On Problems without Polynomial Kernels

Hans L. Bodlaender

Rodney G. Downey

Michael R. Fellows

Danny Hermelin

Technical Report UU-CS-2007-046
November 2007

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

On Problems Without Polynomial Kernels

Hans L. Bodlaender[1⋆], Rodney G. Downey[⋆⋆2],
Michael R. Fellows[⋆⋆⋆3], and Danny Hermelin[†4]

[1] Department of Information and Computing Sciences,
Utrecht University, 3508 TB Utrecht 80.089 - Netherlands
hansb@cs.uu.nl
[2] School of Mathematics, Statistics and Computer Science,
Victoria University of Wellington, Wellington 600 - New Zealand
rod.downey@vuw.ac.nz
[3] The University of Newcastle, Callaghan NSW 2308 - Australia
michael.fellows@newcastle.edu.au
[4] Department of Computer Science,
The University of Haifa, Haifa 31905 - Israel
danny@cri.haifa.ac.il

Abstract. Kernelization is a strong and widely-applied technique in the design of fixed-parameter algorithms. In a nutshell, a kernelization algorithm is a polynomial-time transformation that transforms any given parameterized instance to an equivalent instance of the same problem, with size and parameter bounded by a function of the parameter in the input. A kernelization algorithm is called a polynomial kernel if the size and parameter of the output are polynomially-bounded by the parameter of the input.

In this paper, we give evidence that a wide range of FPT problems do not have polynomial kernels. Our evidence relies on hypothesis made in the classical world (*i.e.* P vs. NP), and evolves around a new type of algorithm for classical decision problems, called a distillation algorithm, which might be of independent interest. Using the notion of distillation algorithms, we develop a generic lower-bound engine which allows us to show that a variety of FPT problems, fulfilling certain criteria, cannot have polynomial kernels unless the polynomial hierarchy collapses. These include important FPT problems such as k-Path, k-Minor Order Test, k-Planar Graph Subgraph Test, and several others. We also show that a polynomial kernel for parameterized problems fulfilling a slightly different criteria implies a distillation algorithm for all coNP-complete problems. Example of such problems are k-Pathwidth, k-Treewidth, and k-Cutwidth. Moreover, our framework also allows us to establish kernelization lower bounds for optimization problems (*e.g.* Minimum Dominating Set) parameterized by treewidth, cliquewidth, max. degree, and so forth. In the last part of the paper, we study sub-exponential kernelization algorithms, and look into these types of kernelizations from a more structural point of view.

Our results are the first negative results concerning polynomial kernels for natural FPT problems. Apart from these, the only previously known techniques for showing lower bounds were limited to linear lower bounds. We believe our results provide a glimpse of what makes polynomial kernelization intrinsically hard in certain type of problems, and also offer an accessible platform for showing similar results.

⋆ This author was partially supported by project BRICKS (Basic Research for Creating the Knowledge Society).
⋆⋆ Research supported by the Marsden Fund of New Zealand.
⋆⋆⋆ Research supported by the Australian Research Council Center of Excellence in Bioinformatics.
† Supported by the Adams Fellowship of the Israel Academy of Sciences and Humanities.

1 Introduction

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 small 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 *et al.* [27], Chen *et al.* [15], and Buhrman [9] have applied the concepts in this paper to other arenas.

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:

k-Vertex Cover:

Instance: A graph G, and $k \in \mathbb{N}$ in unary.

Question: Does G have a vertex cover of size k?

Parameter: k.

When viewed classically, this problem is NP-complete. However, its parameterized variant can be solved in $O(2^k n)$ time [18] (see [33] 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 [18]. The class of all 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 [12].

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 to safely reduce 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 [31]. For instance, Weihe showed in the late 90's that a simple reduction rule can be applied to efficiently solve the problem of covering all European trains with train stations that facilitate maintenances and repairs [40]. Nemhauser and Trotter's classical kernelization algorithm for k-Vertex Cover [35] is widely used as a preprocessing step in many approximation algorithms for this problem (see *e.g.* [4, 34]).

As an example of kernelization, consider the following algorithm for k-Vertex Cover suggested by Sam Buss which is now folklore: Given an instance (G, k) for k-Vertex Cover,

if there exists a vertex v in G with at least $k+1$ neighbors, remove v from G (along with all of its incident edges) and decrease k by one. This reduction rule is safe since any k-vertex cover of G (if one exists) must include v, as otherwise all edges incident to v cannot be covered with at most k vertices. Continuing to apply this rule until no longer possible, the graph G' in the remaining instance (G', k') has maximum degree $k' \leq k$. Since k' vertices of maximum degree k' can cover at most k'^2 edges, we know that either G' does not have a vertex cover of size k', or G' has size (excluding isolated vertices) $O(k'^2) = O(k^2)$. This algorithm exemplifies the power of kernelization in that sometimes very simple and easily implementable reduction rules allow for a dramatic reduction in the total input size. Nevertheless, the reader should not be mislead to thinking that all kernelization algorithms are that simple. Indeed, an increasing amount of research over the years has lead to the development of some rather sophisticated and involved kernelization techniques, see *e.g.* [14, 20, 26, 30, 32].

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 [11]. 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 [8], k-Feedback Vertex Set [5, 10], k-Planar Dominating Set [1], k-Cluster Editing [29], k-Hitting Set for Sets of Bounded Size [36], 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* [21], or *tree-decomposition dynamic programming* [3]. Usually, the algorithms given by these frameworks are impractical in practice. For instance, consider the k-Path problem:

k-Path:

Instance: A graph G, and $k \in \mathbb{N}$ in unary.

Question: Does G have a simple path of length k?

Parameter: 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. 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?

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 sequences 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 study the possibility of the existence of distillation algorithms for NP-complete problems, and conjecture that this is highly implausible. As evidence for this, we use an argument of Fortnow and Santhanam to show that a distillation algorithm for any NP-complete problem would imply the collapse of the polynomial hierarchy to the third level.. This allows us to 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* PH $= \Sigma_{\mathrm{p}}^{3}$*, none of the following* FPT *problems have polynomial kernels:*

- k-Path*,* k-Cycle*,* k-Exact Cycle *and* k-Short Cheap Tour*.*
- k-Graph Minor Order Test *and* k-Bounded Treewidth Subgraph Test*.*
- k-Planar Graph Subgraph Test *and* k-Planar Graph Induced Subgraph Test*.*
- k, σ-Short NonDeterministic Turing Machine Computation*.*
- 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*, and* k-Search Number*.*
- k-Pathwidth*,* k-Treewidth*, and* k-Branchwidth*.*
- k-Gate Matrix Layout *and* k-Front Size*.*
- w-3-Coloring *and* w-3-Domatic Number*.*

Both theorems above are unique in the sense that there are no previous results showing the unlikelihood of polynomial kernelization for any FPT problem. The only other negative results concerning kernelization are linear lower bounds that are either obtained through inapproximability results or through the dual-parameter approach. For instance, k-Vertex Cover cannot have a kernel of size $1.36 \cdot k$ unless P = NP [31], due to the lower bounds on the approximation factor guarantee of any approximation for the optimization variant of this problem [17]. k-Planar Vertex Cover cannot have a kernel of size $(4/3 - \varepsilon) \cdot k$ for any $\varepsilon > 0$ (unless P = NP), due to the dual-parameter approach [14]. We note that these methods have rather limited applicability, and in any case cannot deal with polynomial lower bounds. Indeed, finding methods for showing polynomial lower bounds for kernel sizes has been stated as a major open problem in Guo and Niedermeier's recent SIGACT kernelization survey [31].

In the last part of the paper, we 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 $2^{o(k)}$. Furthermore, we show that there are problems whose unparameterized classical versions are outside of P, but still they admit *arbitrary small sub-exponential kernels*. That is, kernelization algorithms with output size bounded by $(1 + \varepsilon)^k$, for any $\varepsilon > 0$.

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, 18, 24].

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 $\widetilde{L} = \{x\#1^k \mid (x, k) \in L\}$.

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*.

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 use an argument of Fortnow and Santhanam [27] to show that a distillation algorithm for any NP-complete problem implies the collapse of the polynomial hierarchy to at least three levels. We 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. We begin with the central notion of our framework:

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.

It seems highly implausible that NP-complete problems have distillation algorithms. The evidence for this in the following lemma, which shows that a distillation algorithm for an NP-complete problem implies a collapse in the polynomial hierarchy (see [39] for formal definition). The proof was provided by Fortnow and Santhanam in private communication, and is differed to the Appendix due to space limitation.

Lemma 1 ([27]). *If any* NP*-complete problem has a distillation algorithm then* $\mathrm{PH} = \Sigma_{\mathrm{p}}^3$.

Proof. Let L be an NP-complete problem with a distillation algorithm $\mathcal{A}$, and let $\overline{L}$ denote the complement of L. We show that using $\mathcal{A}$, we can design a non-deterministic Turing-machine (NDTM) that, with the help of polynomial advice, can decide $\overline{L}$ in polynomial-time. This will that show coNP $\subseteq$ NP/poly, and combined with Yap's theorem [41] (coNP $\subseteq$ NP/poly $\Rightarrow$ PH $\subseteq \Sigma_{\mathrm{p}}^3$), this will prove the statement in the theorem.

Let $n \in \mathbb{N}$ be a sufficiently large integer. Denote by $\overline{L}_n$ the subset of strings of length at most n in the complement of L, *i.e.* $\overline{L}_n = \{x \notin L : |x| \leq n\}$. By its definition, given any $x_1, \ldots, x_t \in \overline{L}_n$, the distillation algorithm $\mathcal{A}$ maps $(x_1, \ldots, x_t)$ to some $y \in \overline{L}_{n^c}$, where c is some constant independent of t. Any sequence containing a string $x_i \notin \overline{L}_n$ is mapped to a string $y \notin \overline{L}_{n^c}$. The main part of the proof consists in showing that there exists a set $S_n \subseteq \overline{L}_{n^c}$, with $|S_n|$ polynomially bounded in n, such that for any $x \in \Sigma^{\leq n}$ we have the following:

- If $x \in \overline{L}_n$, then there exist strings $x_1, \ldots, x_t \in \Sigma^{\leq n}$ with $x_i = x$ for some i, $1 \leq i \leq t$, such that $\mathcal{A}(x_1, \ldots, x_t) \in S_n$.
- If $x \notin \overline{L}_n$, then for all strings $x_1, \ldots, x_t \in \Sigma^{\leq n}$ with $x_i = x$ for some i, $1 \leq i \leq t$, we have $\mathcal{A}(x_1, \ldots, x_t) \notin S_n$.

Given such a set $S_n \subseteq \overline{L}_{n^c}$ as advice, a NDTM M can decide whether a given $x \in \Sigma^{\leq n}$ is in $\overline{L}$ as follows: It first guesses t strings $x_1, \ldots, x_t \in \Sigma^{\leq n}$, and checks whether one of them is x. If not, it immediately rejects. Otherwise, it computes $\mathcal{A}(x_1, \ldots, x_t)$, and accepts iff the output is in S_n. It is immediate to verify that M correctly determines (in the non-deterministic sense) whether $x \in \overline{L}_n$. In the remaining part of the proof, we show that there exists such an advice $S \subseteq \overline{L}_{n^c}$ as required above.

We view $\mathcal{A}$ as a function mapping strings from $(\overline{L}_n)^t$ to $\overline{L}_{n^c}$, and say a string $y \in \overline{L}_{n^c}$ *covers* a string $x \in \overline{L}_n$ if there exist $x_1, \ldots, x_t \in \Sigma^{\leq n}$ with $x_i = x$ for some i, $1 \leq i \leq t$, and with $\mathcal{A}(x_1, \ldots, x_t) = y$. Clearly, our goal is to find polynomial-size subset of $\overline{L}_{n^c}$ which covers all strings in $\overline{L}_n$. By the pigeonhole principle, there is a string $y \in Y$ for which $\mathcal{A}$ maps at least $|(\overline{L}_n)^t|/|\overline{L}_{n^c}| = |\overline{L}_n|^t/|\overline{L}_{n^c}|$ tuples in $(\overline{L}_n)^t$ to. Taking the t'th square root, this gives us $|\overline{L}_n|/|\overline{L}_{n^c}|^{1/t}$ distinct strings in $\overline{L}_n$ which are covered by y. Hence, by letting $t = \lg |\overline{L}_{n^c}| = O(n^c)$, this gives us a constant fraction of the strings in $\overline{L}_n$. It follows that we can repeat this process recursively in order to cover all strings in $\overline{L}_n$ with a polynomial number of strings in $\overline{L}_{n^c}$. □

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 the maximum of all parameters in 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 parametric-distillations 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 2. *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, we have

$$\begin{aligned}
\Psi(\textstyle\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i)) \in \widetilde{L} &\iff \textstyle\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i) \in \text{SAT} \\
&\iff \exists i, 1 \leq i \leq r : \Phi(\widetilde{z}_i) \in \text{SAT} \\
&\iff \exists i, 1 \leq i \leq r : \widetilde{z}_i \in \widetilde{L} \\
&\iff \exists i, 1 \leq i \leq r : (z_i, k''_i) \in L \\
&\iff \exists i, 1 \leq i \leq r : (y_i, k'_i) \in L \\
&\iff \exists i, 1 \leq i \leq t : (x_i, k_i) \in L \\
&\iff \exists i, 1 \leq i \leq t : \widetilde{x}_i \in \widetilde{L}.
\end{aligned}$$

Furthermore, as each step in the algorithm runs in polynomial-time in the total size of its input, and since the output of each step is the input of the next step, 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 $\Psi(\bigvee_{1 \leq i \leq r} \Phi(\widetilde{z}_i))$. □

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. Due to space considerations, most of the proofs in this section are differed to the Appendix.

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 $(G_1, k) \in L \vee (G_2, k) \in L \iff (G_1 \cup G_2, k) \in L$, where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then L is compositional.*

Proof. Given $(G_1, k), \ldots, (G_t, k)$, take G to be the disjoint union $G_1 \cup \cdots \cup G_t$. The instance (G, k) satisfies all requirements of Definition 3. □

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 [37], and is again compositional according to Lemma 3.

In fact, Lemma 3 implies that any parameterized problem which asks to determine whether a given graph H is a "subgraph of some kind" of another given graph G, for almost

any natural notion of subgraph, is compositional when parameterized by the size of H. 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 the numeric encoding of H (*i.e.* the position of H in some canonical ordering of simple graphs). 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 [19].

As an example of a non graph-theoretic problem which is compositional, consider the parameterized variant of Cook's generic NP-complete problem [16] – the k, σ-Short Non-Deterministic Turing Machine Computation problem. In this problem, we receive as input a non-deterministic Turing machine M with alphabet-size σ, and an integer k, and the goal is to determine in FPT-time, with respect to both k and σ, whether M has a computation path halting on the empty input in at most k steps. This problem can be shown to be in FPT by applying the algorithm which exhaustively checks all global configurations of M [13].

Lemma 4. *k, σ-Short NonDeterministic Turing Machine Computation is compositional.*

Proof. Given $(M_1, k, \sigma), \ldots, (M_t, k, \sigma)$, we can assume that the alphabet of each M_i, $1 \leq i \leq t$, is $\{1, \ldots, \sigma\}$. We create a new NDTM M, which is the disjoint union of all M_i's, in addition to a new unique initial state which is connected the initial states of all M_i by an ε-edge. (That is, by a non-deterministic transition that does not write anything on the tape, nor moves the head.) Note that M has alphabet size σ. Letting $k' = 1 + k$, the instance (M, k', σ) satisfies all requirements of Definition 3. □

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. We call the unparameterized variant of the problem above 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 work with a 'guarantee' version of this problem.

w-Independent Set Refinement:

Instance: A graph G, a tree-decomposition $\mathcal{T}$ of G, and an independent set I in G.
Question: Does G have an independent set of size $|I|+1$?
Parameter: 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 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 the set of new vertices I to each node in $\mathcal{T}$.

Lemma 5. *w-Independent Set Refinement is compositional, and furthermore, if w-Independent Set has a polynomial kernel then so does w-Independent Set Refinement.*

Proof. To prove the first part of lemma, suppose we are given t instances $(G_1, \mathcal{T}_1, I_1), \ldots, (G_t, \mathcal{T}_t, I_t)$ of w-Independent Set Refinement. Consider the algorithm which maps this sequence of instances to $(G, \mathcal{T}, I)$, with G the disjoint union $\bigcup_{i=1}^{t} G_i$, $\mathcal{T}$ the tree obtained by connecting all $\mathcal{T}_i$'s, $1 \leq i \leq t$, and with $I = \bigcup_{i=1}^{t} I_i$. Note that G has an independent set of size $|I|+1$ if and only if there exists an i, $1 \leq i \leq t$, such that G_i an independent set of size $|I_i|+1$. Moreover, as the width of each tree-decomposition $\mathcal{T}_i$ is w, $1 \leq i \leq t$, $\mathcal{T}$ also has width w.

We next show that a polynomial kernel for w-Independent Set implies a polynomial kernel for w-Independent Set Refinement. For this, suppose w-Independent Set has a polynomial kernel, and consider a given instance $(G, \mathcal{T}, I)$ of w-Independent Set Refinement. Forgetting I, we create an equivalent instance $(G, \mathcal{T})$ of w-Independent Set, and apply the polynomial kernelization algorithm on this instance to obtain the instance $(G', \mathcal{T}')$, with $|G'|$ and $|\mathcal{T}'|$ polynomially bounded by the width w of $\mathcal{T}$. We now consider the instance $(G', \mathcal{T}')$ as an instance of the unparameterized Independent Set with Treewidth problem. Using the reduction discussed above, we transform $(G', \mathcal{T}')$ in polynomial-time to an equivalent instance $(G'', \mathcal{T}'', I'')$ of Independent Set Refinement. The parameterized instance $(G'', \mathcal{T}'', I'')$ is equivalent to $(G, \mathcal{T}, I)$, and has size polynomial in the width w of $\mathcal{T}$. □

The proof of the lemma above 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, we prove 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. Note that an instance of Dominating Set Refinement with Treewidth consists of a graph G, a tree decomposition $\mathcal{T}$ of G, and a dominating set $D \subseteq V(G)$, and the goal is to determine whether there exists a dominating set in G of size $|D|-1$.

Lemma 6. Dominating Set Refinement with Treewidth *is* NP*-complete.*

Proof. We prove the lemma by showing that Dominating Set Refinement is NP-complete via a reduction from Dominating Set. The fact that Dominating Set Refinement with Treewidth is NP-complete will follow immediately.

Let (G, k) be an instance of Dominating Set. We construct an instance (G', D) of Dominating Set Refinement by creating $k+1$ copies $v_1, \ldots, v_{k+1} \in V(G')$ of each vertex $v \in V(G)$, and then connecting all pairs of vertices:

- $\{v_i, v_j\}$, for all $1 \leq i < j \leq k+1$ and all $v \in V(G)$,
- $\{u_i, v_i\}$, for all $1 \leq i \leq k+1$ and all $u \neq v \in V(G)$,
- $\{u_i, v_j\}$, for all $1 \leq i < j \leq k+1$ and $\{u, v\} \in E(G)$.

The "guaranteed" $(k+1)$-dominating set of G' is taken as $D = \{v_1, \ldots, v_{k+1}\}$ for some arbitrary $v \in V(G)$. We argue that G has a k-dominating set iff G' has a k-dominating set. First, if G has a k-dominating set $D_G \subseteq V(G)$, then $\{v_1 \mid v \in D_G\}$ is a k-dominating set of G'. Conversely, if $D_{G'} \subseteq V(G')$ is a k-dominating set of G', then $\{v \mid v_i \in D_{G'} \text{ for some } i\}$ is a k-dominating set of G. This is since there must be an $i \in \{1, \ldots, k+1\}$ with $v_i \notin D_{G'}$ for all $v \in V(G)$, as $|D_{G'}| < k+1$, and therefore for this particular i, any vertex $v_i \in V(G')$ is dominated by some $u_j \in D_{G'}$, $j \in \{1, \ldots, k+1\} \setminus \{i\}$. By our construction, it follows that either $u = v$ or $\{u, v\} \in E(G)$, and so u dominates v in G. □

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 provide a complete 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 then 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 it complement in FPT is compositional.

Lemma 7. *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.*

Proof. Given $(G_1, k), \ldots, (G_t, k)$, take G to be the disjoint union $G_1 \cup \cdots \cup G_t$. Then $(G, k) \in L$ iff $(G_i, k) \in L$ for all i, $1 \leq i \leq t$. But then $(G, k) \in \overline{L}$ iff $(G_i, k) \in \overline{L}$ for any i, $1 \leq i \leq t$. It follows that (G, k) satisfies all requirements of Definition 3. □

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 [7] for formal definitions and FPT algorithms for these problems). Three closely related relatives of these problems are k-Search Number [6, 22], k-Front Size [7], and k-Gate Matrix Layout [23], which all have AND-composition by the lemma above. Lemma 7

also implies that two other famous FPT "width problems" are AND-compositional, namely, k-Cutwidth and k-Modified Cutwidth [6, 22].

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 unparameterized versions of these two problems is NP-complete by recalling that one can color planar graphs with four colors in polynomial-time (see *e.g.* [38]), 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 domatic-colored with two colors in polynomial-time (see *e.g.* [28]). 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.

6 Sub-Exponential Kernels

In this section we turn to explore sub-exponential kernels, *i.e.* kernelization algorithms that output an instance which are sub-exponentially bounded by the parameter of the input instances. We study sub-exponential kernelization from a more structural point of view. In particular, we show that there are parameterized languages solvable in $O(2^k n)$ time, with no kernelization of size $g(k) = 2^{o(k)}$. Furthermore, we show that there are problems with classically derived counterparts outside of P that have arbitrarily small exponential kernels (see definition below). Due to space considerations, the results in this section are presented without proofs.

In [25], Flum, Grohe, and Weyer introduced the notion of "bounded fixed-parameter tractability" as 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 $2^{k^{O(1)}}n^{O(1)}$. It is tempting to think that the classes of problems with such running-times will align themselves with the classes of problems having linear and polynomial kernels respectively. We have already seen evidence in this paper that this attractive idea fails: k-Path can be solved in $2^{O(k)}n^{O(1)}$, but is unlikely to have a polynomial

kernel (Theorem 1). In the following we show that this idea fails in a sharper sense, and without having to rely on any complexity assumption.

Theorem 3. *There is an* FPT *langauge* $L \subseteq \Sigma^* \times \mathbb{N}^+$ *solvable in* $O(2^k n)$ *time,* $n = |x| + k$*, with no kernelization of size* $g(k) = 2^{o(k)}$*.*

Proof. Let $\Phi_1, \Phi_2, \ldots$ denote the set of all kernelization algorithms, and let $g_1, g_2, \ldots$ denote the set of all computable functions which are bounded by $2^{o(k)}$. We will assume that we have a linear-time enumeration of all possible pairs $\langle \Phi, g \rangle$, where for convenience we will actually assume we have an enumeration $\{\langle \Phi_\ell, g_\ell \rangle \mid \ell \in \omega\}$, with each $\langle \Phi, g \rangle$ occurring infinitely-many often. We will say that $\langle \Phi, g \rangle$ is a *proper pair*, if g is in fact the size of Φ. Our argument is via diagonalization. We give an algorithm that decides a langauge $L \subseteq \Sigma^* \times \mathbb{N}^+$ in $O(2^k n)$ time, $n = |x| + k$, where for each proper pair $\langle \Phi, g \rangle$ there is an $(x, k) \in \Sigma^* \times \mathbb{N}^+$ such that

$$(x, k) \notin L \iff \Phi(x, k) \in L. \tag{1}$$

Clearly this will imply that Φ is not a kernelization algorithm of L for all proper pairs $\langle \Phi, g \rangle$.

The langauge L that our algorithm decides will be rather sparse: If there is no $\ell \in \mathbb{N}^+$ with $\ell = \lg \lg k$, then (x, k) will not be in L for all $x \in \Sigma^*$. In other words, the non-empty slices of L will be at least 2^{2^k} far apart. Furthermore, L will only contain pairs $(x, k) \in \Sigma^* \times \mathbb{N}^+$ with $x = 1^{g_\ell(k)}$, for $\ell = \lg \lg k$ and $1 \in \Sigma$. This will ensure us enough time to diagonalize. Our algorithm proceeds in the following steps:

1. Check whether there exists an $\ell \in \mathbb{N}^+$ with $\ell = \lg \lg k$. If not, determine $(x, k) \notin L$.
2. Compute $\langle \Phi_\ell, g_\ell \rangle$ and check whether $x = 1^{g_\ell(k)}$. If not, determine $(x, k) \notin L$.
3. Run Φ_ℓ on input (x, k) for at most $2^k n$ steps. If Φ_ℓ does not terminate, determine $(x, k) \notin L$.
4. Set $(x', k') = \Phi_\ell(x, k)$. If $|x'| + k' > g_\ell(k)$, determine $(x, k) \notin L$.
5. Determine $(x, k) \notin L \iff (x', k') \in L$.

There are two important claims we need to make here. First, we need to argue that our algorithm indeed has a running-time of $O(2^k n)$, as promised above. Second, we need to show that (1) is satisfied for every proper pair $\langle \Phi, g \rangle$. In other words, we need to show that for every proper pair $\langle \Phi, g \rangle$ there is an $\ell \in \omega$ and a pair $(x, k) \in \Sigma^* \times \mathbb{N}^+$ for which Φ_ℓ terminates on in step 3 of the algorithm.

For bounding the running time of our algorithm, first observe that both steps 1 and 2 can be performed in $O(k)$ time, and that steps 3 and 4 together require $O(2^k n)$ time. Step 5 is recursive. To bound the time required for the recursion, first note that step 4 guarantees that $(x, k) \neq (x', k')$, and so the recursion will terminate. (Here is where we actually use the fact that we only need to diagnolize on proper pairs $\langle \Phi, g \rangle$.) Moreover, notice that the only way we will simulate another kernelization algorithm $\Phi_{\ell'}$, for some $\ell' \neq \ell$, is if $k' \leq \lg \lg k$ and if $x' = 1^{g_{\ell'}(k)}$. This implies that the running-time of $\Phi_{\ell'}$ will be

$$2^{k'}(|x'| + k') = 2^{k'}(g_{\ell'}(k') + k') \leq 2^{\lg k}(g_{\ell'}(\lg k) + \lg k) << k \cdot (2^{\lg k} + \lg k) = O(k).$$

It is now easy to see that our algorithm will have less then $\lg^* k$ recursive steps, and in each step the running-time will decrease at least logarithmically. Hence, our algorithm is $O(2^k n)$ time.

To see that (1) is satisfied for every proper pair, let $\langle \Phi, g \rangle$ be a proper pair, and let n^c denote the running-time of Φ on input $(x,k) \in \Sigma^* \times \mathbb{N}^+$, with $n = |x| + k$ and $c \in \mathbb{N}^+$. Since $g(k) = 2^{o(k)}$, for $(x,k) = (1^{g(k)}, k)$ with k sufficiently large, we will have

$$2^k n = 2^k \cdot (g(k) + k) > (g(k) + k)^c = n^c.$$

Let k be an integer for which the above holds. Since $\langle \Phi, g \rangle$ occurs infinitely-many often in $\{\langle \Phi_\ell, g_\ell \rangle \,|\, \ell \in \omega\}$, there is an $\ell \in \omega$ with $\ell > \lg \lg k$ and $\langle \Phi_\ell, g_\ell \rangle = \langle \Phi, g \rangle$. For this ℓ, Φ_ℓ will terminate on $(1^{g_\ell(k)}, k)$ in at most $2^k n$ steps. The theorem follows. □

We next turn to study problems which admit arbitrarily small exponential kernels. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}^+$ has an *arbitrarily small exponential kernel* if it has a kernel of size bound by $(1+\varepsilon)^k$ for any $\varepsilon > 0$. Are there problems outside P with arbitrarily small exponential kernels? Before answering this question, we recall the any FPT algorithm for a parameterized langauge $L \subseteq \Sigma^* \times \mathbb{N}^+$ with running-time $f(k)(|x| + k)^c$ yields a kernelization for L with size $g(k) = f(k)$. To see this, note that when $f(k) \leq |x| + k$ the above algorithm is polynomial, and a trivial canonical $O(1)$ kernel can be computed in this case. When $f(k) > |x| + k$, (x,k) is already kernelized. We will use this simple observation, first noted in [11], to prove the following:

Theorem 4. *There is a parameterized problem, whose classically derived problem is not in* P*, with an arbitrarily small exponential kernel.*

Proof. The proof is similar to the proof of Theorem 3. We construct $L \notin \mathrm{P}$, such that each slice of L has only a single element, with non-empty slices at least 2^{2^k} apart, and such that for each $\varepsilon > 0$ there is an algorithm accepting L in time $O((1+\varepsilon)^k(|x| + k))$. To make sure that $L \notin \mathrm{P}$, we will use for diagonalization a linear-time enumeration $\{\Phi_\ell \,|\, \ell \in \omega\}$ of all polynomial-time procedures, with each procedure occurring in $\{\Phi_\ell \,|\, \ell \in \omega\}$ infinitely-many often, and where $(|x| + k)^{c_\ell}$ the running-time of Φ_ℓ on inputs $(x,k) \in \Sigma^* \times \mathbb{N}^+$ with $|x| + k$ sufficiently large. We will also consider a polynomial-time computable sequence $\{\varepsilon_\ell\}_{\ell \in \omega}$, of descending rational numbers with $\lim_\ell \epsilon_\ell = 0$.

Our algorithm, given $(x,k) \in \Sigma^* \times \mathbb{N}^+$, will automatically determine that $(x,k) \notin L$, if $x \neq 1 \in \Sigma$. For pairs (x,k) with $x = 1$, our algorithm will first check whether there is some $\ell \in \mathbb{N}^+$ with $\ell = \lg \lg k$, and if not, it again determines $(x,k) \notin L$. Otherwise, it computes Φ_ℓ and ε_ℓ, and checks whether Φ_ℓ terminates on (x,k) in at most $(1+\varepsilon_\ell)^k(|x| + k)$ steps. If not, it determines $(x,k) \notin L$, and if so, it determines $(x,k) \in L \iff \Phi_\ell(x,k) \notin L$.

Bounding the running-time of this algorithm can be done as in the proof of Theorem 3. Indeed, the algorithm has less then $\lg^* k$ recursive steps, where in each step the running-time decreases at least logarithmically, implying that the total-running time will be dominated by the first recursive step, *i.e.* by $O((1+\varepsilon)^k(|x| + k))$. Furthermore, for sufficiently large k, we have $(1 + \epsilon_\ell)^k \geq k^{c_\ell}$, and so we will indeed be able to diagonalize on any procedure Φ occurring in $\{\Phi_\ell \,|\, \ell \in \omega\}$. The theorem therefore follows, recalling that an $f(k)n^c$ algorithm implies an $f(k)$ kernelization. □

7 Conclusions

In this paper we presented a generic framework which allowed us to show that a wide variety of FPT problems are unlikely to have a polynomial kernel. Our results form the first

polynomial lower-bounds on kernelization sizes of natural FPT problems, and provide an initial glimpse into what makes polynomial kernelization intrinsically hard in certain type of problems. There are many future directions of research and open questions stemming from our work. To conclude the paper, we give below an incomplete list which contains four of the more important ones:

- First and foremost, in light of Theorem 2, can one relate the non-existence of distillation algorithms for coNP-complete problems to any known complexity conjecture? In this regard, Buhrmann has shown in private communication that there are oracles relative to which no distillation algorithm exists for coNP-complete problems [9].
- Is there a way to base the non-existence of polynomial kernels for any FPT problem on a conjecture in parameterized complexity, *e.g.* FPT $\neq$ XP or FPT $\neq$ W[t] for some $t \in \mathbb{N}^+$?
- In light of the last items of Theorem 1 and Theorem 2, can one give evidence for the non-existence of polynomial kernels for all NP-complete graph problems parameterized by treewidth?
- Finally, can one obtain sub-exponential lower bounds of any form on the kernel sizes of some of the problems discussed in this paper?

Acknowledgements

We would like to thank Lance Fortnow, Raul Santhanam, and Harry Buhrman for many fruitful discussions. In particular, Lance and Raul provided the proof for Lemma 1 of this paper. The fourth author would also like to thank Moritz Müller for reviewing several preliminary versions, and especially for the countless (and sometimes endless) debates on related topics.

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. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.
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. H.L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Department of Computer Science, University of Utrecht, 1986.
7. H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
8. 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.
9. H. Buhrman. Unpublished, 2007.
10. 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.
11. 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.
12. 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.
13. M. Cesati and M. Di Ianni. Computation models for parameterized complexity. *Mathematical Logic Quarterly*, 43:179–202, 1997.
14. J. Chen, H. Fernau, I.A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. In *Proceedings of the 22nd annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 269–280, 2005.
15. Y. Chen, J. Flum, and M. Müller. Lower Bounds for Kernelizations – Manuscript, 2007.
16. S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory Of Computing (STOC)*, pages 151–158, 1971.
17. I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 33–42, 2002.
18. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
19. 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.
20. V. Estivill-Castro, M. Fellows, M. Langston, and F. Rosemond. FPT is P-time extremal structure I. In *Proceedings of the 1st workshop on Algorithms and Complexity in Durham (ACiD)*, pages 1–41, 2005.
21. M.R. Fellows and M.A. Langston. Nonconstructive proofs of polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1988.
22. 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.
23. H. Fernau. *Parameterized algorithms: A graph-theoretic approach*. PhD thesis, 2005.
24. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
25. 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.
26. F.V. Fomin and D.M. Thilikos. Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 581–592, 2004.
27. L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. Technical Report 96, Electronic Colloquium on Computational Complexity, 2007.
28. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

29. 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.
30. J. Guo and R. Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Networks*, 46(3):124–135, 2005.
31. J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.
32. J. Guo and R. Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP) – to appear*, 2007.
33. 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.
34. D.S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
35. G.L. Nemhauser and L.E. Trotter Jr. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.
36. R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1:89–102, 2003.
37. 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.
38. 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.
39. L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
40. K. Weihe. Covering trains by stations or the power of data reduction. In *Proceedings of the 1st ACM/SIAM workshop on ALgorithm ENgineering and EXperiments (ALENEX)*, pages 1–8, 1998.
41. C.K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, 1983.