Kernelization of Counting Problems

Daniel Lokshtanov
University of California Santa Barbara, CA, USA

Pranabendu Misra
Chennai Mathematical Institute, Chennai, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Meirav Zehavi
Ben-Gurion University of the Negev, Beersheba, Israel

Abstract

We introduce a new framework for the analysis of preprocessing routines for parameterized counting problems. Existing frameworks that encapsulate parameterized counting problems permit the usage of exponential (rather than polynomial) time either explicitly or by implicitly reducing the counting problems to enumeration problems. Thus, our framework is the only one in the spirit of classic kernelization (as well as lossy kernelization). Specifically, we define a compression of a counting problem P into a counting problem Q as a pair of polynomial-time procedures: reduce and lift. Given an instance of P, reduce outputs an instance of Q whose size is bounded by a function f of the parameter, and given the number of solutions to the instance of Q, lift outputs the number of solutions to the instance of P. When $P = Q$, compression is termed kernelization, and when f is polynomial, compression is termed polynomial compression. Our technical (and other conceptual) contributions can be classified into two categories:

Upper Bounds. We prove two theorems: *(i)* The #Vertex Cover problem parameterized by solution size admits a polynomial kernel; *(ii)* Every problem in the class of #Planar $\mathcal{F}$-Deletion problems parameterized by solution size admits a polynomial compression.

Lower Bounds. We introduce two new concepts of cross-compositions: EXACT-cross-composition and SUM-cross-composition. We prove that if a #P-hard counting problem P EXACT-cross-composes into a parameterized counting problem Q, then Q does not admit a polynomial compression unless the polynomial hierarchy collapses. We conjecture that the same statement holds for SUM-cross-compositions. Then, we prove that: *(i)* #Min (s,t)-Cut parameterized by treewidth does not admit a polynomial compression unless the polynomial hierarchy collapses; *(ii)* #Min (s,t)-Cut parameterized by minimum cut size, #Odd Cycle Transversal parameterized by solution size, and #Vertex Cover parameterized by solution size minus maximum matching size, do not admit polynomial compressions unless our conjecture is false.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Kernelization, Counting Problems

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.77

Related Version *Full Version*: https://arxiv.org/abs/2308.02188

Funding *Daniel Lokshtanov*: Supported by NSF award CCF-2008838.
Saket Saurabh: European Research Council (ERC) grant agreement no. 819416, and Swarnajayanti Fellowship no. DST/SJF/MSA01/2017-18.
Meirav Zehavi: European Research Council (ERC) grant titled PARAPATH.

1 Introduction

Preprocessing is an integral part of almost any application, ranging from lossless data compression to microarray data analysis for the classification of cancer types. Therefore, *kernelization* (or, more generally, *compression*), the mathematical paradigm to analyze preprocessing procedures, is termed "the lost continent of polynomial time" [21]. Formally, a decision problem P admits a *compression* into a decision problem Q if there exists a polynomial-time algorithm that, given an instance (I, k) of P, translates it into an equivalent[1] instance (I', k') of Q of size $f(k)$ for some computable function f that depends only on k. When $P = Q$, a compression is termed *kernelization*. It is known that a (decidable) problem admits a kernel if and only if it is in *fixed-parameter tractable (FPT)* [7].[2] Thus, the most central question in kernelization is: Which problems admit compressions (or kernels) of size $f(k)$ where f is polynomial in k, termed *polynomial compressions*? Techniques to show upper bounds on (polynomial or other) kernel sizes have already emerged in the early 1990s [25]. On the other hand, Bodlaender et al. [3] proved that, unless the polynomial hierarchy collapses, there exist problems that do not admit a polynomial compression (and, hence, neither a polynomial kernel).

Due to the centrality and mathematical depth of compression/kernelization, the underlying framework has been extended to capture optimization problems, and, more generally, the computation of approximate (rather than only exact) solutions for optimization problems, by Lokshtanov et al. [34] (building upon [22]). In particular, a compression of an optimization problem P into an optimization problem Q is a pair of polynomial-time procedures: reduce and lift. Given an instance of P, reduce outputs an instance of Q whose size is bounded by a function f of the parameter, and given an optimal solution to the instance of Q, lift outputs an optimal solution to the instance of P. More generally, to encompass the computation of approximate solutions with a loss of factor $\alpha \geq 1$, given a β-approximate solution to the instance of Q, for any $\beta \geq 1$, lift must output an $\alpha \cdot \beta$-approximate solution to the instance of P. Since its introduction, this notion of compression/kernelization (termed *lossy compression/kernelization*) has already found a wide range of applications; see, e.g., [37, 27, 20, 33, 32, 1, 44, 19] for just a few illustrative examples.

In this paper, we introduce a new framework for the analysis of preprocessing routines for parameterized counting problems. Existing frameworks that encapsulate parameterized counting problems permit the usage of exponential (rather than polynomial) time either explicitly or by implicitly reducing counting problems to enumeration problems (see Section 1.1). Thus, our framework is the only one in the spirit of classic compression/kernelization in particular, and lossy compression/kernelization in general. Specifically, we define a compression of a counting problem P into a counting problem Q as a pair of polynomial-time procedures: reduce and lift. Given an instance of P, reduce outputs an instance of Q whose size is bounded by a function f of the parameter, and given the number of solutions to the instance of Q, lift outputs the number of solutions to the instance of P. We demonstrate the depth of our framework by proofs of both positive and negative results (see Section 1.2). In particular, in terms of conceptual contribution, in addition to the framework itself, we also introduce two new types of cross-compositions, termed EXACT- and SUM-cross-compositions, aiming to provide analogs to the classic OR- and AND-cross-compositions used to derive negative results for (classic) kernels.

[1] That is, (I, k) is a yes-instance if and only if (I', k') is a yes-instance.
[2] We refer to Section 3 for basic definitions in parameterized complexity and graph theory.

Over the past two decades, the body of works on parameterized counting problems has grown quite rapidly (see, e.g., [23, 11, 36, 5, 15, 13, 12] for a few illustrative examples of recent developments). In both theory and practice, there are various scenarios where counting the number of solutions might be equally (or more) important than only detecting a single solution (if one exists) [10]. This includes, for example, the computation of graph motifs to observe certain phenomena in social and biological networks [38], and determination of thermodynamic properties of discrete systems by partition functions [28]. However, most natural counting problems are not known to (and unlikely to) admit polynomial-time algorithms: Beyond problems whose decision versions are NP-hard, there also exist numerous problems whose decision versions are solvable in polynomial time, but whose counting versions are unlikely to be (e.g, a prime example of such problems is the Maximum Matching problem on bipartite graphs [43, 42]). Naturally, this makes the study of the parameterized complexity of counting problems very attractive.

1.1 Related Frameworks

Prior to our work, there existed three frameworks relevant to the analysis of preprocessing routines for parameterized counting problems. However, all of these three frameworks (explicitly or implicitly) correspond to computation in exponential (rather than polynomial) time, as well as to either enumeration (rather than counting) or data reduction other than compression/kernelization. Thus, they serve purposes that are very different than what *compression/kernelization* of parameterized *counting* problems should be (though, of course, they are of interest on their own right). Moreover, we are not aware, with respect to any of these three frameworks, of the establishment of any non-trivial lower bound – that is, a lower bound that does not simply follows from fixed-parameter intractability. Below, we elaborate on each of these three frameworks.

Among the three aforementioned frameworks, the one whose utility is most similar to ours was developed by Thurley [41], yet, even this framework concerns, implicitly, enumeration and computation in exponential time (indeed, it is referred to as a formalization of so-called enumeration compactors in [30], and as a reduction of counting to enumeration in [26]). Roughly speaking, the definition of Thurley [41] can be interpreted as follows when using two polynomial-time procedures (as we do), reduce and lift. Here, given an instance of a *counting* problem P, reduce outputs an instance of an *enumeration* problem Q whose size is bounded by a function f of the parameter. We suppose that each solution to the instance of Q corresponds to a set of solutions to the instance of P; then, the collection of sets of solutions to the instance P corresponding to the different solutions to the instance of Q should form a partition of the set of solutions to the instance of P. Accordingly, given a particular *solution* s to the instance of Q, lift outputs the number of solutions to the instance of P that correspond to s. In particular, given an *enumeration* of the solutions to the instance of Q, by calling lift for each one of them, we can obtain (in exponential time, depending on the number of solutions) the number of solutions to the instance of P.

The second framework is explicitly designed for enumeration problems. Still, we briefly discuss it here, since it shares some similarity to the framework of Thurley [41]. This framework was introduced by Creignou et al. [9] and refined by Golovach et al. [26]. Roughly speaking, in its latter incarnation, we are also given two polynomial-time procedures, reduce and lift. Here, given an instance of an *enumeration* problem P, reduce outputs an instance of an *enumeration* problem Q whose size is bounded by a function f of the parameter. Then, lift is defined similarly as before, except that now, given a particular solution s to the instance

of Q, it enumerates (either in polynomial time or with polynomial delay) the solutions to the instance of P that correspond to s. Like before, to derive the number of solutions to the instance of P, it is required to spend exponential time.

The third framework is designed specifically for counting, but it is less in the spirit of compression/kernelization, and, accordingly, it is termed *compaction*. Additionally and similarly to the two aforementioned frameworks, it corresponds to computation in exponential time. This framework was introduced by Kim et al. [30] (and further surveyed in [40]). Roughly speaking, here we consider a polynomial-time procedure compactor (that can be thought of as reduce) and an *exponential-time (or worse)* procedure extractor (that is very different in spirit than lift). Here, given an instance of a counting problem P, compactor outputs an instance of a counting problem Q whose size is bounded by a function f of the parameter. Having computed the output instance, one can essentially discard all knowledge of the input instance, yet call the procedure extractor to solve the input instance. In a sense, the definition of compaction can be viewed as an "intermediate" concept that lies in between those of a fixed-parameter algorithm and a compression algorithm, which is of interest on its own right. Perhaps the main drawback of this third framework is that, because extractor is allowed (and must be allowed, if we deal with a #P-hard problem) to spend exponential-time (or worse) in the size of the output of compactor, we might often want to employ, in the first place, a fixed-parameter algorithm directly on the instance of P.

Very recently, simultaneously and independently of our work, Jansen and van der Steenhoven [29] presented results that are more in-lined in spirit with ours: Specifically, they either solve the given instance, or output an instance of size polynomial in the parameter and with the same number of solutions. They also speculate on developing a meaningful theory of counting kernelization. We answer this speculation as in this paper, as we develop a framework counting kernelization, along with a framework for proving lower-bounds.

1.2 Our Contribution

Our technical (and other conceptual) contributions can be classified into two categories: upper bounds and lower bounds. Here, we discuss the statements our results, and the new concepts that we introduce in the context of lower bounds. The technical aspects of our work are overviewed later, in Section 2. (We remark that some additional simple statements concerning our notion of compression/kernelization are proved in Section 4.)

Upper Bounds. Let us start with the discussion of our upper bounds. We begin by the analysis of the #k-Vertex Cover problem, whose decision version is the most well studied problem in parameterized complexity [14, 17]. The objective is to count the number of vertex covers of size at most k in a given graph G. Here, it is important to note that we count *all* vertex covers of size at most k, and not only the minimal ones (which is a significantly easier task; see Section 5). For the #k-Vertex Cover problem, we prove the following theorem in Section 5.

▶ **Theorem 1.** #k-Vertex Cover *admits a polynomial kernel.*

Next, we turn to consider a wide class of parameterized counting problems, termed the class of #k-Planar $\mathcal{F}$-Deletion problems. In particular, the class of k-Planar $\mathcal{F}$-Deletion problems encompasses a wide variety of well-known problems that have been extensively studied from the viewpoint of parameterized complexity, such as Vertex Cover, Feedback Vertex Set, Treewidth η-Deletion, and more [24]. While we present a meta-theorem that resolves every problem in this class, we do not generalize our previous

theorem – our meta-theorem yields compressions rather than kernelizations. Formally, the class of #k-Planar $\mathcal{F}$-Deletion problems contains one problem for every (finite) set of connected graphs $\mathcal{F}$ that contains at least one planar graph – here, given a graph G and $k \in \mathbb{N}_0$, the objective is to count the number of vertex sets of size at most k whose removal from G yields a graph that does not contain any graph from $\mathcal{F}$ as a minor. For the class of #k-Planar $\mathcal{F}$-Deletion problems, we prove the following theorem.

▶ **Theorem 2.** #k-Planar $\mathcal{F}$-Deletion *admits a polynomial compression.*

Lower Bounds. We present two new types of cross-compositions, which we term EXACT-cross-composition and SUM-cross-composition. To understand the roots of these notions, let us first briefly present the classic notion of OR-cross-composition. Roughly speaking, we say that a decision problem P OR-cross-composes into a parameterized problem Q if, given a set of instances $x_1, x_2, \ldots, x_t$ of P, we can, in polynomial time, output a single instance (y, k) of Q with the following properties: *(i)* the parameter k is bounded by a polynomial function of $\max_{i=1}^{t} |x_i|$ and $\log t$, and *(ii)* (y, k) is a yes-instance if and only if at least one x_i is a yes-instance. The importance of the notion of OR-cross-composition to compression/kernelization is rooted at the following theorem: If an NP-hard problem P OR-cross-composes into a parameterized problem Q, then, Q does not admit a polynomial compression (and, hence, neither a polynomial kernel), unless coNP $\subseteq$ NP/poly [3, 4]. The intuition behind the correctness of this theorem is that, if Q did admit a polynomial compression, then that would have meant that, in polynomial time, we are able to turn t instances of an NP-hard problem to a single instance whose size depends (roughly) only on that size of a polylogarithmic number of them rather than all of them – intuitively, this means that we were able to resolve instances of an NP-hard problem in polynomial time.

Now, let us first discuss our notion of EXACT-cross-composition.[3] Roughly speaking, we say that a counting problem P EXACT-cross-composes into a parameterized counting problem Q if, given a set of instances $x_1, x_2, \ldots, x_t$ of P, we can, in polynomial time, output a single instance (y, k) of Q with the following properties: *(i)* the parameter k is bounded by a polynomial function of $\max_{i=1}^{t} |x_i|$ and $\log t$, and *(ii)* given the number of solutions to (y, k), we can output, in polynomial time, the number of solutions to x_i for every $i \in \{1, 2, \ldots, t\}$. For EXACT-cross-composition, we prove the following theorem.

▶ **Theorem 3.** *Assume that a #P-hard counting problem P EXACT-cross-composes into a parameterized counting problem Q. Then, Q does not admit a polynomial compression, unless #P $\subseteq$ "NP/poly" (which implies that coNP $\subseteq$ NP/poly).*

For an application of Theorem 3, we consider the classic #Min (s, t)-Cut problem. Here, given a graph G and two vertices s, t in G, the objective is to count the number of minimum (s, t)-cuts in G. Notably, the decision version of this problem is solvable in polynomial time [8] (and, hence, it trivially admits a polynomial, and even constant-size, kernel, with respect to any parameter). Moreover, it is easy to see that #Min (s, t)-Cut parameterized by treewidth is in FPT. So, it is natural to ask whether #Min (s, t)-Cut parameterized by treewidth admits a polynomial kernel (or at least a polynomial compression). We answer this question negatively.

[3] In the manuscript, we consider SUM-cross-composition first since the reduction we give in the context of EXACT-cross-composition builds upon one of the reductions that we give in the context of SUM-cross-composition.

▶ **Theorem 4.** *#w-Min (s,t)-Cut does not admit a polynomial compression, unless #P ⊆ "NP/poly" (which implies that coNP ⊆ NP/poly).*

Lastly, let us discuss our notion of SUM-cross-composition. Roughly speaking, we say that a counting problem P SUM-cross-composes into a parameterized counting problem Q if, given a set of instances $x_1, x_2, \ldots, x_t$ of P, we can, in polynomial time, output a single instance (y,k) of Q with the following properties: *(i)* the parameter k is bounded by a polynomial function of $\max_{i=1}^{t} |x_i|$ and $\log t$, and *(ii)* the number of solutions to (y,k) is equal to the sum of the number of solutions to x_i over every $i \in \{1,2,\ldots,t\}$. For SUM-cross-composition, we have the following conjecture, termed the SUM-conjecture: If a #P-hard counting problem P SUM-cross-composes into a parameterized counting problem Q, then Q does not admit a polynomial compression. The reason why we believe that this conjecture is true is rooted at the exact same intuition mentioned earlier for the correctness of the corresponding theorem for OR-cross-composition.

As applications of our conjecture, we again consider the #Min (s,t)-Cut problem, now parameterized by the size of a minimum (s,t)-cut (which is in FPT [2]). Additionally, we consider the #Odd Cycle Transversal problem parameterized by solution size and the #Vertex Cover problem parameterized by solution size minus either its LP-value or the size of a maximum matching (we refer to Section 3 for formal definitions). We remark that the decision versions of these parameterized counting problems are known to admit polynomial kernels [31]. For the aforementioned parameterized counting problems, we prove the following theorem.

▶ **Theorem 5.** *#k-Min (s,t)-Cut, #k-Odd Cycle Transversal, #ℓ-Vertex Cover and #m-Vertex Cover do not admit polynomial compressions, unless the SUM-conjecture is false.*

2 Overview of Our Proofs

In what follows, we present an overview for the proofs of our main theorems. We start by discussing the positive results, being Theorems 1 and 2.

Proof of Theorem 1. Our reduction consists of two steps. Here, we note that most of our efforts are invested in the second step. The first step yields two graphs: G_1 and G_2. We begin by an exhaustive application of the classic Buss rule (Definition 22) on the input instance (G,k). In particular, unless the answer is 0, this yields an instance (G_1,k_1) with $k_1 \leq k$ and $|E(G_1)| \leq k_1^2$ whose number of solution equals the number of solutions to G. At this point, we do not have a kernel (or compression) – $|V(G_1)|$ can contain arbitrarily many isolated vertices. So, we define G_2 as G_1 where all isolated vertices are removed. However, the number of solutions (denoted by x_2) to (G_2,k_1) can be very different than the number of solutions (denoted by x_1) to (G_1,k_1), and it is unclear how to derive the second from the first. Specifically, suppose that y_i, $i \in \{1,2,\ldots,k_2\}$, is the number of solutions to (G_2,k_1) of size exactly i. It is easy to see that $x_1 = \sum_{i=0}^{k_2}(y_i \cdot \sum_{j=0}^{k_2-i} \binom{|V(G_1)|-|V(G_2)|}{j})$. However, by knowing x_2, we cannot know the individual values of the y_i's! Although $x_2 = \sum_{i=0}^{k_2} y_i$, there can be more than one choice (in fact, there can be exponentially many choices) for the y_i's given only the knowledge of x_2.

Due to the above difficulty, we perform the second step of our reduction. Roughly speaking, we define G_3 (in Definition 26) by the replacement of each vertex of G_2 by d copies (false twins) of that vertex, and the addition of t new isolated vertices. To make

latter calculations work, we pick $d = |V(G_2)| \leq \mathcal{O}((k_2)^2)$, and we pick t to be "large enough" compared to d. Now, our main objective is to prove how from the number of solutions to (G_3, k_3) (denoted by x_3), we can derive the individual values of the y_i's.

To achieve the above-mentioned objective, we define a mapping from the set of solutions to (G_2, k_1) to the power set of the set of solutions to (G_3, k_3). Specifically, each vertex subset (in a collection denoted by $\mathsf{Map}(X)$) of G_3 that is mapped to a solution X to (G_2, k_1) is the union of all "copies" of each vertex in X as well as at most $k_3 - d \cdot |X|$ many other vertices from G_3 so that there does not exist a vertex outside X having all of its copies chosen (Definition 29). We first assert that this mapping corresponds to a partition of the set of solutions to (G_3, k_3) (Lemma 30). Then, we turn to analyze the sizes of the mapped collections. Towards this, we begin with a simple proof that for every X is of size i, for some $i \in \{0, 1, \ldots, k_2\}$, the size of $|\mathsf{Map}(X)|$ is the same (denoted by w_i), captured by an explicit formula (Lemma 31). In particular, $x_3 = \sum_{i=0}^{k_2} y_i \cdot w_i$. Consider this equality as Equation (*).

The main property of the w_i's is that, for every $i \in \{0, 1, \ldots, k_2\}$, w_i is "significantly" larger than the sum of all w_j's for $j < i$ (proved in Lemma 32). In particular, based on Equation (*) and this property, we can derive, from x_3, the individual values of the y_i's. Specifically, this can be done by the following loop. For $i = 0, 1, 2, \ldots, k_2$, we let $y_i \leftarrow \lfloor x_3 / w_i \rfloor$, and update $x_3 \leftarrow x_3 - y_i \cdot w_i$. This computation can be performed efficiently (in polynomial time), since the w_i's can be computed efficiently by dynamic programming (Lemma 33). In turn, this computation is the main part of the procedure lift, presented in Section 5.3.

Proof of Theorem 2. At a high level, we follow the approach of [24] who give a polynomial kernel for Planar-$\mathcal{F}$ Deletion. Given an instance (G, k), we compute a modulator X using an approximation algorithm [24]. This modulator has size $k^{\mathcal{O}(1)}$, assuming that G has a $\mathcal{F}$-deletion set of size at most k. Next, we consider the components of $G - X$. A component C is *irrelevant*, if it is disjoint from every minimal $\mathcal{F}$-deletion set of size at most k. Using the properties of $\mathcal{F}$-free graphs, we obtain that all but $k^{\mathcal{O}(1)}$ components of $G - X$ are irrelevant. We delete all irrelevant components in the first phase of the reduction step. Let G' be the resulting graph.

The next reduction step, considers each component of $G' - X$. For each such component C, we observe that it is a *near-protrusion* [24], i.e. a subgraph that has constant-treewidth and after the removal of a $\mathcal{F}$-deletion set from G', has a constant sized boundary. We then apply several powerful results on *boundaried graphs*, to show that the information required to count the number of $\mathcal{F}$-deletion sets of size k' in G', for every $k' \leq k$, can stored in a compressed form using $k^{\mathcal{O}(1)}$ space.

Briefly, a boundaried graph is a graph H where a subset of vertices B are marked as boundary vertices. These boundary vertices are labeled with integers. Given two boundaried graphs H_1 and H_2, whose boundary vertices are labeled using the same set of integers, we can "glue" them to obtain a graph $H_1 \oplus H_2$, which is obtained by first taking a disjoint union of the two graphs and then identifying boundary vertices with the same label. Using the notion of boundaried graphs and gluing, we can define an equivalence relation, $\equiv_{\mathcal{F}}$ such that $H_1 \equiv_{\mathcal{F}} H_2$ if and only if for any other boundaried graph H_3, $H_1 \oplus H_3$ is $\mathcal{F}$-minor free $\iff$ $H_2 \oplus H_3$ is $\mathcal{F}$-minor free. It is known that this equivalence relation has finitely many equivalence class for any fixed $\mathcal{F}$.

Intuitively, our compression for a connected component C of $G' - X$, considers the effect of deleting a $\mathcal{F}$-deletion set S from G', and records the number of ways this can happen. Since C is a near protrusion, it has constant-treewidth and a constant size boundary in

$G - S$ that is a subset of $X \setminus S$. We treat $G[(V(C) \cup N(C)) \setminus S]$ as a boundaried graph with boundary $N(C) \setminus S$, and note that $N(C) \subseteq X$. Note that $G[(V(C) \cup N(C)) \setminus S]$ lies in an equivalence class $\mathcal{R}$ of $\equiv_{\mathcal{F}}$. Then, for each choice of $\mathcal{R}$, $N[C] \setminus S$ and $|S \cap V(C)|$ we record the number of subsets S_C of $V(C)$ such that $G[(V(C) \cup N(C)) \setminus S]$ with boundary $N[C] \setminus S$ forms a boundaried graph that lies in $\mathcal{R}$. We compute and store this information in a table T_C for each component C. We show that the number of such choices is bounded by $k^{\mathcal{O}(1)}$, and each entry of T_C can be computed polynomial time. We then argue that the information stored in the table is sufficient to compute $\mathsf{count}(k')$ which is the number of $\mathcal{F}$-deletion sets in G' of size at most k', for every $k' \leq k$. Note that computing $\mathsf{count}(k')$ takes time exponential in k.

The output of the reduce procedure for #Planar-$\mathcal{F}$ Deletion, given an instance (G, k), is a modulator X of size $k^{\mathcal{O}(1)}$ and a collection of tables $\{T_C\}$, one for each non-irrelevant component of $G - X$. Note that the size of the output is $k^{\mathcal{O}(1)}$. Next, the lift procedure is given the instance (G, k), the modulator X, the collection of tables T_C for each component of $G' - X$, and finally the values $\{\mathsf{count}(k') \mid k' \leq k\}$. The lift procedure first computes τ_{irr} which denotes the total number of vertices in the irrelevant components of $G - X$. Then, from $\{\mathsf{count}(k') \mid k' \leq k\}$ and τ_{irr} it is easy to count the total number of solutions of size at most k in G in polynomial time. The reduce and lift procedures together prove this theorem.

We now turn to discuss the negative results, starting with Theorem 5 and then continuing with Theorems 3 and 4.

Proof of Theorem 5. We start with the proof that #Min (s,t)-Cut (which is #P-hard [39]) SUM-cross-composes into #k-Min (s,t)-Cut. Suppose that we are given ℓ instances of #Min (s,t)-Cut, $(G_1, s_1, t_1), (G_2, s_2, t_2), \ldots, (G_\ell, s_\ell, t_\ell)$, where the size of a minimum (s_i, t_i)-cut in G_i is assumed to be equal to the size of a minimum (s_j, t_j)-cut in G_j, for every $i, j \in [\ell]$. (This assumption is justified by the more general definition of cross-compositions that makes use of equivalence relations.) Then, the reduction to a single instance (G, s, t) of is performed as follows: We take the disjoint union of the input graphs, and unify t_i with s_{i+1}, for every $i \in \{1, 2, \ldots, \ell - 1\}$; additionally, we let $s = s_1$ and $t = t_\ell$. With this construction at hand, it is easy to see that each minimum (s,t)-cut in G corresponds to a minimum (s_i, t_i)-cut in one of the G_i's, and vice versa. Thus, we derive that the number of minimum (s,t)-cuts in G equals the sum of the number of minimum (s_i, t_i)-cuts in G_i, over every $i \in \{1, 2, \ldots, \ell\}$. Moreover, the parameter k is trivially bounded from above by $\max_{i=1}^{\ell} |E(G_i)|$.

Having asserted that #k-Min (s,t)-Cut does not admit a polynomial compression under the SUM-conjecture, we transfer its hardness to the #k-Odd Cycle Transversal problem by the design of a *polynomial parameter transformation* (Definition 16). Suppose that we are given an instance (G, s, t) of #k-Min (s,t)-Cut where G is a connected graph. Then, we first turn G into a graph G_1 be subdividing each edge once. In particular, we thus derive that all paths in G_1 between vertices that correspond to vertices (rather than edges) in G are of even length. Next, we turn G_1 into a graph G_2 by replacing each vertex of G_1 that corresponds to a vertex of G by $k + 1$ copies (false twins). Intuitively, this will have the effect that no minimal solution being of size at most k to our instance of #k-Odd Cycle Transversal (defined immediately) will pick any vertex in G_2 that corresponds to a vertex in G (since we deal with edge-cuts, this property must be asserted for our proof of correctness). Complementary to this, we will (implicitly) prove that our instance has no solution of size smaller than k, so every solution of size at most k is of size exactly k and a minimal one. The last step of the reduction is to turn G_2 into a graph G' by adding two new adjacent vertices, x_i and y_i, for every $i \in \{1, 2, \ldots, k + 1\}$, and making all the x_i's adjacent

to all the copies of s, and all the y_i's adjacent to all the copies of t. With this construction of G' at hand (and keeping the parameter k unchanged), we are able to prove that: *(i)* every odd cycle in G' contains at least one path from a copy of s to a copy of t that corresponds to an (s,t)-path in G, and *(ii)* every (s,t)-path in G can be translated to some particular set of odd cycles in G' such that, to hit that set with at most k vertices, it only "makes sense" to pick vertices in G' that correspond to edges in G. From this, we are able to derive that the number of minimum (s,t)-cuts in G equals the number of odd cycles transversal of G' of size at most k.

Lastly, having asserted that #k-Odd Cycle Transversal does not admit a polynomial compression under the SUM-conjecture, we transfer its hardness to the #ℓ-Vertex Cover problem (where the parameter is k minus the LP-value) and the #m-Vertex Cover problem (where the parameter is k minus the maximum size of a matching) by the design of another polynomial parameter transformation. We remark that, since it always holds that $m \geq \ell$, the hardness for #m-Vertex Cover implies the hardness for #ℓ-Vertex Cover. While the transformation itself is the same as the known reduction from k-Odd Cycle Transversal to m-Vertex Cover (Lemma 3.10 in [14]), the analysis somewhat differs. In particular, for the correctness, we actually cannot use #k-Odd Cycle Transversal as the source problem, but only restricted instances of it, where for every odd cycle transversal S of size at most k, the removal of S from the input graph G yields a connected graph. Then, we are able to show that the number of odd cycle transversals of G of size at most k is exactly half the number of vertex covers of the output graph G' of size at most k'. (The parameter of the output instance, $k' - m$, equals k.)

Proof of Theorems 3 and 4. The proof of Theorem 3 follows the lines of, yet is not identical to, the proof of the analogous statement for OR-cross-composition. For example, one notable difference concerns the part of the proof where we need to define a problem whose solution is a function of solutions of another problem. While for OR-cross-compositions, the chosen function is the logical OR of the given solutions, for us the chosen function is a weighted summation of the given solutions with weights chosen so that, from the weighted sum, we can derive each individual solution (that is similar to the spirit of the lift procedure given as part of the proof of Theorem 1).

For the proof of Theorem 4, we prove that #Min (s,t)-Cut EXACT-cross-composes into #w-Min (s,t)-Cut. The reduction begins by taking the instance (G,s,t) built in the proof of the SUM-cross-composition discussed earlier. We note that the treewidth of G equals the maximum treewidth of G_i, over every $i \in \{1,2,\ldots,\ell\}$. However, recall that this construction only yields that the number of solutions to (G,s,t) (say, q) equals $\sum_{i=1}^{\ell} q_i$ where q_i is the number of solutions to (G_i,s_i,t_i). So, by knowing only q, we are not able to derive the individual q_i's (there can be exponentially many options for their values). So, we further modify the graph G to obtain a graph G' as follows. For the copy of each G_i in G, we add a $2^{m\cdot(\ell-1)}$ internally vertex-disjoint paths from s_i to t_i, where $m = 2\max_{j=1}^{\ell} |E(G_j)|$: $m(i-1)$ of these paths have three internal vertices, and the rest have one internal vertex. Notice that, to separate s_i and t_i in this "extended" copy of G_i, we need to pick at least one edge from each of the newly added paths, and we have two (resp., four) options for which edge to pick from each of the paths with one (resp., three) internal vertices. Having this insight in mind, we are able to show that the number of solutions to (G',s,t) (say, q') equals $\sum_{i=1}^{\ell} q_i \cdot 2^{m(i-1)+m(\ell-1)}$. In particular, the coefficient of each q_i is "significantly" larger than the sum of the coefficients of all q_j, $j < i$. In turn, this allows us to derive, from q', the individual values of the q_i's (similarly, in this part, to the corresponding parts of the proofs of Theorem 1 and 3). Further, we show that the addition of the aforementioned paths does not increase the treewidth of the graph (unless it was smaller than 2).

Due to space constraints, we only provide the details of the proof of Theorem 1. The complete details of all other proofs can be found in the full version.

3 Preliminaries

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For $n \in \mathbb{N}$, $[n] = \{1, 2, \ldots, n\}$. Given a universe U, $2^U = \{A : A \subseteq U\}$.

Graph Notation. Throughout the paper, we consider finite, simple, undirected graphs. Given a graph G, let $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. Given a subset $U \subseteq E(G)$, let $G - U$ denote the graph on vertex set $V(G)$ and edge set $E(G) \setminus U$. We say that $S \subseteq V(G)$ *covers* $U \subseteq E(G)$ if for every edge $\{u, v\} \in U$, $S \cap \{u, v\} \neq \emptyset$. A *vertex cover* of G is a subset $S \subseteq V(G)$ that covers $E(G)$. The set S is said to be *minimal* if every subset of it is not a vertex cover of G. An *independent set* of G is a subset $S \subseteq V(G)$ such that $E(G) \cap \{\{u, v\} : v \in S\} = \emptyset$. A *matching* in G is a subset $S \subseteq E(G)$ such that no two edges in S share an endpoint. Let $\mu(G)$ denote the maximum size of a matching in G. Given two distinct vertices $s, t \in V(G)$, an *(s,t)-cut in G* is a subset $S \subseteq E(G)$ such that in $G - S$, the vertices s and t belong to different connected components. An (s,t)-cut in G is *minimum* if there does not exist an (s,t)-cut in G of smaller size. Given a subset $U \subseteq V(G)$, let $G[U]$ denote the subgraph of G induced by U, and let $G - U$ denote $G[V(G) \setminus U]$. An *odd cycle transversal of G* is a subset $S \subseteq V(G)$ such that $G - S$ does not contain any odd cycle (i.e., a cycle with an odd number of vertices, or, equivalently, of edges). The subdivision of an edge $\{u, v\} \in E(G)$ is the operation that removes $\{u, v\}$ from G, adds a new vertex x to G, and adds the edges $\{u, x\}$ and $\{v, x\}$ to G. Given a graph H, we write $H \subseteq G$ to indicate that H is a subgraph of G. A graph G is *bipartite* if there exists a partition (X, Y) of $V(G)$ such that $E(G) \subseteq \{\{x, y\} : x \in X, y \in Y\}$, that is, X and Y are independent sets. Note that a graph G is bipartite if and only if it does not contain any odd cycle [16]. We say that a graph H is a *minor* of a graph G if there exists a series of vertex deletions, edge deletions and edge contractions in G that yields H. We say that G is a *planar graph* if it can be drawn on the Euclidean plane so that its edges can intersect only at their endpoints.

Treewidth is a structural parameter indicating how much a graph resembles a tree:

▶ **Definition 6.** *A* tree decomposition *of a graph G is a pair $\mathcal{T} = (T, \beta)$ of a tree T and $\beta : V(T) \to 2^{V(G)}$, such that*

1. *for any edge $\{x, y\} \in E(G)$ there exists a node $v \in V(T)$ such that $x, y \in \beta(v)$, and*
2. *for any vertex $x \in V(G)$, the subgraph of T induced by the set $T_x = \{v \in V(T) : x \in \beta(v)\}$ is a non-empty tree.*

The width *of (T, β) is $\max_{v \in V(T)}\{|\beta(v)|\} - 1$. The* treewidth *of G, denoted by $\mathsf{tw}(G)$, is the minimum width over all tree decompositions of G.*

Problems and Counting Problems. A *decision problem* (or *problem* for short) is a language $P \subseteq \Sigma^\star$. Here, Σ is a finite alphabet, and, without loss of generality, we can assume that $\Sigma = \{0, 1\}$. Often, some strings in $\Sigma^\star$ are "irrelevant" to P (specifically, they clearly do not belong to P) – e.g., when P concerns graphs and a given string does not encode a graph; so, the term *instance of P* is loosely used for strings that are relevant to P in some such natural sense. An *algorithm for P* is a procedure that, given $x \in \Sigma^\star$, determines whether $x \in P$. We say that an instance x of a problem P is *equivalent* to an instance x' of a problem Q if: $x \in P$ if and only if $x' \in Q$. A *counting problem* is a mapping F from $\Sigma^\star$ to $\mathbb{N}_0$. As before, the term *instance of F* is loosely used – while one still needs to define the mapping of

"irrelevant" strings, the consideration of this mapping will be immaterial to us. An *algorithm for* F is a procedure that, given $x \in \Sigma^\star$, outputs $F(x)$. A counting problem F is a *counting version* of a problem P if, for every $x \in \Sigma^\star$, $x \in P$ if and only if $F(x) \geq 1$. When we refer to "the" counting version of a problem P, we consider the counting version of P whose choice (among all counting versions of P) is widely regarded the most natural one, and it is denoted by $\#P$.

Parameterized Complexity. We start with the definition of a parameterized problem.

▶ **Definition 7** (Parameterized Problem). *A* parameterized problem *is a language* $P \subseteq \Sigma^\star \times \mathbb{N}_0$, *where* Σ *is a fixed, finite alphabet. For an instance* $(x, k) \in \Sigma^\star \times \mathbb{N}_0$, k *is called the* parameter.

An *algorithm for* P is a procedure that, given $(x, k) \in \Sigma^\star \times \mathbb{N}_0$, determines whether $(x, k) \in P$. We say that P is *fixed-parameter tractable (FPT)* if there exists an algorithm for P that runs in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ where f is some computable function of k. Such an algorithm is called a *fixed-parameter algorithm*. The main tool to assert that one problem is in FPT based on an already known membership of another problem in FPT is the design of a PPT, defined as follows.

▶ **Definition 8** (PPT). *Let* $P, Q \subseteq \Sigma^\star \times \mathbb{N}_0$ *be two parameterized problems. A polynomial-time algorithm* A *is a* polynomial parameter transformation (PPT) *from* P *to* Q *if, given an instance* (x, k) *of* P, A *outputs an equivalent instance* (x', k') *of* Q *(i.e.,* $(x, k) \in P$ *if and only if* $(x', k') \in Q$*) such that* $k' \leq p(k)$ *for some polynomial function* p.

A companion notion of FPT is that of a compression or a kernelization, defined as follows.

▶ **Definition 9** (Compression and Kernelization). *Let* P *and* Q *be two parameterized problems. A* compression *(or* compression algorithm *for* P *is a polynomial-time procedure that, given an instance* (x, k) *of* P, *outputs an equivalent instance* (x', k') *of* Q *where* $|x'|, k' \leq f(k)$ *for some computable function* f. *Then, we say that* P *admits a* compression of size $f(k)$. *When* f *is polynomial, then we say that* P *admits a* polynomial compression. *Further, when* $P = Q$, *we refer to compression also as* kernelization.

Now, we state two central propositions that concern kernelization.

▶ **Proposition 10** ([7]). *Let* P *be a parameterized problem that is decidable. Then,* P *is FPT if and only if it admits a kernel.*

▶ **Proposition 11** (Folklore; See, e.g., Theorem 15.15 in [14]). *Let* P, Q *be two parameterized problems such that there exists a PPT from* P *to* Q. *If* Q *admits a polynomial compression, then* P *admits a polynomial compression.*

Towards the statement of the main tool to refute the existence of polynomial compressions (and, hence, also polynomial kernels) for specific problems, we state the two following definitions.

▶ **Definition 12** (Polynomial Equivalence Relation). *An equivalence relation* R *on a set* $\Sigma^\star$ *is a* polynomial equivalence relation *if the following conditions are satisfied:*

- *There exists an algorithm that, given strings* $x, y \in \Sigma^\star$, *resolves whether* $x \equiv_R y$ *in time polynomial in* $|x| + |y|$.
- *The relation* R *restricted to the set* $\Sigma^{\leq n}$ *has at most* $p(n)$ *equivalence classes, for some polynomial function* p.

▶ **Definition 13** (OR-Cross-Composition). *Let $P \subseteq \Sigma^\star$ be a problem and $Q \subseteq \Sigma^\star \times \mathbb{N}_0$ be a parameterized problem. We say that P* OR-cross-composes *into Q if there exists a polynomial equivalence relation R and an algorithm A, called an* OR-cross-composition*, satisfying the following conditions. The algorithm A takes as input a sequence of strings $x_1, x_2, \ldots, x_t \in \Sigma^\star$ that are equivalent with respect to R, runs in time polynomial in $\sum_{i=1}^{t} |x_i|$, and outputs one instance $(y, k) \in \Sigma^\star \times \mathbb{N}_0$ such that:*

- *$k \leq p(\max_{i=1}^{t} |x_i| + \log t)$ for some polynomial function p, and*
- *$(y, k) \in Q$ if and only if there exists at least one index $i \in [t]$ such that $x_i \in P$.*

Now, we state the main tool to refute the existence of polynomial compressions for specific problems.

▶ **Proposition 14** ([3, 4]). *Assume that an NP-hard problem P OR-cross-composes into a parameterized problem Q. Then, Q does not admit a polynomial compression, unless coNP $\subseteq$ NP/poly.*

We remark that an analogous proposition, where OR is replaced by AND, has been proved in [18].

We proceed to the definition of a parameterized counting problem.

▶ **Definition 15** (Parameterized Counting Problem). *A* parameterized counting problem *is a mapping F from $\Sigma^\star \times \mathbb{N}_0$ to $\mathbb{N}_0$.*

An *algorithm for F* is a procedure that, given $(x, k) \in \Sigma^\star \times \mathbb{N}_0$, outputs $F(x, k)$. As before, we say that P is *fixed-parameter tractable (FPT)* if there exists an algorithm for P that runs in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ where f is some computable function of k. A parameterized counting problem F is a *counting version* of a parameterized problem P if, for every $(x, k) \in \Sigma^\star \times \mathbb{N}_0$, $(x, k) \in P$ if and only if $F(x, k) \geq 1$. When we refer to "the" counting version of a parameterized problem P, we consider the counting version of P whose choice (among all counting versions of L) is widely regarded the most natural one, and it is denoted by $\#P$.

▶ **Definition 16** (PPT (Counting Version)). *Let $P, Q : \Sigma^\star \times \mathbb{N}_0 \to \mathbb{N}_0$ be two parameterized counting problems. A pair of polynomial-time procedures* (reduce, lift) *is a* polynomial parameter transformation (PPT) *from P to Q such that:*

- *Given an instance (x, k) of P,* reduce *outputs an instance (x', k') of Q such that $k' \leq p(k)$ for some polynomial function p.*
- *Given an instance (I, k) of P, the instance (I', k') that is the output of* reduce *on (I, k), and x' such that $P(I', k') = x'$,* lift *outputs x such that $Q(I, k) = x$.*

The main concept to show that a problem is unlikely to be FPT is the one of parameterized reductions analogous to those employed in classical complexity. Here, the concept of W[1]-hardness replaces the one of NP-hardness, and for reductions we need not only construct an equivalent instance in FPT time, but also ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original one. If there exists such a reduction transforming a parameterized problem known to be W[1]-hard to another parameterized problem P, then the problem P is W[1]-hard as well. Central W[1]-hard problems include, for example, deciding whether a nondeterministic single-tape Turing machine accepts within k steps, Clique parameterized be solution size, and Independent Set parameterized by solution size. Naturally, #W[1]-hardness is the concept analogous to W[1]-hardness in the realm of parameterized counting problems. For more information on W[1]-hardness and #W[1]-hardness, we refer to [14, 10, 17].

Problem Definitions. The counting problems studied in this paper are defined as follows.

- #k-Vertex Cover (#k-Minimal Vertex Cover): Given a graph G and a non-negative integer k, output the number of vertex covers (minimal vertex covers) of G of size at most k. Here, the parameter is k.
- #ℓ-Vertex Cover and #m-Vertex Cover: Defined as #k-Vertex Cover with the exception that the parameters ℓ and m are $k - \mathsf{LP}_{\mathsf{VC}}(G)$ and $k - \mu(G)$, respectively. Here, $\mathsf{LP}_{\mathsf{VC}}(G)$ denotes the optimum of the (standard) linear program that corresponds to Vertex Cover (see [14], Section 3.4).
- #k-Planar $\mathcal{F}$-Deletion: Let $\mathcal{F}$ be a finite set of connected graphs that contains at least one planar graph. Given a graph G and a non-negative integer k, output the number of subsets $S \subseteq V(G)$ of size at most k such that $G - S$ does not contain any graph from $\mathcal{F}$ as a minor. We remark that the #k-Planar $\mathcal{F}$-Deletion problem encompasses (based on different choices of $\mathcal{F}$) various other problems, such as #k-Vertex Cover, #k-Vertex Cover and #k-Vertex Cover.
- #k-Min (s,t)-Cut: Given a graph G and two distinct vertices $s, t \in V(G)$, output the number of minimum (s,t)-cuts in G. Here, the parameter k is the size of a minimum (s,t)-cut in G.
- #w-Min (s,t)-Cut: Defined as #k-Min (s,t)-Cut with the exception that the parameter w is the treewidth of G.
- #k-Odd Cycle Transversal: Given a graph G and a non-negative integer k, output the number of odd cycle transversal of G of size at most k. Here, the parameter is k.

4 Kernelization of Counting Problems

We define the notion of kernelization for counting problems as follows.

▶ **Definition 17** (Compression of Counting Problem). *Let P and Q be two parameterized counting problems. A* compression *(or* compression algorithm*) of P into Q is a pair* $(\mathsf{reduce}, \mathsf{lift})$ *of two polynomial-time procedures such that:*

- *Given an instance (x,k) of P,* reduce *outputs an instance (x',k') of Q where $|x'|, k' \leq f(k)$ for some computable function f.*
- *Given an instance (x,k) of P, the instance (x',k') that is the output of* reduce *on (x,k), and s' such that $P(x',k') = s'$,* lift *outputs s such that $Q(x,k) = s$.*

When Q is immaterial, we refer to a compression of P into Q only as a compression of P.

When $P = Q$, a compression is called a *kernel*. The measure $f(k)$ is termed the *size* of the compression. When f is a polynomial function, then the compression (or kernel) is said to be a *polynomial compression* (*polynomial kernel*). The following observation is immediate.

▶ **Observation 18.** *Let P be a parameterized (decision) problem that does not admit a polynomial kernel (or compression). Then, no counting version of P admits a polynomial kernel (or compression).*

Hence, we only consider parameterized counting problems whose decisions versions are either in P, or, if they are not, then they at least admit polynomial kernels. Specifically, Min (s,t)-Cut is in P [8], and polynomial kernels for k-Vertex Cover, ℓ-Vertex Cover (and m-Vertex Cover), k-Planar $\mathcal{F}$-Deletion, and k-Odd Cycle Transversal can be found in [6], [31], [24] and [31] respectively.

Throughout the paper, whenever we discuss a compression, we suppose (implicitly) that the compression is into a well-behaved problem, defined as follows:

▶ **Definition 19** (Well-Behaved Problem). *Let $P : \Sigma^\star \times \mathbb{N}_0 \to \mathbb{N}_0$ be a parameterized counting problem. Then, P is* well-behaved *if there exists a polynomial-time algorithm that, given $n \in \mathbb{N}$ in unary, outputs $N \in \mathbb{N}$ in binary with the following property: for every $(x, k) \in \Sigma^\star \times \mathbb{N}_0$ of size at most n, $P(x, k) \leq N$.*

We remark that, essentially, every "natural" parameterized counting problem (that we know of) is well-behaved.

▶ **Lemma 20.** *Let P be a parameterized counting problem that is solvable in finite time. Then, P is FPT if and only if it admits a kernel.*

Proof. The proof follows lines similar to that of Proposition 10 (we also [35]). For the sake of completeness, we present the details in the full version of the paper. ◀

Due to Lemma 20, every counting problem that is #W[1]-hard (and which is solvable in finite time) does not admit any kernel, even not of exponential (or worse) size. We remark that $\#k$-Min(s, t)-Cut is shown to be FPT by Berge et al. [2], and $\#w$-Min(s, t)-Cut is can be shown to be FPT by the usage of straightforward dynamic programming over tree decompositions (see, e.g. [14]).

▶ **Lemma 21.** *Let P, Q be two parameterized counting problems such that there exists a PPT from P to Q. If Q admits a polynomial compression, then P admits a polynomial compression.*

Proof. The proof follows lines similar to that of Proposition 11. For the sake of completeness, we present the details in the full version of the paper. ◀

We remark that in the full version of the paper, we discuss two new notions of a cross-composition for proofs of the unlikely existence of polynomial compressions for parameterized counting problems.

5 Polynomial Kernel for #Vertex Cover

The purpose of this section is to prove the following theorem.

▶ **Theorem 1.** *$\#k$-Vertex Cover admits a polynomial kernel.*

Towards the proof of this theorem, we first develop the reduction procedure. Then, we discuss properties of the reduced instance. Afterwards, we present the lifting procedure and conclude the correctness of the kernel. For the sake of brevity, throughout this section, we write #Vertex Cover instead of $\#k'$-Vertex Cover (where k' is the current value of the parameter).

5.1 Reduction Procedure and a Corollary for Minimal Vertex Covers

We define the procedure reduce as follows. Given an instance (G, k) of #Vertex Cover, we will first exhaustively apply the following reduction rule, known as Buss Rule [6] (see also [14]):

▶ **Definition 22** (Buss Rule). *If G contains a vertex v of degree at leas $k + 1$, then update $G \leftarrow G - \{v\}$ and $k \leftarrow k - 1$.*

Let (G_1, k_1) be the instance of #Vertex Cover obtained after exhaustive application of Buss Rule. Let G_2 be graph obtained from G_1 by the removal of all isolated vertices, and denote $k_2 = k_1$. Let $\mathcal{S}$ ($\mathcal{S}_1$, $\mathcal{S}_2$) denote the set of vertex covers of G (G_1, G_2) of size at most k (k_1, k_2) . Let $n_1 = |V(G_1)|$ and $n_2 = |V(G_2)|$. We have the following known proposition:

▶ **Proposition 23** ([6, 14]). *The three following properties hold:*
1. $\mathcal{S} = \{S_1 \cup (V(G) \setminus V(G_1)) : S_1 \in \mathcal{S}_1\}$.
2. *If* $|E(G_2)| > (k_2)^2$, *then* G *does not contain any vertex cover of size at most* k.
3. *Else,* $|E(G_2)| \leq (k_2)^2$, *then* $|V(G_2)| \leq 2(k_2)^2$.

Let x (resp., x') be the number of vertex covers (resp., minimal vertex covers) of G of size at most k, let x_1 (resp., x'_1) be the number of vertex covers (resp., minimal vertex covers) of G_1 of size at most k_1, and let x_2 (resp., x'_2) be the number of vertex covers (resp., minimal vertex covers) of G_2 of size at most k_2 . Then, due to the first item of Proposition 23 and since no minimal vertex cover can contain isolated vertices, we have the following corollary.

▶ **Corollary 24.** *The following equalities hold:* $x = x_1$ *and* $x' = x'_1 = x'_2$.

Given this corollary, we can already conclude a polynomial kernel for the variant of #Vertex Cover termed #k-Minimal Vertex Cover. (The challenge, dealt with in the rest of Section 5, would be to derive a polynomial kernel for#Vertex Cover .)

▶ **Theorem 25.** #k-Minimal Vertex Cover *admits a kernel of size* $\mathcal{O}(k^2)$.

Proof. Given an instance (G, k) of #k-Minimal Vertex Cover, the procedure reduce$'$ outputs: *(i)* (G_2, k_2) if $|E(G_2)| \leq (k_2)^2$, and *(ii)* $(G' = (\{u, v\}, \{\{u, v\}\}), k' = 0)$ otherwise. Observe that the procedure runs in polynomial time, and, due to the third item of Proposition 23, the size of the output is bounded by $\mathcal{O}(k^2)$.

Given (G, k), the output of reduce$'$, and the solution x'_2 to this output, the procedure lift$'$ returns x'_2. Observe that the procedure runs in polynomial time, and from the second item of Proposition 23 and Corollary 24, we know that $x' = x'_2$ and hence the procedure is correct. ◀

Unfortunately, for #Vertex Cover, we cannot simply output (G_2, k_2). In particular, observe that different vertex covers of G_1 of size at most k_1 might contain different numbers of vertices that are isolated in G_1, and hence the knowledge of x_2 alone is insufficient in order to deduce x_1 (and x).

We proceed to modify G_2 in order to define the graph that will be the output of the reduction

▶ **Definition 26.** *Let* $d = n_2$ *and* $t = d + dk_2 + 2(dk_2)^2$. *Then, let* G_3 *be the graph whose vertex set* $\{v_i : v \in V(G_2), i \in [d]\} \cup T$, *where* T *is a set of* t *new vertices, and whose edge set is* $\{\{u_i, v_j\} : \{u, v\} \in E(G_2), i, j \in [d]\}$. *Additionally, let* $k_3 = d \cdot k_2$.

That is, G_3 is the result of the replacement of every vertex of G_2 by d copies (false twins) of that vertex and the addition of t new vertices. We are now ready to define reduce.

▶ **Definition 27** (Procedure reduce). *Given an instance* (G, k) *of* #Vertex Cover, *the procedure* reduce *outputs:* (i) (G_3, k_3) *if* $|E(G_2)| \leq (k_2)^2$, *and* (ii) $(G' = (\{u, v\}, \{\{u, v\}\}), k' = 0)$ *otherwise.*

Due to the third item of Proposition 23, we have the following immediate observation.

▶ **Observation 28.** reduce *runs in polynomial time, and the size of its output is bounded by* $k^{\mathcal{O}(1)}$.

5.2 Properties of the Reduced Instance

For every $i \in \{0, 1, \ldots, k_2\}$, let $\mathcal{S}_2^i$ be the set of vertex covers of G_2 of size exactly i. Then, $\mathcal{S}_2 = \bigcup_{i=0}^{k_2} \mathcal{S}_2^i$ is the set of vertex covers of size at most k_2 of G_2. Let $\mathcal{S}_3$ be the set of vertex covers of G_3 of size at most k_3. Let $n_3 = |V(G_3)|$. We say that a subset $U \subseteq V(G_3)$ is *valid* if there does not exist $v \in V(G_2)$ such that $\{v_1, v_2 \ldots, v_d\} \subseteq U$. We proceed to define the following mappings.

▶ **Definition 29** (Mappings map and Map)**.** *The mappings* $\mathsf{map} : \mathcal{S}_2 \to 2^{V(G_3)}$ *and* $\mathsf{Map}\mathcal{S}_2 \to 2^{2^{V(G_3)}}$ *are defined as follows.*

- *Given* $X \in \mathcal{S}_2$*, let* $\mathsf{map}(X) = \{v_j : v \in X, j \in [d]\}$.
- *Given* $X \in \mathcal{S}_2$*, let* $\mathsf{Map}(X) = \{\mathsf{map}(X) \cup U : U \text{ is valid}, U \cap \mathsf{map}(X) = \emptyset, |U| \leq k_3 - |\mathsf{map}(X)|\}$.

We have the following lemma regarding the vertex covers of G_3.

▶ **Lemma 30.** *We have that* (i) $\mathcal{S}_3 = \bigcup_{X \in \mathcal{S}_2} \mathsf{Map}(X)$*, and* (ii) *for distinct* $X, Y \in \mathcal{S}_2$*,* $\mathsf{Map}(X) \cap \mathsf{Map}(Y) = \emptyset$.

Proof. We first prove the correctness of the first item. On the one hand, consider some $A \in \mathcal{S}_3$. Let $X = \{v \in V(G_2) : \{v_1, v_2, \ldots, v_d\} \subseteq X\}$, and $U = A \setminus X$. We claim that $X \in \mathcal{S}_2$. Since $|A| \leq k_3$ (because $A \in \mathcal{S}_3$) and $k_3 = d \cdot k_2$, it follows that $|X| \leq k_2$. Moreover, consider an edge $\{u, v\} \in E(G_2)$. If there exists $u_i, v_j \in V(G_3)$ such that $\{u_i, v_j\} \cap A = \emptyset$, then we have a contradiction since A is a vertex cover of G_3. Hence, $\{u, v\} \cap X \neq \emptyset$. In turn, we derive that X is a vertex cover of G_2, which yields that $X \in \mathcal{S}_2$. Now, notice that, by Definition 29, $A = \mathsf{map}(X) \cup U$ and $\mathsf{map}(X) \cup U \in \mathsf{Map}(X)$. So, $A \in \bigcup_{X \in \mathcal{S}_2} \mathsf{Map}(X)$.

On the other hand, let $B \in \bigcup_{X \in \mathcal{S}_2} \mathsf{Map}(X)$. So, $B \in \mathsf{Map}(X)$ for some $X \in \mathcal{S}_2$. By Definition 29, this implies that $B = \mathsf{map}(X) \cup U$ for some valid subset $U \subseteq V(G_3)$ disjoint from $\mathsf{map}(X)$, and $|B| \leq k_3$. So, to derive that $B \in \mathcal{S}_3$, it suffices to argue that $\mathsf{map}(X)$ is a vertex cover of G_3. To this end, consider some edge $\{u_i, v_j\} \in E(G_3)$. Then, $\{u, v\} \in E(G_2)$. Because X is a vertex cover of G_2, we have that $\{u, v\} \cap X \neq \emptyset$. However, by Definition 29, this implies that $\{u_i, v_j\} \cap \mathsf{map}(X) \neq \emptyset$. Thus, the proof of the first item of the lemma is complete.

For the second item of the lemma, consider some distinct $X, Y \in \mathcal{S}_2$. Without loss of generality, suppose that $|X| \geq |Y|$. So, there exists $v \in V(G_2)$ such that $v \in X \setminus Y$, and, hence, $\{v_1, v_2, \ldots, v_d\} \subseteq \mathsf{map}(X)$ while $\{v_1, v_2, \ldots, v_d\} \cap \mathsf{map}(Y) = \emptyset$. So, since a valid set cannot contain $\{v_1, v_2, \ldots, v_d\}$, we derive that $\{v_1, v_2, \ldots, v_d\} \setminus A \neq \emptyset$ for every $A \in \mathsf{Map}(Y)$. However, since $\{v_1, v_2, \ldots, v_d\} \setminus A = \emptyset$ for every $A \in \mathsf{Map}(X)$, it follows that $\mathsf{Map}(X) \cap \mathsf{Map}(Y) = \emptyset$. ◀

We consider the sizes of the sets assigned by Map in the following lemma.

▶ **Lemma 31.** *For every* $X \in \mathcal{S}_2^i$ *for* $i \in \{0, 1, \ldots, k_2\}$*, it holds that* $|\mathsf{Map}(X)| = w_i$*, where*

$$w_i = \sum_{(a^\star, a_1, a_2, \ldots, a_{n_2 - i}) \in W_i} \binom{t}{a^\star} \prod_{j=1}^{n_2 - i} \binom{d}{a_j}, \quad \text{and}$$

$$W_i = \{(a^\star, a_1, a_2, \ldots, a_{n_2 - i}) : a^\star + \sum_{j=1}^{n_2 - i} a_j \leq k_3 - d \cdot i, a^\star \leq t, \text{and for each } j \in [n_2 - i], a_j \in \{0, 1, \ldots, d-1\}\}.$$

Towards the proof of this lemma and a latter lemma, for every $r \in \{0, 1, \ldots, k_3 - i \cdot d\}$, let us denote $W_i^r = \{(a^\star, a_1, a_2, \ldots, a_{n_2-i}) : a^\star + \sum_{j=1}^{n_2-i} a_j = r, a^\star \leq t, \text{and for each } j \in [n_2 - i], a_j \in \{0, 1, \ldots, d-1\}\}$, and $w_i^r = \sum_{(a^\star, a_1, a_2, \ldots, a_{n_2-i}) \in W_i^r} \binom{t}{a^\star} \prod_{j=1}^{n_2-i} \binom{d}{a_j}$.

Proof of Lemma 31. Let $X \in \mathcal{S}_2^i$. So, we need to count the number of subsets $U \subseteq V(G_3)$ such that U is valid, $U \cap \mathsf{map}(X) = \emptyset$, and $|U| \leq k_3 - |\mathsf{map}(X)|$. Observe that $|\mathsf{map}(X)| = d \cdot i$. So, because we demand that $U \cap \mathsf{map}(X) = \emptyset$, every choice of U corresponds to the choice of some $r \leq k_3 - d \cdot i$ vertices from $V(G_3) \setminus \mathsf{map}(X)$ such that the resulting set would be valid. In turn, every such choice, for a specific r, corresponds to the choice of *(a)* how many vertices to pick from T and how many vertices (a number between 0 and $d-1$, due to validity) to pick from $\{v_1, v_2, \ldots, v_d\}$ for every $v \notin X$, so that in total we pick r vertices, and *(b)* given a choice of type *(a)*, the choice of which specific vertices to pick from T and which specific vertices to pick from $\{v_1, v_2, \ldots, v_d\}$ for every $v \notin X$. Clearly, we have a natural 1-to-1 correspondence between the the choices of type *(a)* and the vectors in W_i^r. Then, given a choice of such a vector $(a^\star, a_1, a_2, \ldots, a_{n_2-i})$, we have $\binom{t}{a^\star} \prod_{j=1}^{n_2-i} \binom{d}{a_j}$ choices of type *(b)*. Considering all choices for r, we attain the formula stated in the lemma. ◀

In particular, we prove that the sizes in Lemma 31 satisfy the following.

▶ **Lemma 32.** *For every $i \in \{0, 1, \ldots, k_2\}$,*

$$w_i > \sum_{j=i+1}^{k_2} \binom{n_2}{j} \cdot w_j.$$

Proof. Fix $i \in \{0, 1, \ldots, k_2\}$. First, observe that $w_p \geq w_q$ for all $p, q \in \{0, 1, \ldots, k_2\}$ such that $p \leq q$, and $\binom{n_2}{j} \leq 2^{n_2}$ for all $j \in \{0, 1, \ldots, k_2\}$. Hence, it suffices to prove that $w_i \geq k_2 \cdot 2^{n_2} \cdot w_{i+1}$. For this purpose, notice that $n_3 = dn_2 + t$. Additionally, on the one hand, for all $i' \in \{0, 1, \ldots, k_2\}$,

$$w_{i'} \leq k_3 \cdot \binom{n_3 - di'}{k_3 - di'} = dk_2 \cdot \binom{d(n_2 - i') + t}{d(k_2 - i')}.$$

We refer to this inequality as Inequality (1). To see its correctness, note that $w_{i'}^r$ is maximum when r is maximum (restricted to $\{0, 1, \ldots, k_3 - di'\}$), i.e., when $r = k_3 - di' \leq k_3$. Hence, $w_{i'} \leq k_3 \cdot w_{i'}^{k_3 - di'}$. Now, observe that $w_{i'}^{k_3-di'}$ corresponds to the number of choices of $k_3 - di'$ elements out of a universe of size $n_3 - di'$ that satisfy particular restrictions. Specifically, we have a partition of the universe into $n_2 - i' + 1$ parts – one of size t and the others of size d – and we can pick at most $d-1$ elements from each of the parts of size d. In particular, this simply means that $w_{i'}^{k_3-di'}$ is bounded from above by the number of choices of $k_3 - di'$ elements out of a universe of $n_3 - di'$ elements, which is $\binom{n_3-di'}{k_3-di'}$. Thus, Inequality (1) is correct.

On the other hand,

$$w_{i'} \geq \binom{n_3 - di' - n_2}{k_3 - di'} = \binom{d(n_2 - i') + t - n_2}{d(k_2 - i')}.$$

We refer to this inequality as Inequality (2). To see its correctness, note that $w_{i'} \geq w_{i'}^r$ for all $r \in \{0, 1, \ldots, k_3 - di'\}$. So, in particular, $w_{i'} \geq w_{i'}^{k_3-di'}$. Recall the combinatorial interpretation of $w_{i'}^{k_3-di'}$ discussed above for the correctness of Inequality (1). Now, out of that universe, suppose that we remove (arbitrarily) one element from each of the parts of size d – so, in total, we remove $n_2 - i'$ elements. Then, we remove i' additional elements.

Hence, we remain with a universe of size $n_3 - di' - n_2$. However, every choice of $k_3 - di'$ elements from this universe satisfies the particular restrictions stated in the aforementioned combinatorial interpretation. Hence, $w_{i'}^{k_3-di'}$ is bounded from below by the number of choices of $k_3 - di'$ elements out of a universe of $n_3 - di' - n_2$ elements, which is $\binom{n_3-di'-n_2}{k_3-di'}$. Thus, Inequality (2) is correct.

Hence, having Inequality (2) and since $d = n_2$,

$$w_i \geq \binom{d(n_2-i)+t-n_2}{d(k_2-i)}.$$
$$= \frac{(d(n_2-i)+t-d(k_2-i))(d(n_2-i)+t-d(k_2-i)-1)\cdots(d(n_2-i)+t-d(k_2-i)-n_2+1)}{(d(k_2-i))(d(k_2-i)-1)\cdots(d(k_2-i)-n_2+1)}$$
$$\cdot\binom{d(n_2-i)+t-d}{d(k_2-i)-d}.$$

Recall that $t = d + dk_2 + 2(dk_2)^2$. So, for all $j \in [d]$, $d(n_2 - i) + t - d(k_2 - i) - j + 1 \geq 2(dk_2)^2 \geq 2dk_2 \cdot (d(k_2 - i) - j + 1)$. In particular, we derive that

$$\frac{(d(n_2-i)+t-d(k_2-i))(d(n_2-i)+t-d(k_2-i)-1)\cdots(d(n_2-i)+t-d(k_2-i)-n_2+1)}{(d(k_2-i))(d(k_2-i)-1)\cdots(d(k_2-i)-n_2+1)}$$
$$\geq (2dk_2)^{n_2} > d(k_2)^2 \cdot 2^{n_2}.$$

Hence, the calculation above implies that

$$\begin{aligned} w_i &> d(k_2)^2 \cdot 2^{n_2} \cdot \binom{d(n_2-i)-d}{d(k_2-i)-d} \\ &\geq k_2 \cdot 2^{n_2} \cdot w_{i+1}, \end{aligned}$$

where the last inequality follows from Inequality (1). As discussed earlier, this completes the proof. ◀

5.3 Procedure lift and Proof of Theorem 1

We start with a computation of the values w_i, $i \in \{0, 1, \ldots, k_2\}$, defined in Lemma 31.

▶ **Lemma 33.** *There exists a polynomial-time algorithm that, given $i \in \{0, 1, \ldots, k_2\}$ and having t, d, k_2 and n_2 at hand, outputs w_i. Here, the input numbers are encoded in unary, and the output number is encoded in binary.*

Proof. Observe that $w_i = \sum_{r=0}^{k_3-id} w_i^r$. Hence, for the proof, it suffices to fix some $r \in \{0, 1, \ldots, k_2 - id\}$, and show how to compute w_i^r in polynomial time. Now, denote $\ell = n_2 - i$, $q = d - 1$, and

$$\widehat{w}_i^p = \sum_{\substack{(a_1,a_2,\ldots,a_\ell) \\ \text{s.t. } \sum_{j=1}^{\ell} a_j = p, \text{ and } \forall j\in[\ell], a_j \in \{0,1,\ldots,q\}}} \prod_{j=1}^{\ell} \binom{d}{a_j}.$$

Then, $w_i^r = \sum_{a^\star=0}^{t} \binom{t}{a^\star} \widehat{w}_i^{r-a^\star}$. So, for the proof, it suffices to fix some $a^\star \in \{0, 1, \ldots, t\}$, and show how to compute $\widehat{w}_i^p$, for $p = r - a^\star$, in polynomial time.

In what follows, we employ dynamic programming to compute $\widehat{w}_i^p$. To this end, for every $\ell' \in [\ell]$ and $p' \in \{0, 1, \ldots, \min(p, \ell' \cdot q)\}$, we allocate a table entry $\mathfrak{M}[\ell', p']$. We define (for the analysis):

$$W_{\ell',p'} = \sum_{\substack{(a_1,a_2,\ldots,a_{\ell'}) \\ \text{s.t. } \sum_{j=1}^{\ell'} a_j = p', \text{ and } \forall j\in[\ell'], a_j \in \{0,1,\ldots,q\}}} \prod_{j=1}^{\ell'} \binom{d}{a_j}.$$

The purpose of $\mathfrak{M}[\ell', p']$ would be to store $W_{\ell',p'}$. Then, since $\widehat{w}_i^p = W_{\ell,p}$, we would output $\mathfrak{M}[\ell, p]$.

The basis is when $\ell' = 1$. Then, for every $p' \in \{0, 1, \ldots, \min(p, \ell \cdot q)\}$, we initialize $\mathfrak{M}[\ell', p'] = \binom{d}{p'}$.

Now, for every $\ell' \in [\ell]$ in increasing order, and every $p' \in \{0, 1, \ldots, p\}$ in arbitrary order, we perform the following computation:

$$\mathfrak{M}[\ell', p'] \leftarrow \sum_{s=0}^{\min(p',q)} \binom{d}{s} \cdot \mathfrak{M}[\ell' - 1, p' - s].$$

Clearly, the computation can be performed in polynomial time (since the input numbers are encoded in unary, and the numbers stored in the table are encoded in binary).

Combinatorially, the interpretation of $W_{\ell',p'}$ is of the number of choices to pick exactly p' elements from a universe that is partitioned into ℓ' parts of size d each, such that we can pick at most q elements from each part. Equivalently, we can consider the number of choices to pick exactly $s \leq p'$ elements from the last part of the universe, and then, for each such choice, we can consider the number of choices to pick exactly $p' - s$ additional elements from the remainder of the universe, such that we can pick at most q elements from each part. This yields the following equality:

$$W_{\ell',p'} = \sum_{s=0}^{\min(p',q)} \binom{d}{s} \cdot W_{\ell'-1,p'-s}.$$

In turn, using straightforward induction, this equality yields the correctness of the computation. ◀

Now, we define lift as follows.

▶ **Definition 34** (Procedure lift)**.** *Given an instance (G, k) of #Vertex Cover, the output of* reduce*, and the solution $x^\star$ to this output, the procedure* lift *performs the following operations:*

1. *Initialize $\widehat{x} \leftarrow x^\star$.*
2. *For $i = 0, 1, \ldots, k_2$:*
   a. *Use the algorithm in Lemma 33 to compute w_i.*
   b. *Let $y_i \leftarrow \lfloor \widehat{x}/w_i \rfloor$.*
   c. *Update $\widehat{x} \leftarrow \widehat{x} - y_i \cdot w_i$.*
   d. *Let $z_i \leftarrow y_i \cdot \sum_{j=0}^{k_2-i} \binom{n_1-n_2}{j}$.*
3. *Return $z = \sum_{i=0}^{k_2} z_i$.*

We start the analysis with the following observation, whose correctness is immediate from Lemma 33 and the definition of lift.

▶ **Observation 35.** lift *runs in polynomial time.*

For every $X \in \mathcal{S}_2$, define $\mathsf{Pull}(X) = \{X \cup U : U \subseteq V(G_1) \setminus V(G_2), |X| + |U| \leq k_1\}$. For the correctness of lift, we prove the two following lemmas.

▶ **Lemma 36.** *We have that* (i) $\mathcal{S}_1 = \bigcup_{X \in \mathcal{S}_2} \mathsf{Pull}(X)$*, and* (ii) *for distinct $X, Y \in \mathcal{S}_2$, $\mathsf{Pull}(X) \cap \mathsf{Pull}(Y) = \emptyset$.*

Proof. Recall that G_2 is obtained from G_1 be the removal of all isolated vertices, and that $k_2 = k_1$. Hence, every vertex cover of G_1 of size at most k_1 is the union of two sets, A and B, where A is a vertex cover of G_2 of size at most k_2, and $B \subseteq V(G_1) \setminus V(G_2)$ is of size at most $k_1 - |A|$. So, the first item follows, and the second item is immediate. ◀

▶ **Lemma 37.** *For every* $i \in \{0, 1, \ldots, k_2\}$*, we have that* (i) $y_i = |\mathcal{S}_2^i|$*, and* (ii) $z_i = |\bigcup_{X \in \mathcal{S}_2^i} \mathsf{Pull}(X)|$.

Proof. From Lemma 30, we have that $x^\star = \sum_{X \in \mathcal{S}_2} |\mathsf{Map}(X)| = \sum_{i=0}^{k_2} \sum_{X \in \mathcal{S}_2^i} |\mathsf{Map}(X)|$. So, by Lemma 31, we derive that $x^\star = \sum_{i=0}^{k_2} |\mathcal{S}_2^i| \cdot w_i$. Observe that for every $i \in \{0, 1, \ldots, k_2\}$, $|\mathcal{S}_2^i| \leq \binom{n_2}{i}$. Hence, due to Lemma 32, it follows that for every $i \in \{0, 1, \ldots, k_2\}$, $w_i \geq \sum_{j=i+1}^{k_2} |\mathcal{S}_2^j| \cdot w_j$. Given the manner in which lift handles the variables $\widehat{x}$ and $y_0, y_1, \ldots, y_{k_2}$, this implies the correctness of the first item of the lemma.

Now, observe that for any $X \in \mathcal{S}_2^i$, $|\mathsf{Pull}(X)| = \sum_{j=0}^{k_2-i} \binom{n_1-n_2}{j}$, and from the second item of Lemma 36, it follows that $|\bigcup_{X \in \mathcal{S}_2^i} \mathsf{Pull}(X)| = \sum_{X \in \mathcal{S}_2^i} |\mathsf{Pull}(X)|$. From these arguments, and since we have already proved the correctness of the first item, we derive the correctness of the second item as well. ◀

Having Corollary 24 and Lemmas 36 and 37 at hand, we prove the following lemma, which implies the correctness of lift.

▶ **Lemma 38.** *We have that* $|\mathcal{S}| = z$.

Proof. By Corollary 24, $|\mathcal{S}| = |\mathcal{S}_1|$. From Lemma 36, we have $|\mathcal{S}_1| = |\bigcup_{X \in \mathcal{S}_2} \mathsf{Pull}(X)|$, which equals $\sum_{i=0}^{k_2} |\bigcup_{X \in \mathcal{S}_2^i} \mathsf{Pull}(X)|$. Further, from Lemma 37, we have $\sum_{i=0}^{k_2} |\bigcup_{X \in \mathcal{S}_2^i} \mathsf{Pull}(X)| = \sum_{i=0}^{k_2} z_i = z$. So, we conclude that $|\mathcal{S}| = z$. ◀

Thus, the correctness of Theorem 1 follows from Observations 28 and 35, and Lemma 38.

6 Conclusion

In this paper, we modified the framework of kernelization to be suitable for counting problems. Within this framework, we presented both upper and lower bounds on the sizes of kernels for the counting versions of several central parameterized problems. In a similar fashion, one can also modify the more general frameworks of Turing kernelization and lossy kernelization to be suitable for counting problems. For the case of Turing kernels, one may focus on the restricted case where the number of calls should be bounded by a polynomial in k, which may be more interesting in practice. In particular, it is easy to see that this already substantially simplifies the case of #k-Vertex Cover, as then we can produce $k + 1$ many instances (rather than just one) of the problem corresponding to $k' = 0, 1, 2, \ldots, k$. Another motivation to consider Turing kernels for counting problems is that standard methods for counting problems often use tricks such as polynomial interpolation to compute or extract one hard-to-compute quantity from the answers to a collection of problem instances. We leave these directions to future research.

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.

2 Pierre Bergé, Benjamin Mouscadet, Arpad Rimmel, and Joanna Tomasik. Fixed-parameter tractability of counting small minimum (s, t)-cuts. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 79–92. Springer, 2019.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

4 Hans L Bodlaender, Bart MP Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

5 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 276–285, 2021.

6 Jonathan F Buss and Judy Goldsmith. Nondeterminism within pˆ. *SIAM Journal on Computing*, 22(3):560–572, 1993.

7 Liming Cai, Jianer Chen, Rodney G Downey, and Michael R Fellows. Advice classes of parameterized tractability. *Annals of pure and applied logic*, 84(1):119–138, 1997.

8 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms second edition*. MIT Press and McGraw-Hill, 2001.

9 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory of Computing Systems*, 60(4):737–758, 2017.

10 Radu Curticapean. Counting problems in parameterized complexity. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 1:1–1:18, 2018.

11 Radu Curticapean. A full complexity dichotomy for immanant families. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1770–1783, 2021.

12 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017.

13 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099, 2018.

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

15 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2201–2211, 2020.

16 Reinhard Diestel. Extremal graph theory. In *Graph Theory*, pages 173–207. Springer, 2017.

17 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

18 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM Journal on Computing*, 44(5):1443–1479, 2015.

19 Eduard Eiben, Danny Hermelin, and MS Ramanujan. Lossy kernels for hitting subgraphs. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, pages 67:1–67:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

20 Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM Journal on Discrete Mathematics*, 33(3):1743–1771, 2019.

21 Michael R Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *International Workshop on Parameterized and Exact Computation*, pages 276–277. Springer, 2006.

22 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018.

23 Jacob Focke and Marc Roth. Counting small induced subgraphs with hereditary properties. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1543–1551, 2022.

24 Fedor V Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal fpt algorithms. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 470–479. IEEE, 2012.

25 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

26 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. *J. Comput. Syst. Sci.*, 123:76–102, 2022.

27 Fabrizio Grandoni, Stefan Kratsch, and Andreas Wiese. Parameterized approximation schemes for independent set of rectangles and geometric knapsack. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, pages 53:1–53:16, 2019.

28 Akira Isihara. *Statistical physics*. Academic Press, 2013.

29 Bart M. P. Jansen and Bart van der Steenhoven. Kernelization for counting problems on graphs: Preserving the number of minimum solutions. In *Accepted to 18th IInternational Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2023.

30 Eun Jung Kim, Maria J. Serna, and Dimitrios M. Thilikos. Data-compression for parametrized counting problems on sparse graphs. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 20:1–20:13, 2018.

31 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM (JACM)*, 67(3):1–50, 2020.

32 R Krithika, Diptapriyo Majumdar, and Venkatesh Raman. Revisiting connected vertex cover: Fpt algorithms and lossy kernels. *Theory of Computing Systems*, 62(8):1690–1714, 2018.

33 Ramaswamy Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

34 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017.

35 Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 224–237, 2017.

36 Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Efficient computation of representative weight functions with applications to parameterized counting (extended version). In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 179–198, 2021.

37 Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, pages 15:1–15:21, 2019.

38 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

39 J Scott Provan and Michael O Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

40 Dimitrios M Thilikos. Compactors for parameterized counting problems. *Computer Science Review*, 39:100344, 2021.

41 Marc Thurley. Kernelizations for parameterized counting problems. In *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007, Proceedings*, pages 703–714, 2007.

42 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

43 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

44 René van Bevern, Till Fluschnik, and Oxana Yu Tsidulko. On approximate data reduction for the rural postman problem: Theory and experiments. *Networks*, 76(4):485–508, 2020.