On Problems Without Polynomial Kernels
(Extended Abstract)

Hans L. Bodlaender[*], Rodney G. Downey[**],
Michael R. Fellows[***], and Danny Hermelin[†]

Abstract. Kernelization is a central technique used in parameterized algorithms, and in other techniques for coping with NP-hard problems. In this paper, we introduce a new method which allows us to show that many problems do not have polynomial size kernels under reasonable complexity-theoretic assumptions. These problems include k-PATH, k-CYCLE, k-EXACT CYCLE, k-SHORT CHEAP TOUR, k-GRAPH MINOR ORDER TEST, k-CUTWIDTH, k-SEARCH NUMBER, k-PATHWIDTH, k-TREEWIDTH, k-BRANCHWIDTH, and several optimization problems parameterized by treewidth or cliquewidth.

1   Introduction

Parameterized complexity extends classical complexity theory in a way that allows a refined categorization of tractable and intractable computational problems. This is done by a two-dimensional analysis of problems instances – one dimension used as usual for measuring the input-length, and the other used for measuring other structural-properties of the input, *e.g.* its witness size. A problem is considered tractable, if there is an algorithm solving it with any super-polynomial running-time confined strictly to the parameter. As an example, consider the k-VERTEX COVER problem: Given a graph G and a parameter $k \in \mathbb{N}$, determine whether G has a vertex cover of size k. When viewed classically, this problem is NP-complete. However, its parameterized variant can be solved in $O(2^k n)$ time [11] (see [22] for improvements), which is practical for instances with small parameter values, and in general is far better than the $O(n^{k+1})$ running time of the brute-force algorithm. More generally, a problem is said to be *fixed-parameter tractable* if it has an algorithm running in time $f(k)p(n)$ (FPT-time), where f is any computable function solely in the parameter k, and $p(n)$

is a polynomial in the total input length n [11]. The class of all fixed-parameter tractable problems is denoted by FPT. The first class of *fixed-parameter intractable* problems is W[1], and it is known that if FPT = W[1] then n variable 3-SAT can be solved in $2^{o(n)}$ time [10].

A fundamental and very powerful technique in designing FPT algorithms is *kernelization*. In a nutshell, a kernelization algorithm for a parameterized problem is a *polynomial-time transformation* that transforms any given instance to an equivalent instance of the same problem, with size and parameter bounded by a function of the parameter in the input. Typically this is done using so-called *reduction rules*, which allow the safe reduction of the instance to an equivalent "smaller" instance. In this sense, kernelization can be viewed as polynomial-time preprocessing which has universal applicability, not only in the design of efficient FPT algorithms, but also in the design of approximation and heuristic algorithms [21].

It is clear that any (decidable) language which has a kernelization algorithm is in FPT. Somewhat more surprising, but still very simple to show, is that all problems in FPT have kernelization algorithms [9]. This is seen by considering the two cases $f(k) \geq n$ and $f(k) < n$ separately, where $f(k)$ is the parameter-dependent time-bound of the algorithm solving the given problem. Since every FPT problem has a kernelization algorithm, it is interesting to study problems that are kernelizable in a stricter sense - for example, problems which allow kernelization algorithms that reduce instances to a size which is *polynomially* bounded by the parameter. Such problems are said to have a polynomial kernelization algorithm, or a *polynomial kernel*. For instance, the kernelization algorithm of Buss discussed above is a polynomial kernel, and so k-Vertex Cover has a polynomial kernel. Other problems known to have polynomial kernels include k-Leaf Spanning Tree [6], k-Feedback Vertex Set [5, 8], k-Planar Dominating Set [1], k-Cluster Editing [20], k-Hitting Set for Sets of Bounded Size [27], and many more.

On the other hand, there are also several problems for which no polynomial kernel has yet been found. These clearly include all problems known to be W[1]-hard, as the existence of a kernel for such a problem would imply W[1] = FPT. So we focus on parameterized problems known to be in FPT. A great number of such problems, are problems shown to be in FPT using heavy machinery such as *color-coding* [2], the *graph minor technique* [13], or *tree-decomposition dynamic programming* [3]. In many cases, the algorithms given by these frameworks are impractical in practice. For instance, consider the k-Path problem: Given a graph G and a parameter $k \in \mathbb{N}$, determine whether G has a simple path of length k.

This problem can be solved in $O(2^{O(k)}n^2 \lg n)$ time using the color-coding technique of Alon, Yuster, and Zwick [2]. This time complexity might seem similar to the complexity of the algorithm for k-Vertex Cover mentioned above, however the hidden constant in the $O(k)$ exponent is quite large, ruling-

out any possibility for practical usefulness[1]. Nevertheless, an efficient polynomial kernel could be a promising path in making this algorithm practical. Does k-PATH have a polynomial kernel? k-MINOR ORDER TEST and k-TREEWIDTH are other good examples, as both serve as highly time-consuming subroutines in most algorithms deploying the graph minor technique or tree-decomposition dynamic programming. Do k-MINOR ORDER TEST and k-TREEWIDTH have polynomial kernels?

In this paper, we introduce a new method which allows us to show that many problems do not have polynomial kernels under reasonable complexity-theoretic assumptions. We believe that this material is significant and will have wide applications. For instance, learning of our material, three other teams of authors, namely Fortnow and Santhanam, Chen *et al.*, and Buhrman have applied the concepts in this paper to other arenas.

Questions such as these are the motivating starting point of this paper. Clearly, if P = NP then all parameterized problems based on NP-complete problems have constant size kernels. Thus, any method we generate to show that a problem is unlikely to have a polynomial kernel will entail a complexity-theoretic hypothesis. For developing such a hypotheses, we introduce the notion of a *distillation algorithm*. Intuitively speaking, a distillation algorithm for a given problem functions like a Boolean *OR* gate of problem-instances – it receives as input a sequence of instances, and outputs yes-instance iff at least one of the instances in the sequence is also a yes-instance. The algorithm is allowed to run in time polynomial in the total length of the sequence, but must output an instance whose size is polynomially bounded by the size of the maximum-size instance in its input sequence. We remark that independently and somewhat earlier, a similar notion had been formulated by Harnik and Naor [23] as an example in relation to compression-related cryptographic problems. Or paper, as well as the subsequent papers mentioned above, show that the notion of distillation is of central importance in complexity considerations.

We study the possibility of the existence of distillation algorithms for NP-complete problems, and conjecture that this is highly implausible. It is clear that if any NP-complete problem has a distillation algorithm, then they all do. This seems very unlikely. Intuitively, large amounts of information cannot be coalesced into a single small instance. This notion seems rather similar to the notion of P-selectivity which collapses the polynomial hierarchy [25]. It turns out that this intuition can be realized to also relate the existence of distillation algorithms for NP-complete problems to a similar collapse. After correspondence about this issue, Fortnow and Santhanam verified a conjecture of ours proving that the existence a of distillation algorithm for any NP-complete problem would imply the collapse of the polynomial hierarchy to the third level [18]. This allows us to

---

[1] It was brought to our attention that there are recent improvements to the k-PATH algorithm mentioned above which have rather practical running-times [24, 26]. Both these algorithms employ variants of the color-coding technique, however, these variants are not known to be as general as the original technique. In any case, the reader should mostly consider the k-PATH problem as a case-study example.

prove, via a carefully defined parametric-analog of distillation, the unlikelihood of polynomial kernels for FPT problems such as k-Path, k-Minor Order Test and others. In particular, our study gives rise to the following theorem.

Theorem 1. *Unless all* NP-*complete problems have distillation algorithms, none of the following* FPT *problems have polynomial kernels:* k-Path, k-Cycle, k-Exact Cycle, k-Short Cheap Tour, k-Graph Minor Order Test, k-Bounded Treewidth Subgraph Test, k-Planar Graph Subgraph Test, k-Planar Graph Induced Subgraph Test, w-Independent Set, w-Clique, *and* w-Dominating Set.

Here, w-Independent Set, w-Clique, and w-Dominating Set denote the classical Independent Set, Clique, and Dominating Set problems parameterized by the treewidth of their given graphs. These are given as mere examples. Many other graph-theoretic problems parameterized by the treewidth of the graph could have been used in the theorem.

We next turn to study distillation of coNP-complete problems. Although we are unable to relate the existence of distillation algorithms for coNP-complete problems to any known complexity conjecture, we can still show that polynomial kernels for some important FPT problems not captured by Theorem 1, imply distillation algorithms for coNP-complete problems.

Theorem 2. *Unless all* coNP-*complete problems have distillation algorithms, none of the following* FPT *problems have polynomial kernels:* k-Cutwidth, k-Modified Cutwidth, k-Search Number, k-Pathwidth, k-Treewidth, k-Branchwidth, k-Gate Matrix Layout, k-Front Size, w-3-Coloring *and* w-3-Domatic Number.

We remark that in unpublished work, Buhrman [7] has shown that there are oracles relative to which no coNP-complete problem has a distillation algorithm. We believe that the same information-theoretical intuition applies here, and that no coNP-complete problem can have a distillation algorithm.

In the full version of this paper, we also study *sub-exponential kernels*, *i.e* kernelization algorithms that reduce instances to a size which is *sub-exponentially* bounded by the parameter. In particular, we prove that there are problems solvable in $O(2^k n)$ time which (unconditionally) do not have any sub-exponential kernel of size $2^{o(k)}$. This relates our material to the work of Flum, Grohe, and Weyer [17] who introduced the notion of "bounded fixed-parameter tractability" ass an attempt to provide a theory for feasible FPT algorithms. They argued that for an FPT algorithm to be useful in practice, it should most likely have a running time of $2^{O(k)} n^{O(1)}$ or perhaps $2^{k^{O(1)}} n^{O(1)}$. We show that the notion of small kernel and small running-time are quite different.

2   Preliminaries

Throughout the paper, we let Σ denote a finite alphabet, and $\mathbb{N}$ the set of natural numbers. A (classical) problem L is a subset of Σ^*, where Σ^* is the set of all

finite length strings over Σ. In natural cases, the strings in L will be an encoding of some combinatorial object, *e.g.* graphs. We will call strings $x \in \Sigma^*$ which are proper encodings, *input* of L, regardless of whether $x \in L$. We will often not distinguish between a combinatorial object and its string encoding, using for example G to denote both a graph and a string in Σ^*.

A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$. In this way, an input (x, k) to a parameterized language consists of two parts, where the second part k is the *parameter*. A parameterized problem L is *fixed-parameter tractable* if there exists an algorithm which on a given $(x, k) \in \Sigma^* \times \mathbb{N}$, decides whether $(x, k) \in L$ in $f(k)p(n)$ time, where f is an arbitrary computable function solely in k, and p is a polynomial in the total input length (including the unary encoding of the parameter) $n = |x| + k$. Such an algorithm is said to run in FPT-*time*, and FPT is the class of all parameterized problems that can be solved by an FPT-time algorithm (*i.e.* all problems which are fixed-parameter tractable). For more background on parameterized complexity, the reader is referred to [5, 11, 16].

To relate notions from parameterized complexity and notions from classic complexity theory with each other, we use a natural way of mapping parameterized problems to classical problems. The mapping of parameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \notin \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the *unparameterized version* of a parameterized problem L is the langauge $\widetilde{L} = \{x\#1^k \,|\, (x, k) \in L\}$. We next give a formal definition for the central notion of this paper:

Definition 1 (Kernelization). *A* kernelization algorithm, *or in short, a* kernel *for a parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *is an algorithm that given* $(x, k) \in \Sigma^* \times \mathbb{N}$, *outputs in* $p(|x| + k)$ *time a pair* $(x', k') \in \Sigma^* \times \mathbb{N}$ *such that*

– $(x, k) \in L \Leftrightarrow (x', k') \in L$,
– $|x'|, k' \leq f(k)$,

where f is an arbitrary computable function, and p a polynomial. Any function f as above is referred to as the size *of the kernel.*

That is, if we have a kernel for L, then for any $(x, k) \in \Sigma \times \mathbb{N}$, we can obtain in polynomial time an equivalent instance with respect to L whose size is bounded by a function of the parameter. If the size of the kernel is polynomial, we say that the parameterized langauge L has a *polynomial kernel*.

There is also a more general definition for kernelization than the one given above which sometimes appears in practice. This definition allows a kernelization algorithm for a parameterized problem L to map an instance of L to an instance of another problem L'. We remark that all results in this paper easily follow for most cases of the more general definition. Nevertheless, we will present these results with Definition 1 for the sake of clarity and simplicity.

3 A Generic Lower-Bounds Engine

In the following we develop the main engine for proving Theorems 1 and 2. This engine evolves around the notion of *distillation algorithms* for NP-complete

problems. We first introduce this notion, and then carefully define a parametric-analog of a distillation algorithm which we call a composition algorithm. Following this, we show that if a compositional parameterized problem has a polynomial kernel, then its unparameterized counterpart has a distillation algorithm.

Definition 2 (Distillation). *A* distillation algorithm *for a classical problem* $L \subseteq \Sigma^*$ *is an algorithm that receives as input a sequence* $(x_1, \ldots, x_t)$, *with* $x_i \in \Sigma^*$ *for each* $1 \leq i \leq t$, *uses time polynomial in* $\sum_{i=1}^{t} |x_i|$, *and outputs a string* $y \in \Sigma^*$ *with*

1. *$y \in L \iff x_i \in L$ for some $1 \leq i \leq t$.*
2. *$|y|$ is polynomial in $\max_{1 \leq i \leq t} |x_i|$.*

That is, given a sequence of t instances of L, a distillation algorithm gives an output that is equivalent to the sequence of instances, in the sense that a collection with at least one yes-instance (*i.e.* instance belonging to L) is mapped to a yes-instance, and a collection with only no-instances is mapped to a no-instance. (In a certain sense, this functions like a Boolean *OR* operator.) The algorithm is allowed to use polynomial-time in the total size of all instances. The crux is that its output must be bounded by a polynomial in the size of the largest of the instances from the sequence, rather than in the total length of the instances in the sequence. We next introduce the notion of a composition algorithm for parameterized problems. In some sense, one can view a composition algorithm as the parametric-analog of a distillation algorithm.

Definition 3 (Composition). *A* composition algorithm *for a parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *is an algorithm that receives as input a sequence* $((x_1, k), \ldots, (x_t, k))$, *with* $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ *for each* $1 \leq i \leq t$, *uses time polynomial in* $\sum_{i=1}^{t} |x_i| + k$, *and outputs* $(y, k') \in \Sigma^* \times \mathbb{N}^+$ *with*

1. *$(y, k') \in L \iff (x_i, k) \in L$ for some $1 \leq i \leq t$.*
2. *k' is polynomial in k.*

Hence, given a sequence of instances for L, a composition-algorithm outputs an equivalent instance to this sequence in same sense of a distillation algorithm, except that now the parameter of the instance is required to be polynomially-bounded by parameter appearing in all instances of the sequence, rather than the size of the instance bounded by the maximum size of of all instances.

We call classical problems with distillation algorithms *distillable problems*, and parameterized problems with composition algorithms *compositional problems*. Despite the similarities between the two definitions, as we shall soon see, the existence of composition algorithms for some parameterized problems is much more plausible than the existence of distillations for their unparameterized counterparts. Nevertheless, there is still a deep connection between distillation and composition, obtained via polynomial kernelization. In particular, in the following lemma we prove that combining a composition algorithm for a parameterized problem L, with a polynomial kernel for it, admits a distillation algorithm for the unparameterized counterpart of L.

Lemma 1. *Let L be a compositional parameterized problem whose unparameterized version $\widetilde{L}$ is NP-complete. If L has a polynomial kernel, then $\widetilde{L}$ is also distillable.*

Proof. Let $\widetilde{x}_1, \ldots, \widetilde{x}_t \in \Sigma^*$ be instances of $\widetilde{L}$, and let $(x_i, k_i) \in \Sigma^* \times \mathbb{N}^+$ denote the instance of L derived from $\widetilde{x}_i$, for all $1 \leq i \leq t$. Since $\widetilde{L}$ is NP-complete, there exist two polynomial-time transformations $\Phi : \widetilde{L} \to \text{SAT}$ and $\Psi : \text{SAT} \to \widetilde{L}$, where SAT is the problem of deciding whether a given boolean formula is satisfiable. We use the composition and polynomial kernelization algorithms of L, along with Φ and Ψ, to obtain a distillation algorithm for $\widetilde{L}$. The distillation algorithm proceeds in three steps.

Set $k = \max_{1 \leq i \leq t} k_i$. In the first step, we take the subsequence in $((x_1, k_1), \ldots, (x_t, k_t))$ of instances whose parameter equals ℓ, for each $1 \leq \ell \leq k$. We apply the composition algorithm on each one of these subsequence separately, and obtain a new sequence $((y_1, k_1'), \ldots, (y_r, k_r'))$, where (y_i, k_i'), $1 \leq i \leq r$, is the instance obtained by composing all instances with parameters equaling the i'th parameter value in $\{k_1, \ldots, k_t\}$. In the second step, we apply the polynomial kernel on each instance of the sequence $((y_1, k_1'), \ldots, (y_r, k_r'))$, to obtain a new sequence $((z_1, k_1''), \ldots, (z_r, k_r''))$, with (z_i, k_i'') the instance obtained from (y_i, k_i'), for each $1 \leq i \leq r$. Finally, in the last step, we transform each $\widetilde{z}_i$, the unparameterized instance of $\widetilde{L}$ derived from (z_i, k_i''), to a Boolean formula $\Phi(\widetilde{z}_i)$. We output the instance of $\widetilde{L}$ for which Ψ maps the disjunction of these formulas to, i.e. $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i))$.

We argue that this algorithm distills the sequence $(\widetilde{x}_1, \ldots, \widetilde{x}_t)$ in polynomial time, and therefore is a distillation algorithm for $\widetilde{L}$. First, by the correctness of the composition and kernelization algorithms of L, and by the correctness of Φ and Ψ, it is not difficult to verify that $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i)) \in \widetilde{L} \iff \widetilde{x}_i \in \widetilde{L}$ for some i, $1 \leq i \leq t$. Furthermore, the total running-time of our algorithm is polynomial in $\sum_{i=1}^{t} |\widetilde{x}_i|$. To complete the proof, we show that the final output returned by our algorithm is polynomially bounded in $n = \max_{1 \leq i \leq t} |\widetilde{x}_i|$. The first observation is that since each $\widetilde{x}_i$ is derived from the instance (x_i, k_i), $1 \leq i \leq t$, we have $r \leq k = \max_{1 \leq i \leq t} k_i \leq \max_{1 \leq i \leq t} |\widetilde{x}_i| = n$. Therefore, there are at most n instances in the sequence $((y_1, k_1'), \ldots, (y_r, k_r'))$ obtained in the first step of the algorithm. Furthermore, as each (y_i, k_i'), $1 \leq i \leq r$, is obtained via composition, we know that k_i' is bounded by some polynomial in $\ell \leq k \leq n$. Hence, since for each $1 \leq i \leq r$, the instance (z_i, k_i'') is the output of a polynomial kernelization on (y_i, k_i'), we also know that (z_i, k_i'') and $\widetilde{z}_i$ have size polynomially-bounded in n. It follows that $\sum_{i=1}^{r} |\widetilde{z}_i|$ is polynomial in n, and since both Φ and Ψ are polynomial-time, so is $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i))$. $\square$

We conclude this section by stating a lemma proven by by Fortnow and Santhanam [18], which verifies our initial intuition that NP-complete problems are unlikely to have distillation algorithms. It is clear from Definition 2, that if any NP-complete problem were distillable, then they all would be – we can use the polynomial-time reductions provided for any NP-complete problem $\widetilde{L}$ to and from our presumed distillable NP-complete problem to distill $\widetilde{L}$. Fortnow and

Santhanam proved that a distillation algorithm for any NP-complete problem would imply coNP $\subseteq$ NP/poly, which on its turn implies that the polynomial hierarchy collapses to at most three levels [30], a hierarchy generally believed to be proper.

Lemma 2 ([18]). *If any* NP*-complete problem has a distillation algorithm then* coNP $\subseteq$ NP/poly*.*

4 Applications

Lemmas 1 and 2 which together form our lower bound engine together imply that any compositional parameterized problem whose unparameterized counterpart is NP-complete cannot have a polynomial kernel, unless the polynomial hierarchy collapses. In the following we exemplify the strength of our lower bound engine by giving several examples of compositional FPT problems that are based on unparameterized classical NP-complete problems. We focus only on natural examples, and in particular, we complete the proof of Theorem 1.

Let us call a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ a *parameterized graph problem*, if for any $(x, k) \in L$, x is an encoding of a graph.

Lemma 3. *Let L be a parameterized graph problem such that for any pair of graphs G_1 and G_2, and any integer $k \in \mathbb{N}$, we have $(G_1, k) \in L \lor (G_2, k) \in L \iff (G_1 \cup G_2, k) \in L$, where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then L is compositional.*

As an immediate corollary of the simple lemma above, we get that our case-study problem k-Path is compositional, and thus is unlikely to have a polynomial kernel. Indeed, the disjoint union of two graphs has a k-path iff one of the graphs has a k-path. Two other similar examples are the k-Cycle and k-Exact Cycle problems, which respectively ask to determine whether a given graph has a (not necessarily induced) subgraph which is isomorphic to a cycle with at least k vertices and a cycle with exactly k vertices. Both these problems are also in FPT by the color-coding technique of Alon *et al.* [2], and are compositional by the lemma above. Another example is k-Short Cheap Tour, which given an edge-weighted graph, asks whether there is a tour of length at least k in the graph with total weight not more than some given threshold. This problem is in FPT due to [28], and is again compositional according to Lemma 3.

In fact, Lemma 3 implies that any parameterized problem which asks to determine whether a specific graph H (*e.g.* a k-clique) is a "subgraph of some kind" of an input graph G, for almost any natural notion of subgraph, is compositional when parameterized by H (or more precisely, by the *numeric encoding of H*, the position of H in some canonical ordering of simple graphs). For example, consider the k-Minor Order Test problem, famously in FPT due to Robertson and Seymour's celebrated Graph Minor Theorem. This problem asks to decide whether a given graph H is a minor of another given graph G, and the parameter k is H. Clearly, if we slightly relax the problem and require H to

be connected, the disjoint union construction of Lemma 3 above gives a composition algorithm for this problem. If H is not connected, we can connect it by adding a new *global* vertex adjacent to all other vertices in H, and then add such a global vertex to each G_i, $1 \leq i \leq t$. By similar arguments we can also show that k-BOUNDED TREEWIDTH SUBGRAPH TEST – the problem of determining whether a given bounded treewidth graph occurs as a subgraph in another given graph (in FPT again via color-coding [2]) – is also compositional. Other two good examples are k-PLANAR GRAPH SUBGRAPH TEST and k-PLANAR GRAPH INDUCED SUBGRAPH TEST, both in FPT due to [12].

We now turn to proving the last item of Theorem 1. In particular, we show that many natural NP-complete problems parameterized by treewidth are unlikely to have a polynomial kernel. We illustrate the technique with one example, and then state the general result that can be obtained using the same way. Consider the w-INDEPENDENT SET problem: Given a graph G, a tree-decomposition $\mathcal{T}$ of G of width $w \in \mathbb{N}^+$, and an integer $k \in \mathbb{N}^+$, determine whether G has an independent set of size k. Note that the parameter here is w and not k. We call the unparameterized variant of w-INDEPENDENT SET the INDEPENDENT SET WITH TREEWIDTH problem. Clearly, INDEPENDENT SET WITH TREEWIDTH is NP-complete by the straightforward reduction from INDEPENDENT SET which appends a trivial tree-decomposition to the given instance of INDEPENDENT SET.

To show that w-INDEPENDENT SET is compositional, we will work with a 'guarantee' version, the w-INDEPENDENT SET REFINEMENT problem: given a graph G, a tree-decomposition $\mathcal{T}$ of G, and an independent set I in G, determine whether G has an independent set of size $|I| + 1$. The parameter is the width w of $\mathcal{T}$. The unparameterized variant of w-INDEPENDENT SET REFINEMENT is INDEPENDENT SET REFINEMENT WITH TREEWIDTH. It is easy to see that this problem is NP-complete by the following reduction from INDEPENDENT SET WITH TREEWIDTH – Given an instance $(G, \mathcal{T}, k)$, construct the instance $(G', \mathcal{T}', I)$, where G' is the graph obtained by adding $k - 1$ new pairwise non-adjacent vertices I to G which are connected to all the old vertices, and $\mathcal{T}'$ is the tree-decomposition obtained by adding I to each node in $\mathcal{T}$.

Lemma 4. w-INDEPENDENT SET REFINEMENT *is compositional, and furthermore, if* w-INDEPENDENT SET *has a polynomial kernel then so does* w-INDEPENDENT SET REFINEMENT.

The proof of the lemma above (which we omit due to space constraints) implies that to fit a natural NP-complete graph problem parameterized by treewidth into the context of our lower-bound framework, one has to basically show two things: First, that the refinement variant of the problem is compositional, and second, that the unparameterized version of the refinement variant is NP-complete. In fact, this technique is not necessarily limited to treewidth, but can be used with almost any other structural parameter such as cliquewidth, max. degree, min. vertex-cover, and so forth. To complete the proof of Theorem 1, what is left to prove is that DOMINATING SET REFINEMENT WITH TREEWIDTH is NP-complete; CLIQUE REFINEMENT WITH TREEWIDTH can be

seen to be NP-complete by a similar construction shown above. We omit the
details.

5   Extensions

We next extend the framework presented in the previous section so that it captures other important FPT problems not captured by Theorem 1. In particular, we discuss the proof for Theorem 2. The main observation we use for the former is that an AND-variant of a composition algorithm for a parameterized problem L, yields a composition algorithm for $\overline{L}$, the complement of L. This observation is useful since a lot of problems have natural AND-compositions rather than regular compositions. As any FPT problem has a polynomial kernel iff its complement also has one, showing that a coFPT problem is compositional is just as good for our purposes as showing that its complement in FPT is compositional.

Lemma 5. *Let L be a parameterized graph problem such that for any pair of graphs G_1 and G_2, and any integer $k \in \mathbb{N}$, we have $(G_1, k) \in L \wedge (G_2, k) \in L \iff (G_1 \cup G_2, k) \in L$, where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then $\overline{L}$, the complement of L, is compositional.*

There are many FPT problem with a natural composition as above. These include the classical "width problems" k-PATHWIDTH, k-TREEWIDTH, and k-BRANCHWIDTH (see [4] for formal definitions and FPT algorithms for these problems). Three closely related relatives of these problems are k-SEARCH NUMBER [14], k-FRONT SIZE [4], and k-GATE MATRIX LAYOUT [15], which all have AND-composition by the lemma above. Lemma 5 also implies that two other famous FPT "width problems" are AND-compositional, namely, k-CUTWIDTH and k-MODIFIED CUTWIDTH [14].

We prove the last item of Theorem 2 by using refinement variants as done for the treewidth parameterized problems in Theorem 1. In this context, it is worth mentioning that partitioning problems seem more adaptable to AND-compositions, as opposed to subset problems which are better suited for regular composition. Recall that w-3-CHROMATIC NUMBER is the problem of determining, given a graph G and a tree-decomposition $\mathcal{T}$ of G, whether there exists a partitioning (or *coloring*) Π of $V(G)$ into three classes, where each class induces an independent set in G. The parameter is the width of $\mathcal{T}$. The w-3-DOMATIC NUMBER problem is defined similarly, except that here the goal is to partition (or *domatic-color*) $V(G)$, again into three classes, with each class inducing a dominating set of G. Indeed, we selected w-3-CHROMATIC NUMBER and w-3-DOMATIC NUMBER for Theorem 2 as they are two of the more well-known graph partitioning problems. Many other natural partitioning problems could have been selected as well.

The refinement variants of these two problems, w-3-CHROMATIC NUMBER REFINEMENT and w-3-DOMATIC NUMBER REFINEMENT, are defined by adding to the input an appropriate vertex-partitioning Π (with respect to the problem definition), of cardinality four for w-3-CHROMATIC NUMBER REFINEMENT and

two for w-3-DOMATIC NUMBER REFINEMENT. It is easy to see that the unparameterized versions of these two problems are NP-complete by recalling that one can color planar graphs with four colors in polynomial-time (see *e.g.* [29]), while it is NP-complete to decide whether a planar graph is 3-colorable, and by recalling that every graph without an isolated vertex can be domatic-colored with two colors in polynomial-time (see *e.g.* [19]). Furthermore, it is easy to see that the standard disjoint union algorithm is an AND-composition for these two problems. Thus, by similar arguments used in Section 4, we can conclude that a polynomial-kernel for either w-3-CHROMATIC NUMBER or w-3-DOMATIC NUMBER implies that all coNP-complete problems are distillable.

Acknowledgements

References

1. J. Alber, M.R. Fellows, and R. Niedermeier. Efficient data reduction for DOMINATING SET: A linear problem kernel for the planar case. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 150–159, 2002.
2. N. Alon, R. Yuster, and U. Zwick. Color coding. *Journal of the ACM*, 42(4):844–856, 1995.
3. S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey. *BIT Numerical Mathematics*, 25(1):2–23, 1985.
4. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
5. Hans L. Bodlaender. A cubic kernel for feedback vertex set. In *Proceedings of the 24th annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 320–331, 2007.
6. P.S. Bonsma, T. Brüggemann, and G.J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In *Proceedings of the 28th international symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 259–268, 2003.
7. H. Buhrman. Private communication, 2007.
8. K. Burrage, V. Estivill-Castro, M.R. Fellows, M.A. Langston, S. Mac, and F.A. Rosamond. The undirected feedback vertex set problem has a Poly(k) kernel. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 192–202, 2006.
9. L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. Advice classes of paramterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997.
10. L. Cai and D.W. Juedes. Subexponential parameterized algorithms collapse the W-hierarchy. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 273–284, 2001.

11. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
12. D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the 6th annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 632–640, 1995.
13. M.R. Fellows and M.A. Langston. Nonconstructive proofs of polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1988.
14. M.R. Fellows and M.A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal of Discrete Math*, 5(1):117–126, 1992.
15. H. Fernau. *Parameterized algorithms: A graph-theoretic approach*. PhD thesis, University of Tübingen, 2005.
16. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
17. J. Flum, M. Grohe, and M. Weyer. Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 555–567, 2004.
18. L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th Symposium on the Theory of Computing (STOC),* to appear, 2008.
19. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
20. J. Gramm, J. Guo, F. Huffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Mathematical Systems Theory*, 38:373–392, 2005.
21. J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.
22. J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of generalized vertex cover problems. In *Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS)*, pages 36–48, 2005.
23. D. Harnik and M. Naor. On the compressibility of NP instances and cryptographic applications. In *Proceedings of 47th annual IEEE symposium on Foundations Of Computer Science (FOCS)*, pages 719–728, 2006.
24. J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *Proceedings of the international Workshop on Graph-theoretic concepts in computer science (WG)*, pages 58–67, 2006.
25. K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, 1983.
26. Y. Liu, S. Lu, J. Chen, and S. Sze. Greedy localization and color-coding: Improved matching and packing algorithms. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 84–95, 2006.
27. R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1:89–102, 2003.
28. J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In *Proceedings of the 16th international Workshop on Graph-theoretic concepts in computer science (WG)*, pages 18–29, 1990.
29. N. Robertson, D.P. Sanders, P. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the 28th annual ACM Symposium on the Theory Of Computing (STOC)*, pages 571–575, 1996.
30. L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.