a set of processed vertices in $2^{\mathcal{O}(k)}$ many ways.

To implement the idea mentioned in the above paragraph, we need some bound on the total number of sets of 'processed vertices' we need to consider. In order to do that, we exploit the properties of the desired partition. Consider the set $\{x_4, y_4, x_5, y_5, x_6, y_6\}$ in Figure 5. Because of the arguments used in the first and the second cases, either this set is entirely contained in V_L or in V_R. To find such cycles, we introduce a directed version of the problem called Constrained Digraph MaxCut. The input of the problem contains a digraph and the objective is to find a partition of the vertex set such that all the arcs across this partition are in the same direction, and the rank of these arcs is at least k. For our case, consider the digraph D obtained from G by directing every edge from X to Y and then 'merging' all edges in the matching M. Recall that M is a matching saturating X. Here, we do not delete parallel or anti-parallel edges while merging an arc in a directed graph. See Figure 5 for an example. In D, sets like these correspond to a directed cycle. And as mentioned before, vertices in these directed cycles move together. Hence, we can obtain a directed acyclic graph by merging these cycles into a vertex. The topological ordering of this resulting graph gives a natural order to process the vertices in G. We formalize these ideas in the next subsection.

5.3.5 Reducing to Constrained Digraph MaxCut

In this section, we consider directed graphs that can have parallel arcs. For a digraph D, we define its *underlying undirected graph* G as the graph obtained from D by forgetting the directions of the arcs. Formally, $V(G) = V(D)$ and $E(G) = \{uv \mid (u, v) \in A(D)\}$. We define the *rank* of a digraph, and the rank of a subset of its vertices or arcs using its underlying undirected graph. Formally, $\mathsf{rank}(D) = \mathsf{rank}(G)$, for a subset $S \subseteq V(D)$, $\mathsf{rank}(S) = \mathsf{rank}(G[S])$, and for a subset $B \subseteq A(D)$, $\mathsf{rank}(B) = \mathsf{rank}(G[V(B)])$.

Constrained Digraph MaxCut
Input: A digraph D, a tuple $\langle X_L, X_R \rangle$ of disjoint subsets of X, and an integer k.
Question: Does there exist a partition (V_L, V_R) of $V(G)$ such that (i) $A(V_R, V_L) = \emptyset$, (ii) $\mathsf{rank}(A(V_L, V_R)) \geq k$, and (iii) $X_L \subseteq V_L$ and $X_R \subseteq V_R$?

We say that a partition $\langle V_L, V_R \rangle$ is a *solution* of $(D, \langle X_L, X_R \rangle, k)$ if it satisfies all the three conditions in the statement of the problem. We present a reduction that, given an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Constrained MaxCut, returns an instance $(D, \langle X_L, X_R \rangle, k)$ of Constrained Directed MaxCut.

The reduction: The reduction takes as input an instance $((G, k, d), X, \langle X_L, X_R \rangle)$ of Constrained MaxCut on which Reduction Rule 5.2 is not applicable. It starts with a copy of the graph G and constructs a digraph D. The reduction finds (in polynomial time) a matching M in G that saturates all vertices in X. For every $xy \in E(G)$, where $x \in X$ and $y \in Y$, it deletes edge xy and adds arc (x, y) (i.e., it directs edges from X to Y). For every arc (x, y) in M, it does as follows: For every in-neighbour x_1 of y, it adds arc (x_1, x). It

then deletes vertex y. This completes the construction of digraph D. The reduction returns $(D, \langle X_L, X_R\rangle, k)$ as the instance of Constrained Digraph MaxCut. This completes the description of the reduction.

▶ **Lemma 32.** *$((G, k, d), X, \langle X_L, X_R\rangle)$ is a* Yes*-instance of* Constrained MaxCut *if and only if $(D, \langle X_L, X_R\rangle, k)$ is a* Yes*-instance of* Constrained Digraph MaxCut*.*

Proof. Note that the edges in $E(V_L \cap Y, V_R \cap X)$ correspond to arcs in $A(V_R \cap X, V_L \cap X)$ and the edges in $E(V_L \cap X, V_R \cap Y)$ correspond to $A(V_L \cap X, V_R \cap X)$. Moreover, by the construction of D, two edges in $E(V_L \cap X, V_R \cap Y)$ share an endpoint if and only if the corresponding two arcs in $A(V_L \cap X, V_R \cap X)$ share an endpoint. This implies that $\mathsf{rank}(E(V_L \cap X, V_R \cap Y)) = \mathsf{rank}(A(V_L \cap X, V_R \cap X))$.

Consider a solution $\langle V_L, V_R\rangle$ of $((G, k, d), X, \langle X_L, X_R\rangle)$. It is easy to see that $\langle V_L \cap X, V_R \cap X\rangle$ is a solution of $(D, \langle X_L, X_R\rangle, k)$.

Similarly, consider a solution $\langle V'_L, V'_R\rangle$ of $(D, \langle X_L, X_R\rangle, k)$. As $V(D) = X$, $\langle V'_L, V'_R\rangle$ is a partition of X. Let V_L be the collection of vertices in V'_L as well as vertices in Y that are adjacent to vertices in V'_L via edges in M. It is easy to verify that $\langle V_L, V_R = V(G) \setminus V_L\rangle$ is a solution of $((G, k, d), X, \langle X_L, X_R\rangle)$. ◀

Consider a digraph D° obtained from D by *merging* a directed cycle C into a single vertex in D. Formally, this operation adds a vertex x_C to $V(D)$, and for every arc (x, x_1) in $A(V(C), V(D) \setminus V(C))$, it adds arc (x_C, x_1) and for every arc (x_1, x) in $A(V(D) \setminus V(C))$, it adds arc (x_1, x_C). Note that, unlike with the edge contraction operation, this operation does not delete parallel or anti-parallel arcs. Consider a map $\psi : V(D) \mapsto V(D^\circ)$ where $\psi(x) = x$ or it is the vertex added to $V(D^\circ)$ while merging a directed cycle containing x. It is easy to verify that ψ defines a surjective function. Consider a directed acyclic graph D' obtained from D by repeatedly merging directed cycles. Let $D = D_1, D_2, \ldots, D_q = D'$ be the sequence of the digraphs such that D_{i+1} is obtained by merging a cycle C_i in D_i, and let $\psi_i : V(D_i) \mapsto V(D_{i+1})$ be the function as defined above. We define $\psi : V(D) \mapsto V(D')$ inductively, i.e., $\psi(x) = \psi_q(\psi_{q-1}(\cdots(\psi_1(x))))$. Once again, it is easy to verify that ψ defines a surjective function. For any $x' \in V(D')$, we define $\psi^{-1}(x') := \{x \in V(D) \mid \psi(x) = x'\}$, and for any subset $U \subseteq V(D')$, $\psi^{-1}(U) := \bigcup_{x' \in U} \psi^{-1}(x')$. A *topological ordering* of a directed acyclic graph D' is a linear ordering $\sigma : V(D') \mapsto [|V(D')|]$ such that for every arc (x, x_1), $\sigma(x) < \sigma(x_1)$.

We are now in position to present an algorithm for Constrained Directed MaxCut.

▶ **Lemma 33.** *There is an algorithm that, given an instance $(G, \langle X_L, X_R\rangle, k)$ of* Constrained Directed MaxCut*, runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a* Yes*-instance.*

Proof. The algorithm takes as input an instance $(D, \langle X_L, X_R\rangle, k)$ of Constrained Directed MaxCut, and returns either Yes or No. It starts by constructing a directed acyclic graph D' from D by merging directed cycles in D as described above. Suppose that D' is the resulting directed acyclic graph obtained after this procedure. The algorithm applies the following two reduction rules exhaustively.

- If there is an arc (x_1, x) such that $x \in X_L$, then delete arc (x_1, x), and add x_1 to X_L.
- If there is an arc (x, x_1) such that $x \in X_R$, then delete arc (x, x_1), and add x_1 to X_R.

The correctness of these reduction rules follows from the fact that, for any solution $\langle V_L, V_R\rangle$ of $(G, \langle X_L, X_R\rangle, k)$, $X_L \subseteq V_L$, $X_R \subseteq V_R$, and $A(V_R, V_L) = \emptyset$.

For notational convenience, we denote the resulting instance obtained after exhaustive application of these reduction rules by $(D', \langle X_L, X_R\rangle, k)$. Note that every vertex in X_L has

in-degree zero and every vertex in X_R has out-degree zero. Let n' be the number of vertices in $V(D')$ and $\sigma = \{x_1, x_2, \ldots, x_{n'}\}$ be a topological ordering of $V(D')$.

We assume, without loss of generality, that the first $|X_L|$ vertices are in X_L, and that the last $|X_R|$ vertices are in X_R. For every $i \in [n']$, define $U^i := \{u_i, u_{i+1}, \ldots, u_{n'}\}$, and $W^i \subseteq \psi^{-1}(U^i)$ as the collection of endpoints of arcs in $A(\psi^{-1}(V(D) \setminus U^i), \psi^{-1}(U^i))$ that are in $\psi^{-1}(U^i)$.

If there is an integer $i \in [n']$, such that $X_L \subseteq \psi^{-1}(V(D) \setminus U^i)$, $X_R \subseteq \psi^{-1}(U^i)$, and the rank of the arcs across the partition $\langle \psi^{-1}(V(D) \setminus U^i), \psi^{-1}(U^i)\rangle$ is at least k, then the algorithm returns Yes. Otherwise, the algorithm constructs a dynamic programming table such that $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ]$ is a 'valid partition' $\langle V^i_L, V^i_R\rangle$ of U^i which is 'consistent' with $\langle W^i_L, W^i_R\rangle$, and has rank at least k°, i.e., $\mathsf{rank}(A(V^i_L, V^i_R)) \geq k^\circ$, if such a partition exists, otherwise it is $\langle \emptyset, \emptyset\rangle$. Here, $i \in [n']$, $\langle W^i_L, W^i_R\rangle$ is a partition of W^i, and $k^\circ \in \{0\} \cup [k]$. The details follow.

▶ **Definition 34.** *We say that a partition $\langle V^i_L, V^i_R\rangle$ of U^i is a* valid partition *if (i) $A(V^i_R, V^i_L) = \emptyset$, and (ii) $(X_L \cap U^i) \subseteq V^i_L$ and $(X_R \cap U^i) \subseteq V^i_R$. We say that a valid partition $\langle V^i_L, V^i_R\rangle$ of U^i is* consistent *with $\langle W^i_L, W^i_R\rangle$ if $W^i_L \subseteq V^i_L$ and $W^i_R \subseteq V^i_R$.*

The algorithm initializes $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ]$ to $\langle \emptyset, \emptyset\rangle$ for every $i \in [n']$, for every partition $\langle W^i_L, W^i_R\rangle$ of W^i, and for every $k^\circ \in \{0\} \cup [k]$. It sets the following values:

- $\mathcal{T}[n', \langle W^n_L = \{u_{n'}\}, W^n_R = \emptyset\rangle, k^\circ = 0]$ to $\langle V^n_L = \{u_{n'}\}, V^n_R = \emptyset\rangle$, and
- $\mathcal{T}[n', \langle W^n_L = \emptyset, W^n_R = \{u_{n'}\}\rangle, k^\circ = 0]$ to $\langle V^n_L = \emptyset, V^n_R = \{u_{n'}\}\rangle$.

To compute $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ]$, the algorithm considers the following three cases: (1) $u_i \in W^i_L$, (2) $u_i \in W^i_R$, and (3) $u_i \notin W^i_L \cup W^i_R$.

1. In the first case, if there exists a table entry $(i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k')$ for some $k' \in \{0\} \cup [k]$ such that
   - $W^i_L \setminus \{u_i\} \subseteq W^{i+1}_L$ and $W^i_R \subseteq W^{i+1}_R$, and
   - $\mathsf{rank}(A(V^{i+1}_L \cup \{u_i\}, V^{i+1}_R)) \geq k^\circ$,

   where $\langle V^{i+1}_L, V^{i+1}_R\rangle = \mathcal{T}[i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k']$, then the algorithm sets $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ] = \langle V^{i+1}_L \cup \{u_i\}, V^{i+1}_R\rangle$.
2. In the second case, if there exists a table entry $(i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k')$ for some $k' \in \{0\} \cup [k]$ such that
   - $W^i_L \subseteq W^{i+1}_L$ and $W^i_R \setminus \{u_i\} \subseteq W^{i+1}_R$,
   - $N_{\mathsf{out}}(u_i) \cap W^{i+1}_L = \emptyset$, and
   - $\mathsf{rank}(A(V^{i+1}_L, V^{i+1}_R \cup \{u_i\})) \geq k^\circ$,

   where $\langle V^{i+1}_L, V^{i+1}_R\rangle = \mathcal{T}[i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k']$, then the algorithm sets $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ] = \langle V^{i+1}_L, V^{i+1}_R \cup \{u_i\}\rangle$.
3. In the third case, if there exists a table entry $(i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k')$ for some $k' \in \{0\} \cup [k]$ such that
   - $W^i_L \subseteq W^{i+1}_L$ and $W^i_R \subseteq W^{i+1}_R$, and
   - $\mathsf{rank}(A(V^{i+1}_L \cup \{u_i\}, V^{i+1}_R)) \geq k^\circ$,

   where $\langle V^{i+1}_L, V^{i+1}_R\rangle = \mathcal{T}[i+1, \langle W^{i+1}_L, W^{i+1}_R\rangle, k']$, then the algorithm sets $\mathcal{T}[i, \langle W^i_L, W^i_R\rangle, k^\circ] = \langle V^{i+1}_L \cup \{u_i\}, V^{i+1}_R\rangle$.

If $\mathcal{T}[1, \langle W_L^1, W_R^1\rangle, k]$ is not $\langle \emptyset, \emptyset\rangle$ for some partition $\langle W_L^1, W_R^1\rangle$ of W^1, then the algorithm returns Yes, otherwise it returns No. This concludes the description of the algorithm.

We now argue that if the algorithm computes the above dynamic programming table, then all the entries are correct. It is easy to verify that all the entries corresponding to indices where $i = n'$ are correct. Suppose this is true for every integer in $[n']$ which is greater than i. Consider an index $(i, \langle W_L^i, W_R^i\rangle, k^\circ)$ and suppose that there exists a valid partition $\langle V_L^i, V_R^i\rangle$ of U^i which is consistent with $\langle W_L^i, W_R^i\rangle$. By the definition, $W_L^i \setminus \{u_i\} \subseteq W_L^{i+1}$ and $W_R^i \setminus \{u_i\} \subseteq W_R^{i+1}$. We consider the following three exhaustive cases.

- If $u_i \in W_L^i$ (which implies $u_i \in V_L^i$) , then $\langle V_L^i \setminus \{u_i\}, V_R^i\rangle$ is a valid partition of U^{i+1} which is consistent with some partition $\langle W_L^{i+1}, W_R^{i+1}\rangle$ of W^{i+1}, such that $W_L^i \setminus \{u_i\} \subseteq W_L^{i+1}$ and $W_R^i \subseteq W_R^{i+1}$.
- If $u_i \in W_R^i$ (which implies $u_i \in V_R^i$), then $\langle V_L^i, V_R^i \setminus \{u_i\}\rangle$ is a valid partition of U^{i+1} which is consistent with some partition $\langle W_L^{i+1}, W_R^{i+1}\rangle$ of W^{i+1}, such that $W_L^i \subseteq W_L^{i+1}$ and $W_R^i \setminus \{u_i\} \subseteq W_R^{i+1}$. Note that, as $N_{\mathsf{out}}(u_i) \subseteq W_L^{i+1} \cup W_R^{i+1}$, we have $N_{\mathsf{out}}(u_i) \cap W_L^{i+1} = \emptyset$.
- If $u_i \notin W_L^i \cup W_R^i$, then there is no arc whose endpoint is u_i. In this case, $\langle V_L^i \setminus \{u_i\}, V_R^i\rangle$ is a valid partition of U^{i+1} which is consistent with some partition $\langle W_L^{i+1}, W_R^{i+1}\rangle$ of W^{i+1}, such that $W_L^i \subseteq W_L^{i+1}$ and $W_R^i \setminus \{u_i\} \subseteq W_R^{i+1}$.

As the algorithm correctly computes these values for every integer greater than i, it correctly computes the value of $\mathcal{T}[i, \langle W_L^i, W_R^i\rangle, k^\circ]$.

We now argue about the correctness of the algorithm. Suppose that the input instance $(D, \langle X_L, X_R\rangle, k)$ is a Yes-instance. If there exists a partition $\langle V_L, V_R\rangle$ of $V(D)$ which is a solution, such that $V_R = \psi^{-1}(U_i)$ for some $i \in [n']$, then the rank of the arcs across the partition $\langle \psi^{-1}(V(D) \setminus U^i), \psi^{-1}(U^i)\rangle$ is at least k. In this case, the algorithm correctly concludes that the input is a Yes-instance. Otherwise, it computes the table as described above. By the description of the algorithm, $U^1 = V(D')$, and the fact that all the entries in the table are correct, in this case the algorithm correctly concludes that the input is a Yes-instance.

Suppose that the algorithm returns Yes on the input instance $(D, \langle X_L, X_R\rangle, k)$. Consider the case when the algorithm returns Yes without constructing the table. Note that every arc across the partition $\langle V(D') \setminus U^i, U^i\rangle$ has its startpoint in $V(D') \setminus U^i$ and endpoint in U^i. Alternately, $A(U^i, V(D') \setminus U^i) = \emptyset$. By the construction of D', $A(\psi^{-1}(U^i), V(D) \setminus \psi^{-1}(U^i)) = \emptyset$ for every $i \in [n']$. As there exists $i \in [n']$ such that $X_L \subseteq \psi^{-1}(V(D) \setminus U^i)$, $X_R \subseteq \psi^{-1}(U^i)$, and the rank of the arcs across the partition $\langle \psi^{-1}(V(D) \setminus U^i), \psi^{-1}(U^i)\rangle$ is at least k, $\langle V(D) \setminus U^i), \psi^{-1}(U^i)\rangle$ is a solution of $(G, \langle X_L, X_R\rangle, k)$. In the other case, when the table is constructed, as every entry in the table is correct, the algorithm correctly concludes that the input is a Yes-instance. This completes the proof of correctness of the algorithm.

To argue about the running time of the algorithm, notice that all the other steps, apart from computing the table, can be performed in polynomial time. The algorithm computes the table if and only if for every integer $i \in [n']$, the rank of arcs across the partition $\langle \psi^{-1}(V(D) \setminus U^i), \psi^{-1}(U^i)\rangle$ is less than k. By arguments similar to those of Observation 4, it is easy to see that $|W^i| < k$ for all $i \in [n']$. Hence, the number of entries in the table is $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. The algorithm takes $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time to compute each table entry. This implies that the algorithm terminates in the desired time, and the proof of the lemma is complete. ◀

5.3.6 Proof of Lemma 25

Lemma 33 implies that there is an algorithm, say $\mathcal{A}$, that given an instance $(G, \langle X_L, X_R\rangle, k)$ of Constrained Directed MaxCut, runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, and correctly determines whether it is a Yes-instance.

Consider an algorithm, say $\mathcal{B}$, that takes as input an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Constrained MaxCut with the extra guarantee that $k = d$, and returns Yes or No. It starts by exhaustively applying Reduction Rule 5.2. It then uses the reduction mentioned in Subsubsection 5.3.5 to obtain an instance $(D, \langle X_L, X_R\rangle, k)$ of Constrained Directed MaxCut. Using Algorithm $\mathcal{A}$, it determines whether the constructed instance is a Yes-instance. If it is indeed the case, then it returns Yes, otherwise it returns No. This concludes the description of Algorithm $\mathcal{B}$.

The correctness of Algorithm $\mathcal{B}$ follows from Lemma 31, Lemma 32, and the correctness of Algorithm $\mathcal{A}$. Its running time follows from the running time of Algorithm $\mathcal{A}$. Hence, Algorithm $\mathcal{B}$ takes as input an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Constrained MaxCut with a guarantee that $k = d$, runs in time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$, and correctly determines whether it is a Yes-instance.

Algorithm $\mathcal{B}$ and Lemma 30 imply that there is an algorithm, say $\mathcal{C}$, that given an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Constrained MaxCut (without any guarantee), runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$, and correctly determines whether it is a Yes-instance. Recall that since we are in the case $d \leq k < 2d$, it holds that $2^{\mathcal{O}(k)} = 2^{\mathcal{O}(d)}$.

Consider an algorithm, say $\mathcal{D}$, that takes as input an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Annotated Contraction(vc), and returns Yes or No. It exhaustively applies Reduction Rule 5.1. It then uses Algorithm $\mathcal{C}$ to determine whether the resulting instance is a Yes-instance of Constrained MaxCut. If it is indeed the case, it returns Yes, otherwise it returns No. This concludes the description of Algorithm $\mathcal{D}$.

The correctness of Algorithm $\mathcal{D}$ follows from Lemma 28, Lemma 29, and the correctness of Algorithm $\mathcal{C}$. Its running time follows from the running time of Algorithm $\mathcal{C}$. Hence, Algorithm $\mathcal{D}$ takes as input an instance $((G, k, d), X, \langle X_L, X_R\rangle)$ of Annotated Contraction(vc), runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$ and correctly decides whether it is a Yes-instance.

Algorithm $\mathcal{D}$ and Lemma 27 imply that there is an algorithm, say $\mathcal{E}$, that given an instance (G, k, d) of Contraction(vc) with Guarantee 5.1, runs in time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(k-d)}$ and correctly decides whether it is a Yes-instance.

Consider an algorithm, say $\mathcal{F}$, that takes as input an instance (G, k, d) of Contraction(vc) with the guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, and returns Yes or No. It first applies the algorithm mentioned in Lemma 26 as a subroutine. If the subroutine concludes that (G, k, d) is a Yes-instance, then it returns Yes. Otherwise, let X be the minimum vertex cover of G returned by the subroutine. Recall that $\mathsf{rank}(X) < d$. The algorithm constructs instances (G, k°, d) for every value of $k^\circ \in [k]$. In the increasing order of k°, it determines whether (G, k°, d) is a Yes-instance using Algorithm $\mathcal{E}$. If for any value of k°, Algorithm $\mathcal{E}$ concludes that (G, k°, d) is a Yes-instance, then the algorithm returns Yes, otherwise it returns No. This concludes the description of Algorithm $\mathcal{F}$.

Note that for $k^\circ = 1$, instance (G, k°, d) satisfies every condition mentioned in Guarantee 5.1. The algorithm uses Algorithm $\mathcal{E}$ with (G, k°, d) as input only if Algorithm $\mathcal{E}$ concludes that $(G, k^\circ - 1, d)$ is a No-instance. The correctness and the running time of Algorithm $\mathcal{F}$ follow from those of Algorithm $\mathcal{E}$. This implies that Algorithm $\mathcal{F}$ receives an instance (G, k, d) of Contraction(vc) with guarantees that $k < \mathsf{rank}(G)$ and $d \leq k < 2d$, runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$, and correctly determines whether it is a Yes-instance. This concludes the proof of Lemma 25.

5.4 Proof of Theorem 3

Consider an instance (G, k, d) of Contraction(vc). If G is a connected graph, we can use the algorithm mentioned at the start of Section 5 to conclude whether (G, k, d) is a Yes-instance. Otherwise, for each each connected component C_i of G, integers $k_i \in [k]$, and $d_i \in [d]$, we use the algorithm to determine whether $(G[C_i], k_i, d_i)$ is a Yes-instance of Contraction(vc). With this information, one can construct a standard Knapsack-type dynamic programming table, which is also mentioned in [34], to determine whether (G, k, d) is a Yes-instance.

The correctness and the running time of this procedure follows from those of Lemma 23, Lemma 24, and Lemma 25. Hence, there is an algorithm that takes as input an instance (G, k, d) of Contraction(vc), runs in time $2^{\mathcal{O}(d)} \cdot n^{k-d+\mathcal{O}(1)}$ and correctly determines whether it is a Yes-instance.

6 Conclusion

In this article we considered the problem of reducing the size of a minimum vertex cover of a graph G by at least d using at most k edge contractions. Note that the problem is trivial when $k < d$. A few simple observations prove that when $d \leq 2k$, the problem is coNP-hard and FPT when parameterized by $k + d$. Almost all of our technical work is to handle the case when $d \leq k < 2d$. We proved that the problem is NP-hard when $k = d + \frac{\ell-1}{\ell+3} \cdot d$ for any integer $\ell \geq 1$ such that k is an integer (in particular, $\ell = 1$). This implies that the problem is hard for various values of $k - d$ in the set $\{0, 1, \ldots, d-1\}$. We were able to prove that if $(k - d)$ is a constant then the problem is FPT when parameterized by $k + d$. However, if no such a condition is imposed, then the problem is W[1]-hard. More precisely, we presented an algorithm with running time $2^d \cdot n^{k-d+\mathcal{O}(1)}$ and proved that the problem is W[1]-hard when parameterized by $k + d$ in the case where $k - d = \frac{d}{3}$ (see the proof of Theorem 2).

We believe that it should be possible to prove that the problem is NP-hard for *every* value of $k - d$ in the set $\{0, 1, \ldots, d-1\}$. Such a reduction has the potential to sharpen the distinction between FPT and W[1]-hard cases as $k - d$ varies in this range. It might also simplify the analysis of our XP algorithm or lead to a simpler algorithm. It would be interesting to analyze the parameterized complexity of the problem with respect to structural parameters like the vertex cover number or the treewidth of the input graph. Note that the problem is trivially FPT when parameterized by the vertex cover number. Finally, it is worth mentioning that we did not focus on optimizing the degree of the polynomial term $n^{\mathcal{O}(1)}$ in our XP algorithm, although it is reasonably small.

References

1 Akanksha Agarwal, Saket Saurabh, and Prafullkumar Tale. On the parameterized complexity of contraction to generalization of trees. *Theory of Computing Systems*, 63(3):587–614, 2019. doi:https://doi.org/10.1007/s00224-018-9892-z.

2 Akanksha Agrawal, Lawqueen Kanesh, Saket Saurabh, and Prafullkumar Tale. Paths to trees and cacti. *Theoretical Computer Science*, 860:98–116, 2021. doi:https://doi.org/10.1016/j.tcs.2021.01.033.

3 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. *ACM Transactions on Computation Theory*, 11(3):1–22, 2019. doi:https://doi.org/10.1145/3319909.

4 Cristina Bazgan, Cédric Bentz, Christophe Picouleau, and Bernard Ries. Blockers for the stability number and the chromatic number. *Graphs and Combinatorics*, 31(1):73–90, 2015. doi:https://doi.org/10.1007/s00373-013-1380-2.

5 Cristina Bazgan, Sonia Toubaline, and Zsolt Tuza. The most vital nodes with respect to independent set and vertex cover. *Discrete Applied Mathematics*, 159:1933–1946, 2011. doi:https://doi.org/10.1016/j.dam.2011.06.023.

6 Rémy Belmonte, Petr A. Golovach, Pim Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014. doi:https://doi.org/10.1007/s00236-014-0204-z.

7 Cédric Bentz, Costa Marie-Christine, Dominique de Werra, Christophe Picouleau, and Bernard Ries. Blockers and transversals in some subclasses of bipartite graphs: when caterpillars are dancing on a grid. *Discrete Mathematics*, 310:132–146, 2010. doi:https://doi.org/10.1016/j.disc.2009.08.009.

8 Hans L. Bodlaender, Pinar Heggernes, and Daniel Lokshtanov. Graph Modification Problems (Dagstuhl Seminar 14071). *Dagstuhl Reports*, 4(2):38–59, 2014. doi:10.4230/DagRep.4.2.38.

9 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *Proc. of the 8th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 8246 of *LNCS*, pages 97–109, 2013. doi:10.1007/978-3-319-03898-8_10.

10 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:https://doi.org/10.1016/j.tcs.2010.06.026.

11 Marie-Christine Costa, Dominique de Werra, and Christophe Picouleau. Minimum d-blockers and d-transversals in graphs. *Journal of Combinatorial Optimization*, 22(4):857–872, 2011. doi:10.1007/s10878-010-9334-6.

12 Bruno Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

13 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification, 2020. arXiv:2001.06867.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

15 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. URL: https://dblp.org/rec/books/daglib/0030488.bib.

16 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction and deletion blockers for perfect graphs and H-free graphs. *Theoretical Computer Science*, 746:49–72, 2018. doi:https://doi.org/10.1016/j.tcs.2018.06.023.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

18 Esther Galby, Paloma T. Lima, Felix Mann, and Bernard Ries. Using edge contractions to reduce the semitotal domination number. *Theoretical Computer Science*, 939:140–160, 2023. doi:https://doi.org/10.1016/j.tcs.2022.10.020.

19 Esther Galby, Paloma T. Lima, and Bernard Ries. Reducing the domination number of graphs via edge contractions and vertex deletions. *Discrete Mathematics*, 344(1):112169, 2021. doi:10.1016/j.disc.2020.112169.

20 Esther Galby, Felix Mann, and Bernard Ries. Blocking total dominating sets via edge contractions. *Theoretical Computer Science*, 877:18–35, 2021. doi:https://doi.org/10.1016/j.tcs.2021.03.028.

21 Esther Galby, Felix Mann, and Bernard Ries. Reducing the domination number of (P_3+kP_2)-free graphs via one edge contraction. *Discrete Applied Mathematics*, 305:205–210, 2021. doi:https://doi.org/10.1016/j.dam.2021.09.009.

22 Petr A. Golovach, Marcin Kaminski, Daniël Paulusma, and Dimitrios M Thilikos. Increasing the minimum degree of a graph by contractions. *Theoretical Computer Science*, 481:74–84, 2013. doi:https://doi.org/10.1016/j.tcs.2013.02.030.

23 Petr A. Golovach, Pim van't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013. doi:https://doi.org/10.1016/j.tcs.2012.12.041.

24 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for Bipartite Contraction. *Information Processing Letters*, 113(22):906–912, 2013. doi:10.1016/j.ipl.2013.09.004.

25 Philip Hall. On Representatives of Subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 01 1935. doi:10.1112/jlms/s1-10.37.26.

26 Pinar Heggernes, Pim van 't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014. doi:10.1007/s00453-012-9670-2.

27 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013. doi:10.1137/130907392.

28 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

29 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

30 Eun Jung Kim, Martin Milanič, Jérôme Monnot, and Christophe Picouleau. Complexity and algorithms for constant diameter augmentation problems. *Theoretical Computer Science*, 2021. doi:https://doi.org/10.1016/j.tcs.2021.05.020.

31 Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson, 2006.

32 R. Krithika, Pranabendu Misra, and Prafullkumar Tale. An FPT algorithm for contraction to cactus. In *Proc. of the 24th International Computing and Combinatorics Conference (COCOON)*, volume 10976 of *LNCS*, pages 341–352, 2018. doi:https://doi.org/10.1007/978-3-319-94776-1_29.

33 John M. Lewis and Mihalis Yannakakis. The Node-Deletion Problem for Hereditary Properties is NP-Complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

34 Paloma T. Lima, Vinícius Fernandes dos Santos, Ignasi Sau, and Uéverton S. Souza. Reducing graph transversals via edge contractions. *Journal of Computer and System Sciences*, 120:62–74, 2021. doi:https://doi.org/10.1016/j.jcss.2021.03.003.

35 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *Proc. of the 8th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 8246 of *LNCS*, pages 243–254, 2013. doi:https://doi.org/10.1007/978-3-319-03898-8_21.

36 Foad Mahdavi Pajouh, Vladimir Boginski, and Eduardo Pasiliao. Minimum vertex blocker clique problem. *Networks*, 64:48–64, 2014. doi:10.1002/net.21556.

37 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and $2K_2$-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015. doi:https://doi.org/10.1016/j.jctb.2014.09.002.

38 N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In *Proc. of the 29th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14 of *LIPIcs*, pages 338–349, 2012. doi:10.4230/LIPIcs.STACS.2012.338.

39 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Critical vertices and edges in H-free graphs. *Discrete Applied Mathematics*, 257:361–367, 2019. doi:10.1016/j.dam.2018.08.016.

40 Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale. On the parameterized complexity of grid contraction. In *Proc. of the 17th Scandinavian Symposium and Workshops*

on Algorithm Theory (SWAT), volume 162 of *LIPIcs*, pages 34:1–34:17, 2020. doi:10.4230/LIPIcs.SWAT.2020.34.

41 Saket Saurabh and Prafullkumar Tale. On the parameterized complexity of maximum degree contraction problem. In *Proc. of the 15th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 180 of *LIPIcs*, pages 26:1–26:16, 2020. doi:10.4230/LIPIcs.IPEC.2020.26.

42 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983. doi:10.1016/0166-218X(83)90101-4.

43 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978. doi:10.1145/800133.804355.