Kernelization for Counting Problems on Graphs: Preserving the Number of Minimum Solutions

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Bart van der Steenhoven
Eindhoven University of Technology, The Netherlands

Abstract

A kernelization for a parameterized decision problem $\mathcal{Q}$ is a polynomial-time preprocessing algorithm that reduces any parameterized instance (x, k) into an instance (x', k') whose size is bounded by a function of k alone and which has the same YES/NO answer for $\mathcal{Q}$. Such preprocessing algorithms cannot exist in the context of counting problems, when the answer to be preserved is the number of solutions, since this number can be arbitrarily large compared to k. However, we show that for counting minimum feedback vertex sets of size at most k, and for counting minimum dominating sets of size at most k in a planar graph, there is a polynomial-time algorithm that either outputs the answer or reduces to an instance (G', k') of size polynomial in k with the same number of minimum solutions. This shows that a meaningful theory of kernelization for counting problems is possible and opens the door for future developments. Our algorithms exploit that if the number of solutions exceeds $2^{\text{poly}(k)}$, the size of the input is exponential in terms of k so that the running time of a parameterized counting algorithm can be bounded by $\text{poly}(n)$. Otherwise, we can use gadgets that slightly increase k to represent choices among $2^{\mathcal{O}(k)}$ options by only $\text{poly}(k)$ vertices.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases kernelization, counting problems, feedback vertex set, dominating set, protrusion decomposition

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.27

Related Version *Full Version*: `http://arxiv.org/abs/2310.04303` [16]

Funding *Bart M. P. Jansen*: Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).
Bart van der Steenhoven: Supported by Project No. ICT22-029 of the Vienna Science Foundation (WWTF).

1 Introduction

Background and motivation. Counting problems, whose answer is an integer giving the number of objects of a certain kind rather than merely YES or NO, have important applications in fields of research such as artificial intelligence [24, 25], statistical physics [17, 20, 31] and network science [22]. They have been studied extensively in classical complexity, underpinning fundamental results such as Toda's theorem [29] and the #P-completeness of the permanent [30]. A substantial research effort has targeted the parameterized complexity of counting problems, leading to parametric complexity-notions like #W[1]-hardness [7, 11, 21] and FPT algorithms to solve several counting problems. For example, FPT algorithms were developed to count the number of size-k vertex covers [10], or the number of occurrences of a size-k pattern graph H in a host graph G [8].

This paper is concerned with an aspect of parameterized algorithms which has been largely neglected for counting problems: that of efficient preprocessing with performance guarantees, i.e., kernelization. A kernelization for a parameterized decision problem $\mathcal{Q}$ is a polynomial-time preprocessing algorithm that reduces any parameterized instance (x, k) into an instance (x', k') whose size is bounded by a function of k alone and which has the same YES/NO answer for $\mathcal{Q}$. Over the last decade, kernelization has developed into an important subfield of parameterized algorithms, as documented in a textbook dedicated to the topic [12]. Given the success of kernelization for decision problems, one may wonder: can a theory of provably-efficient preprocessing for counting problems be developed?

Consider a prototypical problem such as Feedback Vertex Set in undirected graphs, in which the goal is to find a small vertex set whose removal breaks all cycles. What could be an appropriate notion of counting kernelization for such a problem? The concept of efficient preprocessing towards a provably small instance with the same answer could be instantiated as follows: given a pair (G, k), the preprocessing algorithm should output a pair (G', k') whose size is bounded by a function of k such that the number of size-k feedback vertex sets in G is equal to the number of size-k' feedback vertex sets in G'. However, this task is clearly impossible. Given a graph consisting of a length-n cycle with parameter $k = 1$, the number of solutions is n which can be arbitrarily large compared to k, while for any reduced instance (G', k') of size bounded in k, the number of solutions can be at most $2^{|V(G')|} \leq f(k)$. Without allowing the size of the reduced instance to depend on n, it seems that preprocessing while preserving the answer to the counting problem is impossible.

Over the years, there have been two approaches to deal with this obstacle[1]. Thurley [28], inspired by concepts in earlier work [23], proposed a notion of counting kernelization which effectively reduces a counting problem to an enumeration problem. He considered problems such as Vertex Cover and d-Hitting Set. In his framework, the preprocessing algorithm has to output an instance of size bounded by a function of k, in such a way that for any solution to the reduced instance, we can efficiently determine to how many solutions of the original instance it corresponds. Hence by enumerating *all* solutions on the reduced instance, we can obtain the number of solutions to the original instance. A significant drawback of this approach therefore lies in the fact that to solve the counting problem on the original instance, we have to enumerate all solutions on the reduced instance. Since counting can potentially be done much faster than enumeration, it is not clear that this preprocessing step is always beneficial.

A second notion for counting kernelization was proposed by Kim, Selma, and Thilikos [18, 26]. Their framework (which also applies to Feedback Vertex Set) considers two types of algorithms: a *condenser* that maps an input instance (G, k) to an instance (G', k') of an auxiliary *annotated* problem involving weights on the vertices of G', and an *extractor* that recovers (typically not in polynomial time) the number of solutions to (G, k) from the weighted instance (G', k'). The number of vertices of G' is required to be bounded in k, but the weights are allowed to be arbitrarily large, thereby sidestepping the issue described above. This means that in terms of the total encoding size, the weighted graph (G', k') is not guaranteed to be smaller than (G, k) and in general the total number of bits needed to encode the weighted graph cannot be bounded by a function of k alone. The condenser-extractor

[1] A third [19] approach was announced shortly before this paper went to print; it allows a polynomial-time lifting step to compute the number of solutions to the original instance from the number of solutions to the reduced instance.

framework has the same drawback as the framework by Thurley: a standard counting problem is reduced to a more complicated type of problem, in this case one involving weights and annotations.

The goal of this paper is to show that there is an alternative way to overcome the obstacle for counting kernelization, which leads to a notion of preprocessing in which the problem to be solved on the reduced instance is of exactly the same nature as the original. Our solution is inspired by the typical behavior of kernelization algorithms for decision problems: we formalize the option of already finding the answer during the preprocessing phase. Note that many algorithms, such as the famous Buss [5] kernelization for Vertex Cover, work by applying reduction rules to arrive at the reduced instance, or discover the yes/no answer to the decision problem during preprocessing. Our kernelization algorithms for counting problems will have the same behavior: they will either reduce to a poly(k)-sized instance of the same problem that has exactly the same answer to the counting problem, or they outright answer the counting problem during their polynomial-time computation. To our initial surprise, such preprocessing algorithms exist for several classic problems.

Our results. To begin the exploration of this new type of counting kernelization, we revisit two prominent graph problems: Feedback Vertex Set in general undirected graphs and Dominating Set in planar graphs. The decision versions of these problems (does graph G have a solution of size at most k?) have kernels with $\mathcal{O}(k^2)$ [15, 27] and $\mathcal{O}(k)$ vertices [1, 4, 14], respectively. We consider the problem of counting the number of *minimum-size* solutions, parameterized by the size k of a minimum solution. (We discuss counting inclusion-minimal solutions in the conclusion.) For a graph G and integer k, we denote by $\#\text{minFVS}(G,k)$ the number of minimum feedback vertex sets in G of size at most k in G. Hence $\#\text{minFVS}(G,k)$ is equal to 0 if the feedback vertex number of G exceeds k, and otherwise is equal to the number of minimum solutions. The analogous concept for minimum dominating sets is denoted $\#\text{minDS}(G,k)$. Our result for Feedback Vertex Set reads as follows.

▶ **Theorem 1.1.** *There is a polynomial-time algorithm that, given a graph G and integer k, either*

- *outputs $\#\text{minFVS}(G,k)$, or*
- *outputs a graph G' and integer k' such that $\#\text{minFVS}(G,k) = \#\text{minFVS}(G',k')$ and $|V(G')| = \mathcal{O}(k^5)$ and $k' = \mathcal{O}(k^5)$.*

For Dominating Set on planar graphs, we give an analogous algorithm that either outputs $\#\text{minDS}(G,k)$ or reduces to a *planar* instance (G',k') with $|V(G')|, k' = \mathcal{O}(k^3)$ such that $\#\text{minDS}(G,k) = \#\text{minDS}(G',k')$. Hence if the parameter is small, the task of counting the number of minimum solutions can efficiently be reduced to the *same* counting task on a provably small instance.

▶ **Theorem 1.2 (★).** *There is a polynomial-time algorithm that, given a planar graph G and integer k, either*

- *outputs $\#\text{minDS}(G,k)$, or*
- *outputs a planar graph G' and integer k' such that $\#\text{minDS}(G,k) = \#\text{minDS}(G',k')$ and $|V(G')| = \mathcal{O}(k^3)$ and $k' = \mathcal{O}(k^3)$.*

The high-level approach is the same for both problems. We use insights from existing kernels for the decision version of the problem to reduce an input instance (G,k) into one (G',k') with the same number of minimum solutions, such that G' can be decomposed into a "small" core together with poly(k) "simply structured but potentially large" parts. For

Dominating Set, this takes the form of a protrusion decomposition; for Feedback Vertex Set the decomposition is more elementary. Then we consider two cases. If $|V(G)| > 2^k$, we employ an FPT algorithm running in time $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ to count the number of minimum solutions and output it. Since $n > 2^k$, this step runs in polynomial time. If $|V(G)| \leq 2^k$, then we show that each of the poly(k) simply structured parts can be replaced with a gadget of size poly(k) without affecting the number of minimum solutions. In this step, we typically increase the size of minimum solutions slightly to allow a small vertex set to encode exponentially many potential solutions. For example, an instance of Feedback Vertex Set consisting of a cycle of length 2^{10} (which has 2^{10} different optimal solutions), can be reduced to the graph consisting of 10 pairs (a_i, b_i), each pair connected by two parallel edges. The latter graph also has 2^{10} minimum solutions, each of size 10. To carry out this approach, the most technical part is to show how to decompose the input instance into parts in which it is easy to analyze how many different choices an optimal solution can make.

Organization. The remainder of the paper is structured as follows. After presenting preliminaries in Section 2, we illustrate our approach for Feedback Vertex Set in Section 3. The more technical application to Dominating Set on planar graphs is deferred to the full version [16] due to space limitations. We conclude in Section 4 with a reflection on the potential of this approach to counting kernelization.

2 Preliminaries

All graphs we consider are undirected; they may have parallel edges but no self-loops. A graph G therefore consists of a set $V(G)$ of vertices and a multiset $E(G)$ of edges of the form $\{u, v\}$ for distinct $u, v \in V(G)$. For a vertex $v \in V(G)$, we refer to the *open neighborhood* of v in G as $N_G(v)$ and to the *closed neighborhood* of v as $N_G[v]$. For a set of vertices $X \subseteq V(G)$, the open and closed neighborhoods are defined as $N_G(X) = (\bigcup_{v \in X} N_G(v)) \setminus X$ and $N_G[X] = \bigcup_{v \in X} N_G[v]$. The degree of vertex v in graph G, denoted by $\deg_G(v)$, is equal to the number of edges incident to v in G. We refer to the subgraph of G induced by a vertex set $X \subseteq V(G)$ as $G[X]$. We use $G - X$ as a way to write $G[V(G) \setminus X]$ and $G - v$ as a shorthand for $G - \{v\}$. A graph H is a *minor* of G if H can be formed by contracting edges of a subgraph of G.

A *feedback vertex set* of a graph G is a set $S \subseteq V(G)$ such that $G - S$ is a forest, i.e., acyclic. The *feedback vertex number* of a graph is the size of a smallest feedback vertex set of that graph. A *dominating set* of a graph G is a set $D \subseteq V(G)$ such that $N_G[D] = V(G)$. The *domination number* of a graph is the size of a smallest dominating set of that graph. We say that a set $X \subseteq V(G)$ dominates $U \subseteq V(G)$ if $U \subseteq N_G[X]$.

We define $V_{\neq 2}(G)$ to be the set of vertices of graph G that do not have degree two. We refer to a *chain* C of G as a connected component of $G - V_{\neq 2}(G)$. We say that a chain C is a *proper chain* if $N_G(C) \neq \emptyset$ and we then refer to $N_G(C)$ as the *endpoints* of C.

3 Counting feedback vertex sets

In this section, we explain the technique that allows us to either count the number of minimum feedback vertex sets of a graph G in polynomial time, or reduce G to a provably small instance with the same number of minimum solutions. We start by showing that, by using a few reduction rules, we can already reduce G to an equivalent instance with a specific structure. This reduction is based on the $\mathcal{O}(k^3)$-vertex kernel for the decision

Feedback Vertex Set problem presented by Jansen [3]. We choose to use this kernel over the better-known and smaller-size kernels by Thomassé [27] and Iwata [15] because those rely on multiple reduction rules that are not safe for counting minimum solutions.

As is common for Feedback Vertex Set, we consider the graph we are working with to be undirected and we allow parallel edges. In this section, we make use of two reduction rules which are common for kernels of the decision variant of Feedback Vertex Set.

(R1) If there is an edge of multiplicity larger than two, reduce its multiplicity to two.

(R2) If there is a vertex v with degree at most one, remove v.

It can easily be verified that if an instance (G, k) is reduced to (G', k') by one of the rules above, we have $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$. Hence, these rules are safe in the context of counting minimum feedback vertex sets. Observe that if (R2) has exhaustively been applied on a graph G, then all vertices in $V_{\neq 2}(G)$ have degree at least three. For our purposes, we will need one more method to reduce the graph. We first present a lemma that motivates this third reduction rule.

▶ **Lemma 3.1.** *Let X be a (not necessarily minimum) feedback vertex set of a graph G that is reduced with respect to (R2) and let $\mathcal{C}$ be the set of connected components of $G - V_{\neq 2}(G)$. Then:*

(a) $|V_{\neq 2}(G)| \leq |X| + \sum_{v \in X} \deg_G(v)$, *and*

(b) $|\mathcal{C}| \leq |X| + 2\sum_{v \in X} \deg_G(v)$.

Proof. Consider the forest $F := G - X$. Partition the vertices of $V_{\neq 2}(G) \cap V(F) = V_{\neq 2}(G) \setminus X$ into sets $V'_{\leq 1}$, V'_2 and $V'_{\geq 3}$ for vertices that have respectively degree at most one, degree two or degree at least three in F. Furthermore, let V_ℓ denote the leaf nodes of F that have degree two in G. Since G has no vertices of degree at most one by (R2), the leaves of F are exactly the vertices $V_\ell \cup V'_{\leq 1}$. In any tree, the number of vertices of degree at least three is less than the number of leaves, thus $|V'_{\geq 3}| \leq |V'_{\leq 1}| + |V_\ell|$. Each vertex in V'_2 has at least one edge to X since they have degree at least three in G and degree exactly two in F. For a similar reason, each vertex in $V'_{\leq 1}$ has at least two edges to X. Each vertex in V_ℓ has one edge to X. Putting this together gives the following inequality, from which (a) directly follows.

$$\sum_{v \in X} \deg_G(v) \geq |V'_2| + 2|V'_{\leq 1}| + |V_\ell| \geq |V'_2| + |V'_{\leq 1}| + |V'_{\geq 3}| = |V_{\neq 2}(G) \setminus X| \tag{1}$$

To bound the size of the set $\mathcal{C}$ of connected components of $G - V_{\neq 2}(G)$, we instead bound the size of the set $\mathcal{C}'$ of connected components of $G - V_{\neq 2}(G) - X$. Observe that $|\mathcal{C}| \leq |\mathcal{C}'| + |X|$ since removing a vertex from a graph reduces the number of connected components by at most one. (Such a removal can *increase* the number of components by an arbitrary number, which is irrelevant for our argument.) Since X is an FVS, the connected components in $\mathcal{C}'$ can be seen as proper chains. Chains in $\mathcal{C}'$ that have both endpoints in F act as edges between those endpoints when it comes to the connectivity of F. This means that, since F is a forest, there can be at most $|V_{\neq 2}(G) \cap V(F)| \leq \sum_{v \in X} \deg_G(v)$ of such chains by Equation 1. All other chains will have at least one endpoint in X, which means there can be at most $\sum_{v \in X} \deg_G(v)$ of them, implying (b). ◀

Based on Lemma 3.1, the goal of the third reduction rule is to decrease the degree of the vertices of a feedback vertex set. This idea is captured in Lemma 3.2. After presenting this lemma, we combine these results in Lemma 3.3 to create an algorithm to reduce a graph G to an, in context of counting minimum feedback vertex sets, equivalent graph with a bounded number of vertices of degree other than two and a bounded number of chains.

▶ **Lemma 3.2.** *There exists a polynomial-time algorithm that, given a graph G reduced with respect to (R1), an integer k, a vertex $v \in V(G)$ and a feedback vertex set $Y_v \subseteq V(G) \setminus \{v\}$ of G, outputs a graph G' obtained by removing edges from G such that $\deg_{G'}(v) \leq |Y_v| \cdot (k+4)$ and $\#\text{minFVS}(G,k) = \#\text{minFVS}(G',k)$.*

Proof. Consider forest $F := G - (Y_v \cup \{v\})$. For each $u \in Y_v$, mark trees of F that have an edge to both v and u until either all such trees are marked or at least $k+2$ of them are marked. Then, we construct a graph G' from G by removing all edges between v and trees of F that were not marked.

We shall first prove the bound on the degree of v in G'. The vertex v can have edges to vertices in Y_v or in F. Since (R1) has been exhaustively applied, there can be at most $2|Y_v|$ edges between v and Y_v. Each tree in F has at most one edge to v, since otherwise Y_v would not be an FVS of G. In G', only trees that were marked still have an edge to v, and since we mark at most $k+2$ trees per vertex in Y_v, we have at most $|Y_v| \cdot (k+2)$ of such trees. Combining this gives $\deg_{G'}(v) \leq |Y_v| \cdot (k+4)$.

To show that $\#\text{minFVS}(G,k) = \#\text{minFVS}(G',k)$, we prove that a vertex set $X \subseteq V(G)$ with $|X| \leq k$ is an FVS of G if and only if it is an FVS of G'. Clearly any FVS of G is an FVS of G' since G' is constructed from G by deleting edges. For the opposite direction, assume that X is an FVS of G' and assume for a contradiction that X is not an FVS of G. Then $G - X$ has a cycle W. This cycle must contain an edge $\{v,w\}$ of $E(G) \setminus E(G')$ since $G' - X$ is acyclic. By construction of G', we know that w is a vertex that belongs to an unmarked tree T. Since cycle W intersects T and since T is a connected component of $G - (Y_v \cup \{v\})$, there must be a vertex $u \in Y_v$ that has an edge to T. Since the edge between v and T is removed in G', there exist $k+2$ other trees that are marked and have an edge to both v and u. Since $|X| \leq k$, at least two of these trees are not hit by X. Furthermore, since v and u are part of W, they are also not contained in X. Therefore, there is a cycle in $G' - X$ through v, u and two of the aforementioned trees, contradicting that X is an FVS of G'. ◀

▶ **Lemma 3.3.** *There is a polynomial-time algorithm that, given a graph G and integer k, outputs a graph G' and integer k' such that the following properties are satisfied.*

- $\#\text{minFVS}(G,k) = \#\text{minFVS}(G',k')$.
- $k' \leq k$.
- $|V_{\neq 2}(G')| = \mathcal{O}(k^3)$.
- *$G' - V_{\neq 2}(G')$ has $\mathcal{O}(k^3)$ connected components.*

Proof. In our approach, we make use of the linear-time 4-approximation algorithm by Bar-Yehuda et al. [2] that can also approximate the more general problem of: for a given graph, find the smallest FVS that does not contain a given vertex. Our first step is to exhaustively apply (R1) on G and compute a 4-approximate FVS X of the graph. If $|X| > 4k$ then the feedback vertex number of G is larger than k, so we can return a trivial, constant size instance G' and k' such that $\#\text{minFVS}(G',k') = 0$. Otherwise, let G' and k' be a copy of G and k. For each vertex $v \in X$, compute a 4-approximate FVS Y_v of G' that does not contain v. If $|Y_v| > 4k$, then there does not exist a solution of size at most k that does not contain v, so remove v from G' and reduce k' by one. Otherwise, use Lemma 3.2 to reduce the degree of v in G'. Finally, we exhaustively apply reduction rule (R2) on G'.

Computing the 4-approximations and applying the reduction rules can be done in polynomial time. Each rule can only be applied a polynomial number of times, thus the algorithm runs in polynomial time. The fact that $\#\text{minFVS}(G,k) = \#\text{minFVS}(G',k')$

follows from safety of the reduction rules used in the algorithm and $k' \leq k$ follows from the fact that k' is never increased. We know that $|V_{\neq 2}(G')| = \mathcal{O}(k^3)$ and that $G' - V_{\neq 2}(G')$ has $\mathcal{O}(k^3)$ connected components due to Lemma 3.1 and the following bound:

$$\sum_{v \in X} \deg_{G'}(v) \leq \sum_{v \in X} |Y_v| \cdot (k+4) \leq \sum_{v \in X} 4k \cdot (k+4) = |X| \cdot 4k \cdot (k+4) = \mathcal{O}(k^3). \quad \blacktriangleleft$$

The result of Lemma 3.3 is in and of itself not a proper kernel yet, since the chains of the graph it produces can be of arbitrary length. Our strategy to address this is as follows. If these chains are large in terms of k, then we can run an FPT algorithm in $\text{poly}(n)$ time to count the number of minimum solutions. Otherwise, the chains can be replaced by structures of size $\text{poly}(k)$ that do not change the number of minimum feedback vertex sets the instance has. This approach is captured in the following two lemmas and combined in the proof of Theorem 1.1.

▶ **Lemma 3.4.** *There exists a polynomial-time algorithm that, given a graph G with a chain C and an integer k, outputs a graph G' obtained from G by replacing C with a vertex set C', and an integer k', such that:*

- $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$,
- $G - C = G' - C'$,
- $N_G(C) = N_{G'}(C')$,
- $|C'| = \mathcal{O}(\log(|C|)^2)$, *and*
- $k' = k + \mathcal{O}(\log(|C|)^2)$.

Proof. In case C is a proper chain, the endpoints are defined as $N_G(C)$. If C is not a proper chain, which happens if C is a cycle in G, we choose an arbitrary vertex of C to act as its endpoint. For simplicity, we consider C to have two endpoints, where in some cases these two endpoints might be the same vertex.

First, we assume that the number of vertices of the chain, not including its endpoints, is a power of two, i.e. $|C| = 2^p$ for some integer p. Let $v, u \in V(G)$ be the endpoints of C (possibly $v = u$) and let $C = \{c_0, c_1, \cdots, c_{2^p-1}\}$. Then we construct our graph G' by replacing C by a gadget C', of which an example can be seen in Figure 1. It consists of the following elements.

- A vertex w with edges to both v and u.
- Pairs of vertices a_i, b_i for $0 \leq i < p$ such that there is an edge of multiplicity two between w and a_i and between a_i and b_i.

Additionally, we set $k' = k + p$.

We shall now prove that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$. To this end, we define a mapping f from the set of minimum feedback vertex sets of G to those of G' and show that this is a bijection, which immediately implies that the two sets have the same cardinality. For a natural number m, define $\text{bin}(m)$ to be the binary representation of m on p bits and define $\text{bin}(m)_i$ to be the i'th least significant bit of $\text{bin}(m)$.

$$f(X) = \begin{cases} X \cup \{a_i \mid 0 \leq i < p\} & \text{if } X \cap C = \emptyset \\ (X \setminus C) \cup \{w\} \cup \{a_i \mid 0 \leq i < p \wedge \text{bin}(m)_i = 0\} & \\ \cup \{b_i \mid 0 \leq i < p \wedge \text{bin}(m)_i = 1\} & \text{if } X \cap C = \{c_m\} \end{cases}$$

As a first observation, note that for any minimum feedback vertex set X of G, we have $|X \cap C| \leq 1$ as picking any one vertex from C will already break all cycles that go through the chain.

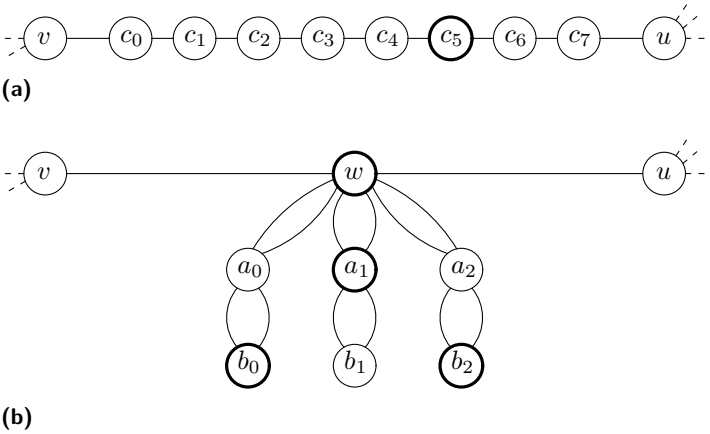

Figure 1 (a) A chain structure of size eight with two endpoints. (b) The replacement of the structure. An example of the mapping f is also illustrated through the vertices with a thicker border.

▷ Claim 3.5. If a set X is a minimum FVS of G, then $f(X)$ is a minimum FVS of G'.

Proof. We prove this in two parts. First we prove that $f(X)$ is an FVS of G' and then we prove that $f(X)$ is indeed an FVS of G' of minimum size.

To prove that $f(X)$ is an FVS of G', we use a proof by contradiction. Assume $f(X)$ is not an FVS of G', in which case a (simple) cycle W exists in $G' - f(X)$. This cycle must intersect C' as $X \setminus C = f(X) \setminus C'$ and $G - C = G' - C'$ thus $G - C - X = G' - C' - f(X)$, which means W would otherwise also exist in $G - X$, contradicting that X is an FVS of G. The cycle W cannot contain a b_j vertex since by definition of f, for $0 \leq i < p$, either a_i or b_i is in $f(X)$, which would either mean b_j is isolated in $G' - f(X)$ or is removed. Also, W can not contain any a_j vertex as, by definition of f, if a_j is not in $f(X)$, then both its neighbors w and b_j are in $f(X)$. This leaves only the option that W intersects C' through only vertex w. This means that $W - \{w\}$ contains a path from v to u in $G' - f(X) - \{w\}$. However, since as mentioned before $G - C - X = G' - C' - f(X)$, this same path also exists in $G - X$. Furthermore, since $w \notin f(X)$, that must mean that $X \cap C = \emptyset$ so $(W \setminus \{w\}) \cup C$ would form a cycle in $G - X$ contradicting that X is an FVS of G.

Next we prove that $f(X)$ is a minimum FVS of G'. Assume for sake of a contradiction that $f(X)$ is not a minimum FVS of G' due to the existence of a $Y \subseteq V(G')$ with $|Y| < |f(X)|$ such that Y is an FVS of G'. We first observe that any FVS of G' contains at least p vertices from C' since all p of the pairs (a_i, b_i) form vertex disjoint cycles. Similarly so, if an FVS of G' contains w, then it contains at least $p + 1$ vertices from C'. We distinguish two cases.

Case $w \notin Y$: Then $Y \setminus C'$ is an FVS of G. If $G - (Y \setminus C')$ would contain a cycle, then there would also exist a cycle in $G' - Y$ since $G - C = G' - C'$ and both graphs $G - (Y \setminus C')$ and $G' - Y$ have a vu path, the former through C and the latter through w. Furthermore, $|Y \setminus C'| \leq |Y| - p < |f(X)| - p = |X| + p - p = |X|$, contradicting that X is a minimum FVS of G.

Case $w \in Y$: Then $X' := (Y \setminus C') \cup \{c_j\}$ is an FVS of G for any $0 \leq j < 2^p$. If this were not the case, then, since X' contains a chain vertex, a cycle would need to exist completely in $G - C - X'$ which is the same graph as $G' - C' - Y$. This would contradict Y being an FVS of G'. Furthermore, $|X'| \leq |Y| - (p + 1) + 1 < |f(X)| - p = |X| + p - p = |X|$, contradicting that X is a minimum FVS of G.

As both cases lead to a contradiction, we conclude that $f(X)$ is a minimum FVS of G'. ◁

▷ Claim 3.6. The function f is bijective.

Proof. We first argue that f is injective. Let X and X' be two minimum feedback vertex sets of G such that $X \neq X'$. That means $X \setminus C \neq X' \setminus C$ or $X \cap C \neq X' \cap C$. The former immediately allows us to conclude that $f(X) \neq f(X')$ since $X \setminus C = f(X) \setminus C'$ and $X' \setminus C = f(X') \setminus C'$. In the second case, we have two options. The first is that $X \cap C = \{c_j\}$ and $X' \cap C = \{c_m\}$ for some $j \neq m$, and since binary representations are unique, they lead to different sets $f(X)$ and $f(X')$. The second option is that either X or X' contains no vertex from C while the other one does. Then only one of $f(X)$ or $f(X')$ contains w and the other one does not, so $f(X) \neq f(X')$.

It remains to show that f is surjective. To this end, we take an arbitrary minimum FVS Y of G' and show that there exists a minimum FVS X of G such that $Y = f(X)$. We distinguish two cases:

Case $w \notin Y$: For this case, first observe that $\{a_i \mid 0 \leq i < p\} \subseteq Y$ since otherwise w would form a cycle with one of these a_i vertices in $G - Y$. Furthermore, $b_i \notin Y$ for $0 \leq i < p$ since Y already contains a_i, leaving b_i isolated in $G - Y$ and thus a redundant choice for a minimum FVS. From this we can conclude that $Y \cap C' = \{a_i \mid 0 \leq i < p\}$. Therefore, taking $X := Y \setminus C'$ would give $f(X) = Y$. We have already argued in case $w \notin Y$, that $Y \setminus C'$ is an FVS of G. To show that it is a minimum FVS of G, note that if there was an $X' \subseteq V(G)$, $|X'| < |X|$ such that X' is an FVS of G, then $f(X')$ is an FVS of G' and $|f(X')| = |X'| + p < |X| + p = |Y \setminus C'| + p = |Y| - p + p = |Y|$ which would contradict Y being a minimum FVS of G'.

Case $w \in Y$: For this case, we instead observe that for each pair $\{a_i, b_i\}$, exactly one of $\{a_i, b_i\}$ is in Y, as otherwise Y would not be an FVS of minimum size. Since there are p of such pairs, there is a unique value $0 \leq j < 2^p$ such that the binary representation of j corresponds to the choice of a and b vertices in Y. We can then choose $X := (Y \setminus C') \cup \{c_j\}$. We clearly have that $f(X) = Y$. Also, we have already shown before that X is an FVS of G and the argument that X is a minimum FVS of G is analogous to that in case $w \notin Y$.

From this reasoning we can conclude that f is a bijective function from the set of minimum feedback vertex sets of G to the set of minimum feedback vertex sets of G. ◁

In the argument above, we assumed that the length of the chain is a power of two. We can address this by noting that any natural number can be written as a sum of unique powers of two. Similarly, we can decompose the chain C into a number of subpaths each having a number of vertices that is a unique power of two. We can then apply the replacement described above on each subpath individually to get multiple replacement structures in a chain between the endpoints of C. As seen before, the size of the replacement is linear in the exponent of the length of the chain. In the worst case, when expressing $|C|$ in binary as a sum of distinct powers of two, the exponents of these powers sum up to $\mathcal{O}(\log(|C|)^2)$, which is also the bound on the number of vertices used in our replacement and on the increase in the parameter value. ◀

We remark for Lemma 3.4 that a similar chain replacement gadget without parallel edges can be constructed, at the expense of a linear increase in the size of the gadget.

By adapting the iterative compression algorithm by Cao et al. [6] for the decision Feedback Vertex Set problem, we can derive the following lemma. Its proof is given in Appendix A.

▶ **Lemma 3.7.** *There is an algorithm that, given a graph G and integer k, computes* $\#\text{minFVS}(G, k)$ *in* $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ *time.*

▶ **Theorem 1.1.** *There is a polynomial-time algorithm that, given a graph G and integer k, either*

- *outputs* $\#\text{minFVS}(G, k)$*, or*
- *outputs a graph G' and integer k' such that* $\#\text{minFVS}(G, k) = \#\text{minFVS}(G', k')$ *and* $|V(G')| = \mathcal{O}(k^5)$ *and* $k' = \mathcal{O}(k^5)$*.*

Proof. First we use the algorithm from Lemma 3.3 to find a graph G^* and integer $k^* \leq k$ such that $\#\text{minFVS}(G, k) = \#\text{minFVS}(G^*, k^*)$, $|V_{\neq 2}(G^*)| = \mathcal{O}(k^3)$ and $G^* - V_{\neq 2}(G^*)$ has $\mathcal{O}(k^3)$ connected components (chains). Then, if there is a chain of size larger than 2^k, we can run the algorithm from Lemma 3.7 to compute $\#\text{minFVS}(G^*, k^*)$ in $\text{poly}(n)$ time since $n > 2^k$. Otherwise, all chains of G^* have size at most 2^k and we can use Lemma 3.4 on G^* to find a graph G' and integer k' such that:

- $\#\text{minFVS}(G', k') = \#\text{minFVS}(G^*, k^*) = \#\text{minFVS}(G, k)$,
- $|V(G')| = \mathcal{O}(k^3) + \mathcal{O}(k^3 \cdot \log(2^k)^2) = \mathcal{O}(k^5)$, and
- $k' = \mathcal{O}(k + k^3 \cdot \log(2^k)^2) = \mathcal{O}(k^5)$. ◀

4 Conclusion

We introduced a new model of kernelization for counting problems: a polynomial-time preprocessing algorithm that either outputs the desired count, or reduces to a provably small instance with the same answer. We showed that for counting the number of minimum solutions of size at most k, a reduction to a graph of size $\text{poly}(k)$ exists for two classic problems.

We believe that the new viewpoint on counting kernelization facilitates a general theory that can be explored for many problems beyond the ones considered here. By following the textbook proof [9, Lemma 2.2] that a decidable parameterized decision problem is fixed-parameter tractable if and only if it admits a kernel (of potentially exponential size), it is easy to show the following equivalence between fixed-parameter tractability of counting problems and our notion of counting kernelization.

▶ **Lemma 4.1.** *Let $\mathcal{P} \colon \Sigma^* \times \mathbb{N} \to \mathbb{N}$ be a computable function for some finite alphabet Σ. Then the following two statements are equivalent:*

1. *There is a computable function $f \colon \mathbb{N} \to \mathbb{N}$ and an algorithm that, given an input $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs $\mathcal{P}(x, k)$ in time $f(k) \cdot |x|^{\mathcal{O}(1)}$.*
2. *There is a computable function $f \colon \mathbb{N} \to \mathbb{N}$ and a polynomial-time algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, either:*
   a. *outputs $\mathcal{P}(x, k)$, or*
   b. *outputs $(x', k') \in \Sigma^* \times \mathbb{N}$ satisfying $|x'|, k' \leq f(k)$ and $\mathcal{P}(x, k) = \mathcal{P}(x', k')$.*

Hence our view of counting kernelization is generic enough to capture all fixed-parameter tractable counting problems. Determining which counting problems have a *polynomial-size* kernel remains an interesting challenge.

In this work, we focused on counting *minimum-size* solutions (of size at most k). Apart from being of practical interest in several applications, this facilitates several steps in the design and analysis of our preprocessing algorithms. At present, we do not know whether the two considered problems have polynomial-size kernels when counting the number of inclusion-minimal solutions of size exactly k, or the number of (not necessarily minimal or minimum) solutions of size exactly k; we leave this investigation to future work. To see the importance of the distinction, observe that the number of *minimum* vertex covers of size

at most k in a graph is bounded by 2^k (since the standard 2-way branching discovers all of them) and the Buss kernel preserves their count. But the total number of vertex covers of size at most k cannot be bounded in terms of k in general.

For both problems we investigated, our preprocessing step effectively consists of reducing to an equivalent instance composed of a small core along with poly(k) simply structured parts, followed by replacing each such part by a small problem-specific gadget. In the world of decision problems, the theory of protrusion replacement [4, 13] gives a generic way of replacing such simply-structured parts by gadgets. Similarly, the condenser-extractor framework [18, 26] can be applied to generic problems as long as they can be captured in a certain type of logic. This leads to the question of whether, in our model of counting kernelization, the design of the gadgets can be automated. Can a notion of meta-kernelization be developed for counting problems?

References

1 Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.

2 Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM J. Comput.*, 27(4):942–959, 1998.

3 Bart M. P. Jansen. The power of preprocessing: Gems in kernelization. https://www.win.tue.nl/~bjansen/talks/BonnGemsInKernelization.pptx, 2016.

4 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.

5 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993. doi:10.1137/0222038.

6 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

7 Radu Curticapean. The simple, little and slow things count: on parameterized counting complexity. *Bull. EATCS*, 120, 2016. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/445.

8 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Henning Fernau. *Parameterized algorithmics: A graph-theoretic approach*. PhD thesis, Habilitationsschrift, Universität Tübingen, Germany, 2005.

11 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

12 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

13 Fedor V. Fomin. Protrusions in graphs and their applications. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, page 3. Springer, 2010. doi:10.1007/978-3-642-17493-3_2.

14 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. *SIAM J. Comput.*, 49(6):1397–1422, 2020. doi:10.1137/16M1080264.

15 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.

16 Bart M. P. Jansen and Bart van der Steenhoven. Kernelization for counting problems on graphs: Preserving the number of minimum solutions. *CoRR*, abs/2310.04303, 2023. arXiv:2310.04303.

17 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.

18 Eun Jung Kim, Maria J. Serna, and Dimitrios M. Thilikos. Data-compression for parametrized counting problems on sparse graphs. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 20:1–20:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.20.

19 Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Kernelization of counting problems. *CoRR*, abs/2308.02188, 2023. doi:10.48550/arXiv.2308.02188.

20 Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Struct. Algorithms*, 15(3-4):229–241, 1999.

21 Catherine McCartin. Parameterized counting problems. *Annals of Pure and Applied Logic*, 138(1):147–182, 2006. doi:10.1016/j.apal.2005.06.010.

22 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science (New York, N.Y.)*, 298:824–7, November 2002. doi:10.1126/science.298.5594.824.

23 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Parameterized counting algorithms for general graph covering problems. In Frank K. H. A. Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, volume 3608 of *Lecture Notes in Computer Science*, pages 99–109. Springer, 2005. doi:10.1007/11534273_10.

24 Pekka Orponen. Dempster's rule of combination is #p-complete. *Artif. Intell.*, 44(1-2):245–253, 1990.

25 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

26 Dimitrios M. Thilikos. Compactors for parameterized counting problems. *Comput. Sci. Rev.*, 39:100344, 2021. doi:10.1016/j.cosrev.2020.100344.

27 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.

28 Marc Thurley. Kernelizations for parameterized counting problems. In Jin-yi Cai, S. Barry Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007, Proceedings*, volume 4484 of *Lecture Notes in Computer Science*, pages 703–714. Springer, 2007. doi:10.1007/978-3-540-72504-6_64.

29 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

30 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

31 D. J. A. Welsh. Graph theory and theoretical physics. *The Mathematical Gazette*, 54(390):432–433, 1970. doi:10.2307/3613919.

A An FPT algorithm for #minFVS

▶ **Lemma 3.7.** *There is an algorithm that, given a graph G and integer k, computes* $\#\text{minFVS}(G,k)$ *in* $2^{\mathcal{O}(k)} \cdot \text{poly}(n)$ *time.*

Proof. The pseudocode of an algorithm to solve the disjoint minimum feedback vertex set counting problem is given in Algorithm 2 and is based on the algorithm by Cao et al [6]. In fact, our algorithm solves a slightly more general version of the problem, where the vertices of the graph G are weighted by a function $w\colon V(G) \to \mathbb{N}$. In case G has no FVS of size at most k that is disjoint from W, then #DJ-FVS(G, w, W, k) will output a pair (a,b) with $a = \infty$ and $b = 0$. Otherwise, a will be the size of a minimum FVS disjoint of W and

$$b = \sum_{\substack{|S|=a,\\ S\in\text{FVS}(G),\\ S\cap W=\emptyset}} \prod_{v\in S} w(v).$$

We refer to this value as the *weighted disjoint minimum FVS sum* of G. The weight of a vertex v essentially models the number of distinct alternatives there are for a vertex v that, in the context of choosing them for a minimum FVS of G, achieve the same result. When we assign a weight of one to each vertex of a graph G, then the weighted minimum FVS sum of G is equal to the number of minimum feedback vertex sets of G.

In the pseudocode we make use of binary operator $\oplus$, which is defined to work on pairs as follows:

$$(a_1,b_1)\oplus(a_2,b_2) = \begin{cases} (a_1,b_1) & \text{if } a_1 < a_2 \\ (a_2,b_2) & \text{if } a_1 > a_2 \\ (a_1,b_1+b_2) & \text{if } a_1 = a_2 \end{cases}$$

For the operators $+$ and $\cdot$ we assume element-wise functionality when applied to pairs, i.e. $(a_1,b_1)+(a_2,b_2) = (a_1+a_2, b_1+b_2)$. Furthermore, for a weight function w of G and $X \subseteq V(G)$, we use $w|_X$ to denote the restriction of w to X.

We shall now explain how the #DJ-FVS algorithm works by going over the pseudocode in Algorithm 2. It makes use of a branching strategy with measure function $\ell + k$, where ℓ is the number of connected components of $G[W]$.

Lines 1-3 are the base cases of the algorithm. In lines 4-5 we remove vertices of degree at most one, which corresponds to (R2). In lines 6-7 we contract the chains of G existing in $G - W$ one edge at a time. Note for line 8 that H is a forest since W is an FVS of G. For lines 9-10, if a vertex v would form a cycle with W it should be contained in all solutions so we recurse on this choice. In lines 11-14, vertex v has at least two neighbors in W, but since $G[W \cup \{v\}]$ does not form a cycle these neighbors must be in different connected components of $G[W]$. Thus we branch on v not being in a solution, which corresponds with adding v to W, and on v being in a solution by removing it from the graph. In the former branch ℓ decreases, while in the latter k decreases.

In line 15, we choose a vertex $v \in V(H)$ that is not a leaf of tree H such that at most one of its neighbors in H is not a leaf of H. Note that such a vertex always exists for a tree that does not consist of only leaves. Furthermore, at this point of the algorithm no tree of H can consist of only leaves. If a tree of H consists of a single leaf, then either it has degree one in G, but then lines 4-5 would have gotten rid of it, or it has degree at least two, in which case the if condition on line 9 or the if condition on line 11 would have been satisfied. If a

tree of H consists of two leaves, then either both have degree two in which case the edge between them would have been contracted by lines 6-7, or one of the if statements of line 9 or line 11 would have been applicable to one of the two leaves.

First consider the case where v has one neighbor in W (lines 16-22). In that case we pick c in line 16 to be a child of v. For these two vertices, we branch over all possible combinations of whether or not they should be in a solution, while realizing that a minimum FVS that contains v can never contain c. Note that if $G[W \cup \{v, c\}]$ does not form a cycle, both v and c must have a neighbor in a different connected component of W so the branch in line 19 decreases ℓ. The branches on line 20 and 21 decrease k while not increasing ℓ.

Finally, we have the case that v has no neighbors in W (lines 23-31). That must mean that v has at least two children, which are all leaves by how we chose v. If v would have had only one child, then the edge between v and the child would have been contracted in lines 6-7. Thus we let c_1 and c_2 be two distinct children of v. Again, we branch over all viable combinations of how these three vertices can be part of solutions. The logic here is similar to that of the previous case. Here as well, in all branches, the measure $k + \ell$ decreases.

Since the algorithm branches in at most five directions and the time per iteration is polynomial in n, the runtime of the disjoint algorithm becomes $5^{k+\ell} \cdot n^{O(1)}$. We can use Algorithm 2 to compute #minFVS(G, k) by taking an FVS Z of G and running the disjoint algorithm for all subsets of Z, simulating the ways an FVS can intersects Z. The pseudocode for this compression algorithm can be seen in Algorithm 1. To get an FVS of G of size at most k, we can simply run one of the existing algorithms designed for this, for example the one by Cao et al. [6] which runs in time $\mathcal{O}(3.83^k) \cdot \text{poly}(n)$. If this reports that no such FVS exists, we output #minFVS$(G, k) = 0$. Otherwise, we use the found FVS for the compression algorithm. Using the fact that the FVS used by the compression algorithm is of size at most k, we can bound the runtime of the complete algorithm to compute #minFVS(G, k) at $26^k \cdot n^{O(1)}$. ◀

Algorithm 1 #FVS-compression(G, k, Z).

Input: Graph G, integer k, FVS Z of G of size at most k
Output: A pair (a, b) with a being the feedback vertex number of G and $b = \#\text{minFVS}(G, k)$

1: $s = (\infty, 0)$
2: **for** $X_Z \subseteq Z$ **do**
3: $\quad s' = (|X_Z|, 0) + \#\text{DJ-FVS}(G - X_Z, w, Z \setminus X_Z, k - |X_Z|)$ with $\forall v \in V(G - X_Z) : w(v) = 1$
4: $\quad s = s \oplus s'$
5: **return** s

Algorithm 2 #DJ-FVS(G, w, W, k).

Input: Graph G, weight function $w\colon V(G) \to \mathbb{N}$, FVS W of G, integer k
Parameter: $k + \ell$, with $\ell = \#$ components of $G[W]$
Output: A pair (a, b) where a is the size of a minimum FVS S of G such that $S \cap W = \emptyset$ and b is the weighted disjoint minimum FVS sum of G. If no such FVS of size at most k exists, then $(a, b) = (\infty, 0)$.

if $k < 0$ **then return** $(\infty, 0)$
if $G[W]$ has a cycle **then return** $(\infty, 0)$
if $G - W$ is empty **then return** $(0, 1)$
if $\exists v \in V(G - W)$: $deg_G(v) \leq 1$ **then**
    return #DJ-FVS($G - v$, $w|_{V(G)\setminus\{v\}}$,W,k)
if $\exists \{v, u\} \in E(G) : \deg_G(v) = \deg_G(u) = 2$ and $v, u \notin W$ **then**
    return #DJ-FVS(G', w' W, k), where G' is G with edge $\{v, u\}$ contracted to a single vertex s and w' is the weight function $w|_{V(G')}$ with $w'(s) = w(v) + w(u)$.
Let H be forest $G - W$.
if $\exists v \in V(H)$: $G[W \cup \{v\}]$ has a cycle **then**
    return $(1, 0) + (1, w(v))\cdot$ #DJ-FVS($G - v$, $w|_{V(G)\setminus\{v\}}$, W, $k - 1$)
if $\exists v \in V(H)$: $|N_G(v) \cap W| \geq 2$ **then**
    $X_0 =$#DJ-FVS(G, w, $W \cup \{v\}$, k)
    $X_1 = (1, 0) + (1, w(v))\cdot$ #DJ-FVS($G - v$, $w|_{V(G)\setminus\{v\}}$, W, $k - 1$)
    return $X_0 \oplus X_1$
Let $v \in V(H)$ be a vertex that is not a leaf of H such that at most one vertex in $N_H(v)$ is not a leaf of H.
if $|N_G(v) \cap W| = 1$ **then**
    Pick $c \in V(H)$ such that $N_G(c) = \{v, x\}$ for some $x \in W$
    if $G[W \cup \{v, c\}]$ forms a cycle **then** $X_{00} = (\infty, 0)$
    else $X_{00} =$ #DJ-FVS(G, w, $W \cup \{v, c\}$, k)
    $X_{10} = (1, 0) + (1, w(v))\cdot$ #DJ-FVS($G - v$, $w|_{V(G)\setminus\{v\}}$, W, $k - 1$)
    $X_{01} = (1, 0) + (1, w(c))\cdot$ #DJ-FVS($G - c$, $w|_{V(G)\setminus\{c\}}$, $W \cup \{v\}$, $k - 1$)
    return $X_{00} \oplus X_{10} \oplus X_{01}$
if $|N_G(v) \cap W| = 0$ **then**
    Pick $c_1, c_2 \in V(H), c_1 \neq c_2$ such that $N_G(c_1) = \{v, x\}$ and $N_G(c_2) = \{v, y\}$ for some $x, y \in W$
    if $G[W \cup \{v, c_1, c_2\}]$ forms a cycle **then** $X_{000} = (\infty, 0)$
    else $X_{000} =$ #DJ-FVS(G, w, $W \cup \{v, c_1, c_2\}$, k)
    $X_{100} = (1, 0) + (1, w(v))\cdot$ #DJ-FVS($G - v$, $w|_{V(G)\setminus\{v\}}$, W, $k - 1$)
    $X_{010} = (1, 0) + (1, w(c_1))\cdot$ #DJ-FVS($G - c_1$, $w|_{V(G)\setminus\{c_1\}}$, $W \cup \{v, c_2\}$, $k - 1$)
    $X_{001} = (1, 0) + (1, w(c_2))\cdot$ #DJ-FVS($G - c_2$, $w|_{V(G)\setminus\{c_2\}}$, $W \cup \{v, c_1\}$, $k - 1$)
    $X_{011} = (2, 0) + (1, w(c_1) \cdot w(c_2))\cdot$ #DJ-FVS($G - \{c_1, c_2\}$, $w|_{V(G)\setminus\{c_1, c_2\}}$, $W \cup \{v\}$, $k - 2$)
    return $X_{000} \oplus X_{100} \oplus X_{010} \oplus X_{001} \oplus X_{011}$