On problems without polynomial kernels

Hans L. Bodlaender [a], Rodney G. Downey [b,1], Michael R. Fellows [c,2], Danny Hermelin [d,*,3]

[a] *Department of Information and Computing Sciences, Utrecht University, 3508 TB Utrecht 80.089, Netherlands*
[b] *School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, Wellington 600, New Zealand*
[c] *The University of Newcastle, Callaghan NSW 2308, Australia*
[d] *Department of Computer Science, The University of Haifa, Mount Carmel, Haifa 31905, Israel*

ARTICLE INFO

ABSTRACT

Kernelization is a strong and widely-applied technique in parameterized complexity. A kernelization algorithm, or simply a kernel, 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 kernel is polynomial if the size and parameter of the output are polynomially-bounded by the parameter of the input.

In this paper we develop a framework which allows showing that a wide range of FPT problems do not have polynomial kernels. Our evidence relies on hypothesis made in the classical world (*i.e.* non-parametric complexity), and revolves around a new type of algorithm for classical decision problems, called a distillation algorithm, which is of independent interest. Using the notion of distillation algorithms, we develop a generic lower-bound engine that allows us to show that a variety of FPT problems, fulfilling certain criteria, cannot have polynomial kernels unless the polynomial hierarchy collapses. 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 and other structural parameters.

© 2009 Elsevier Inc. All rights reserved.

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 well-studied 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 [24] (see [42] 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 [24]. The class of all fixed-parameter tractable problems is denoted by FPT. The main tower of parameterized complexity classes is

$$\text{FPT} \subseteq \text{M}[1] \subseteq \text{W}[1] \subseteq \text{M}[2] \subseteq \text{W}[2] \subseteq \cdots \subseteq XP.$$

It is known that FPT = M[1] iff n variable 3-SAT can be solved in $2^{o(n)}$ time [13,23,32], and FPT = W[1] iff *k*-STEP NONDETER-MINISTIC TURING MACHINE HALTING is fixed-parameter tractable [22,24,29]. It is also known that *k*-CLIQUE is solvable in time $n^{o(k)}$ if and only if FPT = M[1], and that *k*-DOMINATING SET is solvable in time $n^{o(k)}$ if and only if FPT = M[2] [15,17].

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 [40]. 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 [55]. Nemhauser and Trotter's classical kernelization algorithm for *k*-VERTEX COVER [49] is widely used as a preprocessing step in many approximation algorithms for this problem (see *e.g.* [4,44]).

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 $k' \leqslant k$. 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 misled 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.* [16,26,34,39,41].

It is clear that any (decidable) parameterized 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 [12]. This is seen by considering the two cases $f(k) \geqslant n$ 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 classical kernelization algorithm of Buss for *k*-VERTEX COVER is a polynomial kernel (see *e.g.* [24]), and so *k*-VERTEX COVER has a polynomial kernel. Other problems known to have polynomial kernels include *k*-LEAF SPANNING TREE [9], *k*-FEEDBACK VERTEX SET [7,11,54], *k*-PLANAR DOMINATING SET [1], *k*-CLUSTER EDITING [38], *k*-HITTING SET FOR SETS OF BOUNDED SIZE [50], 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 the unlikely event that W[1] = FPT. So we focus on parameterized problems known to be in FPT. Many examples of such problems can be found among the problems shown to be in FPT using heavy machinery such as *color-coding* [2], the *graph minor technique* [27], 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:

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.[4] 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. We introduce a new framework which allows us to show that many FPT 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 [35], Chen et al. [18,19], and Buhrman [10] have applied the concepts in this paper to other arenas. Also, a team of authors including the first author of this paper have recently extended the framework presented here, via the notion of *polynomial-transformations*, so that it captures additional FPT problems [8].

Obviously, 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 [43] in relation to compression-related cryptographic problems. Our 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 [46]. 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 of a distillation algorithm for any NP-complete problem would imply the collapse of the polynomial hierarchy to the third level, *i.e.* PH = Σ_p^3 [35]. 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 = Σ_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.

---

[4] It was brought to our attention that there are recent improvements to the k-Path algorithm mentioned above which have rather practical running-times [45,48].

We remark that in unpublished work, Buhrman [10] has shown that there are oracles relative to which no coNP-complete problem has a distillation algorithm. We believe that the same information-theoretic intuition applies here, and that no coNP-complete problem can have a distillation algorithm.

Prior to our results, there were limited attempts in showing negative results for kernelization. Below we mention all such results known to us: Flum and Grohe showed that trivial parameterizations lead to problems that clearly do not have polynomial kernels (see Example 9.3 in [32]). Frick and Grohe showed a non-elementary lower-bound on the running-time of any FPT algorithm for the model checking problem for monadic second order logic on trees [36], which implies, along with the fact that the unparameterized version of this problem is in PSPACE, that the problem has no kernel of elementary size unless P = NP. There are also negative results which follow from approximation lower-bounds. For instance, k-Vertex Cover cannot have a kernel of size $1.36 \cdot k$ unless P = NP [40], due to the lower bounds on the approximation factor guarantee of any approximation for the optimization variant of this problem [21]. Finally, there is limited approach for showing linear lower-bounds for some FPT problems called the *dual-parameter approach* [16]. An example result obtained by this approach is that k-Planar Vertex Cover cannot have a kernel of size $(4/3 - \varepsilon) \cdot k$ for any $\varepsilon > 0$ (unless P = NP).

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 $2^{o(k)}$. This relates our material to the work of Flum, Grohe, and Weyer [33] who 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 $2^{k^{O(1)}} n^{O(1)}$. 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 encodings of some combinatorial object, *e.g.* graphs. We will call strings $x \in \Sigma^*$ that 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 [7,24,32].

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 language $\widetilde{L} = \{x\#1^k \mid (x, k) \in L\}$. Note that it is easy to obtain the *derived* parameterized instance (x, k) of L, given the instance $x\#1^k$ of $\widetilde{L}$.

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

- $(x, k) \in L \iff (x', k') \in L$,
- $|x'|, k' \leqslant f(k)$,

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. Of particular interest are *polynomial kernels*. These are kernels with polynomial sizes.

There are also generalizations of the above definition that have appeared in the literature. Most notably is the generalization which allows the kernelization algorithm to map the instance to an instance of a different language.

Definition 2 (*Generalized kernelization*). A *generalized kernelization algorithm* from a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ to another 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

- $(x, k) \in L \iff (x', k') \in L'$,
- $|x'|, k' \leqslant f(k)$,

where f is an arbitrary computable function, and p a polynomial.

While the latter definition can prove useful in certain cases, where one can, for instance, generalize the problem at hand in order to obtain additional combinatorial leverage, the former definition is the more natural one. We therefore present our framework in terms of the former definition, though all our results extend easily to the generalized definition as well.

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. We begin with the central notion of our framework:

Definition 3 (*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 $1 \leqslant i \leqslant t$,
- 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 $1 \leqslant i \leqslant t$.
  2. $|y|$ is polynomial in $\max_{1 \leqslant i \leqslant t} |x_i|$.

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 4 (*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 $1 \leqslant i \leqslant t$,
- 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 $1 \leqslant i \leqslant t$.
  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 polynomially-bounded by the maximum size of all instances. The reader should observe that we require the parameter to be equal in all instances given to the composition algorithm.

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 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 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 $1 \leqslant i \leqslant t$. Since $\widetilde{L}$ is NP-complete, there exist two polynomial-time transformations $\Phi : \widetilde{L} \to$ SAT and $\Psi :$ 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 $k = \max_{1 \leqslant i \leqslant t} k_i$. 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 $1 \leqslant \ell \leqslant k$. 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), $1 \leqslant i \leqslant r$, is the instance obtained by composing all instances with parameters equaling the ith 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 $1 \leqslant i \leqslant r$. 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.* $\Psi(\bigvee_{1 \leqslant i \leqslant r} \Phi(\widetilde{z}_i))$.

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

$$
\begin{aligned}
\Psi\left(\bigvee_{1 \leqslant i \leqslant r} \Phi(\widetilde{z}_i) \right) \in \widetilde{L} \quad &\Longleftrightarrow \quad \bigvee_{1 \leqslant i \leqslant r} \Phi(\widetilde{z}_i) \in \text{SAT} \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant r \colon \Phi(\widetilde{z}_i) \in \text{SAT} \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant r \colon \widetilde{z}_i \in \widetilde{L} \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant r \colon (z_i, k''_i) \in L \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant r \colon (y_i, k'_i) \in L \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant t \colon (x_i, k_i) \in L \\
&\Longleftrightarrow \quad \exists i, 1 \leqslant i \leqslant t \colon \widetilde{x}_i \in \widetilde{L}.
\end{aligned}
$$

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 $n = \max_{1 \leqslant i \leqslant t} |\widetilde{x}_i|$.

The first observation is that since each $\widetilde{x}_i$ is derived from the instance (x_i, k_i), $1 \leqslant i \leqslant t$, we have $r \leqslant k = \max_{1 \leqslant i \leqslant t} k_i \leqslant \max_{1 \leqslant i \leqslant t} |\widetilde{x}_i| = n$. 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), $1 \leqslant i \leqslant r$, is obtained via composition, we know that k'_i is bounded by some polynomial in $\ell \leqslant k \leqslant n$. Hence, since for each $1 \leqslant i \leqslant r$, 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 $\Psi(\bigvee_{1 \leqslant i \leqslant r} \Phi(\widetilde{z}_i))$. $\quad\Box$

We conclude this section by stating a lemma proven by Fortnow and Santhanam [35], which verifies our initial intuition that NP-complete problems are unlikely to have distillation algorithms. Fortnow and Santhanam showed that a distillation algorithm for an NP-complete problem would imply the collapse of the polynomial hierarchy [53], a hierarchy believed by many to be proper. Note that it is clear from Definition 3, 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 $\widetilde{L}$. In the following we use PH to denote the polynomial hierarchy, and Σ_{p}^3 to denote its third level.

Lemma 2. *(See [35].) If any* NP-*complete problem has a distillation algorithm then* PH $= \Sigma_{\text{p}}^3$.

4. Applications

Lemma 1, the core of our lower bound engine together, implies that any compositional parameterized problem whose unparameterized counterpart is NP-complete is unlikely to have a polynomial kernel. 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 $(G_1, k) \in L \lor (G_2, k) \in L \Longleftrightarrow (G_1 \cup G_2, k) \in L$, 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 4. $\quad \square$

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 exceeding some given threshold. This problem is in FPT due to [51], and is again compositional according to Lemma 3. This is a good place to point out why we require the parameter to be equal in all input instances of a composition algorithm. If this were not the case, then the disjoint union algorithm of Lemma 3 above would simply not work.

Corollary 1. *k-Path, k-Cycle, k-Exact Cycle, and k-Short Cheap Tour are all compositional.*

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, $1 \leqslant i \leqslant t$. 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 [25].

Corollary 2. *k-Graph Minor Order Test, k-Bounded Treewidth Subgraph Test, k-Planar Graph Subgraph Test, and k-Planar Graph Induced Subgraph Test are all compositional.*

As an example of a non-graph-theoretic problem which is compositional, consider the parameterized variant of Cook's generic NP-complete problem [20] – the k, σ-Short NonDeterministic 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 [14].

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, $1 \leqslant i \leqslant t$, 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 4. $\quad \square$

We now turn to show that many natural NP-complete problems parameterized by treewidth also fall easily into our framework. In particular, we prove the last item of Theorem 1. We begin with the w-Independent Set problem:

w-Independent Set:

Instance: A graph G, a tree-decomposition $\mathcal{T}$ of G, and a positive integer k.
Question: Does G have an independent set of size k?
Parameter: The width w of $\mathcal{T}$.

Note that the parameter in w-Independent Set 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 'refinement' version of this problem where the input is augmented by a solution I, and the goal is to find a solution of size $|I| + 1$:

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 $k - 1$ 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 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, $1 \leqslant i \leqslant t$, 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, $1 \leqslant i \leqslant t$, such that G_i has an independent set of size $|I_i| + 1$. Moreover, as the width of each tree-decomposition $\mathcal{T}_i$ is w, $1 \leqslant i \leqslant t$, $\mathcal{T}$ 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, maximum degree, minimum 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 $1 \leqslant i < j \leqslant k + 1$ and all $v \in V(G)$,
- $\{u_i, v_i\}$, for all $1 \leqslant i \leqslant k + 1$ and all $u \neq v \in V(G)$,
- $\{u_i, v_j\}$, for all $1 \leqslant i < j \leqslant k + 1$ 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 $\{v \mid v_i \in D_{G'}$ 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 $\bar{L}$, 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 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 $\bar{L}$, 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, $1 \leqslant i \leqslant t$. But then $(G, k) \in \bar{L}$ iff $(G_i, k) \in \bar{L}$ for any i, $1 \leqslant i \leqslant t$. It follows that (G, k) satisfies all requirements of Definition 4. □

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 [5] for formal definitions and FPT algorithms for these problems). Three closely related relatives of these problems are k-Search Number [6,28], k-Front Size [5], and k-Gate Matrix Layout [30], 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 [6,28].

Corollary 3. *The complements of k-Cutwidth, k-Modified Cutwidth, k-Search Number, k-Pathwidth, k-Treewidth, k-Branchwidth, k-Gate Matrix Layout, and k-Front Size are all compositional.*

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.* [52]), 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.* [37]). 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 instances 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 $g(k) = 2^{o(k)}$.

In [33], 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 language $L \subseteq \Sigma^* \times \mathbb{N}^+$ 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. 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 $\{\langle \Phi_\ell, g_\ell \rangle \mid \ell \in \mathbb{N}\}$, 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 $2^{o(k)}$, and in fact bounds the size of Φ. Our argument is via a looking-back diagonalization argument in the spirit of Ladner's Theorem [47]. We give an algorithm that decides a language $L \subseteq \Sigma^* \times \mathbb{N}^+$ 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

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

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

The language L that our algorithm decides will be rather sparse: If there is no $\ell \in \mathbb{N}^+$ with $\ell = \lg k$, 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^k far apart. Furthermore, L will only contain pairs $(x, k) \in \Sigma^* \times \mathbb{N}^+$ with $x = 1^{g_\ell(k)}$, for $\ell = \lg k$ 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 $\ell = \lg k$. If not, determine $(x, k) \notin L$.
2. Compute $\langle \Phi_\ell, g_\ell \rangle$.
3. Check whether $g_\ell(k) \leqslant 2^k$. If not, determine $(x, k) \notin L$.
4. Check whether $x = 1^{g_\ell(k)}$. If not, determine $(x, k) \notin L$.
5. Run Φ_ℓ on input (x, k) for at most $2^k n$ steps. If Φ_ℓ does not terminate, determine $(x, k) \notin L$.
6. Set $(x', k') = \Phi_\ell(x, k)$. If $|x'| + k' > |x| = g_\ell(k)$, determine $(x, k) \notin L$.
7. 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 $\ell \in \mathbb{N}$ 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 steps 1 through 6 together require $O(2^k n)$ time. Step 7 is recursive. To bound the time required for the recursion, first note that step 6 guarantees that $|x'| + k' \leqslant |x|$, and so the recursion will terminate. Moreover, notice that the only way we will simulate another kernelization algorithm $\Phi_{\ell'}$, for some $\ell' \neq \ell$, is if $k' \leqslant \lg k$ and if $x' = 1^{g_{\ell'}(k)}$. This implies that the running-time of $\Phi_{\ell'}$ will be

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

It is now easy to see that our algorithm will have at most $\lg^* k$ 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

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

Let k be an integer for which the above holds. Since $\langle \Phi, g \rangle$ occurs infinitely-many often in $\{\langle \Phi_\ell, g_\ell \rangle \mid \ell \in \mathbb{N}\}$, there is an $\ell \in \mathbb{N}$ with $\ell > \lg k$ 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. $\square$

7. Conclusions

In this paper we presented a framework that allows us to show that a wide variety of FPT problems are unlikely to have a polynomial kernel. This framework gives the first polynomial lower-bounds on kernelization sizes of natural FPT problems, and provides an initial glimpse into what makes polynomial kernelization intrinsically hard in certain type of problems. Central to our framework is the notion of distillation for NP-hard problems, which appears to be of interest in other areas of complexity theory [10,18,19,35].

There are many future directions of research and open questions stemming from our work. To conclude the paper, we discuss five 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, Buhrman has shown in private communication that there are oracles relative to which no distillation algorithm exists for coNP-complete problems [10].
- In a recent paper [31], Fernau et al. show that the k-Leaf-Out-Branching problem does not admit a polynomial kernel (unless PH collapses) via a non-trivial use of the framework presented here. However, they show that the problem admits a polynomial *Turing-kernelization*, *i.e.* a polynomial (in n) number of independent polynomial kernels. This raises the obvious question whether one can obtain any lower bounds on Turing-kernelizations for natural problems.

- 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 $\neq$ 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 super-polynomial lower bounds of any form on the kernel sizes of some of the problems discussed in this paper?

Acknowledgments

References

[1] J. Alber, M.R. Fellows, R. Niedermeier, Polynomial-time data reduction for dominating set, J. ACM 51 (3) (2004) 363–384.
[2] N. Alon, R. Yuster, U. Zwick, Color coding, J. ACM 42 (4) (1995) 844–856.
[3] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey, BIT 25 (1) (1985) 2–23.
[4] R. Bar-Yehuda, S. Even, A local-ratio theorem for approximating the weighted vertex cover problem, in: Ann. Discrete Math., vol. 25, 1985, pp. 27–46.
[5] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[6] H.L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, Theoret. Comput. Sci. 209 (1–2) (1998) 1–45.
[7] H.L. Bodlaender, A cubic kernel for feedback vertex set, in: Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2007, pp. 320–331.
[8] H.L. Bodlaender, S. Thomassé, A. Yeo, Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels, Technical Report UU-CS-2008-030, Department of Information and Computing Sciences, Utrecht University, 2008.
[9] P.S. Bonsma, T. Brüggemann, 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), 2003, pp. 259–268.
[10] H. Buhrman, private communication, 2007.
[11] K. Burrage, V. Estivill-Castro, M.R. Fellows, M.A. Langston, S. Mac, 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), 2006, pp. 192–202.
[12] L. Cai, J. Chen, R.G. Downey, M.R. Fellows, Advice classes of parameterized tractability, Ann. Pure Appl. Logic 84 (1) (1997) 119–138.
[13] L. Cai, D.W. Juedes, Subexponential parameterized algorithms collapse the W-hierarchy, in: Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP), 2001, pp. 273–284.
[14] M. Cesati, M. Di Ianni, Computation models for parameterized complexity, MLQ Math. Log. Q. 43 (1997) 179–202.
[15] J. Chen, B. Chor, M. Fellows, X. Huang, D.W. Juedes, I. Kanj, G. Xia, Tight lower bounds for certain parameterized NP-hard problems, in: Proc. of the 19th Annual IEEE Conference on Computational Complexity (CCC), 2004, pp. 150–160.
[16] J. Chen, H. Fernau, I.A. Kanj, 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), 2005, pp. 269–280.
[17] J. Chen, X. Huang, I.A. Kanj, G. Xia, Linear FPT reductions and computational lower bounds, in: Proc. of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 212–221.
[18] Y. Chen, J. Flum, M. Müller, Lower bounds for kernelizations, manuscript, 2007.
[19] Y. Chen, M. Müller, The price of one sided gaps, manuscript, 2008.
[20] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC), 1971, pp. 151–158.
[21] I. Dinur, S. Safra, The importance of being biased, in: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), 2002, pp. 33–42.
[22] R.G. Downey, Parameterized complexity for the skeptic, in: Proceedings of the 18th IEEE Annual Conference on Computational Complexity, 2003, pp. 147–170.
[23] R.G. Downey, V. Estivill-Castro, M.R. Fellows, E. Prieto, F.A. Rosamond, Cutting up is hard to do: The parameterized complexity of k-cut and related problems, in: Electron. Notes Theor. Comput. Sci., vol. 78, 2003.
[24] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer-Verlag, 1999.
[25] D. Eppstein, Subgraph isomorphism in planar graphs and related problems, in: Proceedings of the 6th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 632–640.
[26] V. Estivill-Castro, M. Fellows, M. Langston, F. Rosemond, FPT is P-time extremal structure I, in: Proc. of the 1st Workshop on Algorithms and Complexity in Durham (ACiD), 2005, pp. 1–41.
[27] M.R. Fellows, M.A. Langston, Nonconstructive proofs of polynomial-time complexity, Inform. Process. Lett. 26 (1988) 157–162.
[28] M.R. Fellows, M.A. Langston, On well-partial-order theory and its application to combinatorial problems of VLSI design, SIAM J. Discrete Math. 5 (1) (1992) 117–126.
[29] M.R. Fellows, F. Rosamond, Why is P not equal to NP? in: Proceedings of the 3rd Conference on Computability in Europe (CiE), 2007, pp. 151–160.
[30] H. Fernau, Parameterized algorithms: A graph-theoretic approach, PhD thesis, 2005.
[31] H. Fernau, F.V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, Y. Villanger, Kernel(s) for problems with no kernel: On out-trees with many leaves, in: Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2009, pp. 421–432.
[32] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer-Verlag, 2006.
[33] J. Flum, M. Grohe, 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), 2004, pp. 555–567.
[34] F.V. Fomin, 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), 2004, pp. 581–592.
[35] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, in: Proceedings of the 40th Annual Symposium on the Theory of Computing (STOC), 2008, pp. 133–142.
[36] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, Ann. Pure Appl. Logic 130 (1–3) (2004) 3–31.
[37] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[38] J. Gramm, J. Guo, F. Huffner, R. Niedermeier, Graph-modeled data clustering: Exact algorithms for clique generation, Math. Syst. Theory 38 (2005) 373–392.
[39] J. Guo, R. Niedermeier, Fixed-parameter tractability and data reduction for multicut in trees, Networks 46 (3) (2005) 124–135.
[40] J. Guo, R. Niedermeier, Invitation to data reduction and problem kernelization, SIGACT News 38 (1) (2007) 31–45.
[41] J. Guo, 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), 2007, pp. 375–386.
[42] J. Guo, R. Niedermeier, S. Wernicke, Parameterized complexity of generalized vertex cover problems, in: Proceedings of the 9th Workshop on Algorithms and Data Structures (WADS), 2005, pp. 36–48.
[43] D. Harnik, M. Naor, On the compressibility of NP instances and cryptographic applications, in: Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006, pp. 719–728.
[44] D.S. Hochbaum, Efficient bounds for the stable set, vertex cover and set packing problems, Discrete Appl. Math. 6 (1983) 243–254.
[45] J. Kneis, D. Mölle, S. Richter, P. Rossmanith, Divide-and-color, in: Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG), 2006, pp. 58–67.
[46] K. Ko, On self-reducibility and weak P-selectivity, J. Comput. System Sci. 26 (2) (1983) 209–221.
[47] R. Ladner, On the structure of polynomial time reducibility, J. ACM 22 (1) (1975) 155–171.
[48] Y. Liu, S. Lu, J. Chen, 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), 2006, pp. 84–95.
[49] G.L. Nemhauser, L.E. Trotter Jr, Vertex packings: Structural properties and algorithms, Math. Program. 8 (1975) 232–248.
[50] R. Niedermeier, P. Rossmanith, An efficient fixed-parameter algorithm for 3-hitting set, J. Discrete Algorithms 1 (2003) 89–102.
[51] J. Plehn, B. Voigt, Finding minimally weighted subgraphs, in: Proceedings of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), 1990, pp. 18–29.
[52] N. Robertson, D.P. Sanders, P. Seymour, R. Thomas, Efficiently four-coloring planar graphs, in: Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC), 1996, pp. 571–575.
[53] L.J. Stockmeyer, The polynomial-time hierarchy, Theoret. Comput. Sci. 3 (1977) 1–22.
[54] S. Thomassé, A quadratic kernel for feedback vertex set, in: Proceedings of the 20th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA), 2009, pp. 115–119.
[55] 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), 1998, pp. 1–8.