Lower Bounds on Kernelization

Neeldhara Misra, Venkatesh Raman, and Saket Saurabh
The Institute of Mathematical Sciences
CIT Campus, Chennai 600113
{neeldhara|vraman|saket}@imsc.res.in

Abstract

Preprocessing (data reduction or kernelization) to reduce instance size is one of the most commonly deployed heuristics in the implementation practice to tackle computationally hard problems. However, a systematic theoretical study of them remained elusive so far. One of the reasons for this is that if an input to an NP-hard problem can be processed in polynomial time to an equivalent one of smaller size in general, then the preprocessing algorithm can be used to actually *solve* the problem in polynomial time proving $P = NP$, which is expected to be unlikely. However the situation regarding systematic study changed drastically with the advent of parameterized complexity. Parameterized complexity provides a natural framework to analyse preprocessing algorithms. In a parameterized problem, every instance x comes with a positive integer, or parameter, k. The problem is said to admit a kernel if, in polynomial time, we can reduce the size of the instance x to a function in k, while preserving the answer.

The central notion in parameterized complexity is *fixed parameter tractability (FPT)*, which is the notion of solvability in $f(k) \cdot p(|x|)$ time for any given instance (x, k), where f is an arbitrary function of the parameter k and p is a polynomial in the input size $|x|$. It is folklore that a parameterized problem Π is fixed-parameter tractable if and only if there exists a computable function $g(k)$ such that Π admits a kernel of size $g(k)$. However, the kernels obtained by this theoretical result are usually of exponential (or even worse) sizes, while problem-specific data reductions often achieve quadratic- or even linear-size kernels. So a natural question for any concrete FPT problem is whether it admits polynomial time kernelization to a problem kernel that in the worst case is bounded by a polynomial function of the parameter. Despite several attempts, there are fixed-parameter tractable problems that have only exponential sized kernels. An explanation was provided in a paper by Bodlaender et al. [BDFH09] where it was shown that unless $coNP \subseteq NP/poly$, there are fixed-parameter tractable problems that cannot have a polynomial sized kernel. This triggered further work on showing lower bounds of kernels, and this article surveys recent developments in the area starting from the framework developed in the paper by Bodlaender et al. [BDFH09].

1 Introduction

In attacking computationally hard problems, generally NP-hard problems, it is common in practice to attempt "reducing" the input instance using efficient preprocessing. This generally involves obtaining, from the given instance, an equivalent one that is simpler. To make this notion precise, well-defined measures of efficiency and simplicity need to be established. In this survey, the conventional benchmark of *polynomial time computability* is used for the notion of

efficiency. As for simplicity, we restrict ourselves to considerations of *size*. The attempts described will thus be concentrated on making the instance size as small as possible in polynomial time.

Many input instances have the property that they consist of some parts that are relatively easy to handle, and other parts that form the "really hard" core of the problem. The data reduction paradigm aims to efficiently "cut away easy parts" of the given problem instance and to produce a new and size-reduced instance where exhaustive search methods and other cost-intensive algorithms can be applied.

A striking example of the extent to which data reduction is effective was given by Weihe [Wei98, AFN04], when dealing with the NP-complete Red/Blue Dominating Set problem appearing in the context of the European rail road network. In a preprocessing phase, two simple data reduction rules were applied again and again until no further application was possible. The impressive result of this empirical study was that each of these real-world instances were broken into very small pieces such that for each of these a simple brute-force approach was sufficient to solve the hard problems quite efficiently.

For a long time, the mathematical analysis of polynomial time preprocessing algorithms was neglected. The basic reason for this was that if we seek to start with an instance I of an NP-hard problem and try to find an efficient polynomial time subroutine to replace I with an equivalent instance I' with $|I'| < |I|$ then by repeated application of this subroutine, we can actually solve the problem leading to the unlikely consequence of $P = NP$.

However the situation changed drastically with the advent of parameterized complexity. Parameterized complexity provides a natural framework to analyse preprocessing algorithms. In a parameterized problem, every instance I comes with a positive integer k. The problem is said to admit a kernel if, in polynomial time, we can reduce the size of the instance I to a function in k, while preserving the answer. The central notion in parameterized complexity is *fixed-parameter tractability (FPT)*, which means solvability in time $f(k) \cdot p(|x|)$ for any given instance (x, k), where f is an arbitrary function of the parameter k and p is a polynomial in the input size $|x|$. It is folklore[Nie06] that a parameterized problem Π is FPT if and only if there exists a computable function $g(k)$ such that Π admits a kernel of size $g(k)$. However, the kernels obtained by this theoretical result are usually of exponential (or even worse) sizes. So a natural question for any concrete FPT problem is whether it admits polynomial time kernelization to a problem kernel that in the worst case is bounded by a polynomial function of the parameter. Problem-specific data reductions often achieve quadratic- or even linear-size kernels[Nie06, GN07, Tho09]. Prominent earlier known efficient kernels include a $2k$ vertex kernel for Vertex Cover [CKJ01], and $67k$ vertex kernel for Planar Dominating set [CFKX07]. It was impressive and surprising that for these problems one can reduce the given input to an equivalent one on $O(k)$ vertices in polynomial time.

Despite several attempts, there are fixed-parameter tractable problems that have only exponential sized kernels. Some earlier attempts to prove lower bounds on kernel sizes related the kernel sizes to the approximation ratio of the problems [CFKX07], see also [Nie06]. A breakthrough was provided in a paper by Bodlaender et al. [BDFH09] where it was shown that under reasonable complexity-theoretic assumptions, there are fixed-parameter tractable problems that cannot have a polynomial sized kernel. This result led to a flurry of research in this area leading to kernelization results for concrete parameterized problems and breakthrough results on kernel lower bound techniques. The aim of this survey is to introduce the notion of what is usually meant by efficient data reduction, and survey the literature that gives us insight into when not to expect efficient data reductions, in the framework of parameterized complexity.

Organization: The survey is organized as follows. We first introduce our notations and a few definitions from the field of parameterized complexity in Section 2. In Section 3, we define notions of kernels, polynomial kernels and give an example of polynomial kernel through Vertex Cover.

In section 4 we explain the framework of Bodlaender et al. [BDFH09] that allows us to show when a parameterized problem does not admit polynomial size kernels. In particular, the result rules out the possibility of polynomial size kernels for problems that admit "*composition algorithms*" under complexity theoretic assumptions.

There has been much work in recent times towards showing composition algorithms for many specific problems. We survey some of them in Section 5, and observe that there seems to be a striking similarity among these algorithms. In Section 6, we describe certain restricted fixed-parameter transformations that give us hardness of polynomial kernelization without having to design compositions.

In Section 7, we survey more recent developments including those relating to finer classification of problems having polynomial size kernels and those that don't.

2 Notations and Basic Definitions

2.1 Conventions

The set of natural numbers (that is, nonnegative integers) is denoted by $\mathbb{N}$. For a natural number n let $[n] := \{1, \ldots, n\}$. By $\log n$ we mean $\lceil \log n \rceil$ if an integer is expected. We refer the reader to [Die00] for details on standard graph theoretic notation and terminology we use in the paper.

2.2 Parameterized Complexity

Fixed-parameter tractable algorithms are a class of exact algorithms where the exponential blow up in the running time is restricted to a small parameter associated with the input. Formally the notions of parameterized problem and fixed-parameter tractable algorithms are defined as follows.

Definition 1. [FG06] Let Σ be a fixed finite alphabet. A *parameterized decision problem* is a pair (Q, κ), where $Q \subseteq \Sigma^*$ is a language, and $\kappa : \Sigma^* \to \mathbb{N}$ is a function which is polynomial time computable. The image of a string under κ is called the *parameter* of the problem.

Definition 2. [FG06] A parameterized problem (Q, κ) is *fixed-parameter tractable* if there exists an algorithm that decides in $f(\kappa(x)) \cdot n^{O(1)}$ time whether $x \in Q$, where $n := |x|$ and f is a computable function that does not depend on n. The algorithm is called a *fixed-parameter algorithm* for the problem. The complexity class containing all fixed-parameter tractable problems is called FPT.

There is also a hierarchy of intractable parameterized problem classes above FPT, the main ones are:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq M[2] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP$$

The principal analogue of the classical intractability class NP is $W[1]$. This is because a fundamental problem complete for $W[1]$ is the k-Step Halting Problem for Nondeterministic Turing Machines (with unlimited nondeterminism and alphabet size) — this completeness result provides an analogue of Cook's Theorem in classical complexity. A convenient source of $W[1]$-hardness reductions is provided by the result that k-Clique is complete for $W[1]$. Other highlights of the theory include that k-Dominating Set, by contrast, is complete for $W[2]$. XP is the class of all problems that are solvable in time $O(n^{g(k)})$. For a detailed introduction to the parameterized complexity approach, we point the reader to the books [Nie06, DF99, FG06], and the survey [HNW08].

3 Polynomial Kernels

Kernelization is preprocessing formalized. In this section, we define the notion of kernelization on parameterized languages and give an example kernelization algorithm for vertex cover that results in a polynomial sized kernel.

Kernels and Parameterized Tractability. A kernelization algorithm for a parameterized problem (Q, κ) is a polynomial time procedure that converts an instance x into y, such that $|y| = f(k)$ and $x \in Q$ if and only if $y \in Q$ where f is some computable function. A kernelization procedure gives us a reduced instance whose size can be upper bounded by a function of k alone, and this is usually called the kernel of the problem.

Definition 3 ([FG06])**.** Let (Q, κ) be a parameterized problem over $\{0,1\}^*$. A polynomial time computable function $K : \{0,1\}^* \to \{0,1\}^*$ is a kernelization of (Q, κ) if there is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \{0,1\}^*$ we have $(x \in Q \iff K(x) \in Q)$ and $|K(x)| \leq h(\kappa(x))$. If K is a kernelization of (Q, κ), then for every instance x of Q the image $K(x)$ is called the kernel of x under K.

Observe that a kernelization is a polynomial time many-one reduction of a problem to itself with the additional property that the image is bounded in terms of the parameter of the argument. In fact, fixed-parameter tractability and kernelization are identical notions. Although easy to see, the theorem is at the heart of our understanding of both kernelization and parameterized tractability. This theorem can be found in [FG06], [DF99] and [Nie06].

Theorem 1. *For every parameterized problem (Q, κ), the following are equivalent:*

(1) $(Q, \kappa) \in FPT$.
(2) Q is decidable, and (Q, κ) has a kernelization.

Having realized that FPT *is* the class of kernelizable problems, we also now have another way of getting deeper into this class using, for example, of the quality of kernels as a basis for classification.

A kernelization procedure is usually described as a collection of reduction rules, which are designed to transform the instances preserving equivalence. Each of these rules is applied to the instance recursively. Each application causes the instance to shrink in some way, and one hopes to be able to *prove* that if the rules have been applied until they are not applicable any more, then the resulting instance must be a kernel.

Naturally, we would like the kernel produced by the kernelization algorithm to be as "small" as possible. This motivates the notion of polynomial kernels, which we see next, and we also describe some rules to reduce instances of the Vertex Cover problem.

Polynomial Kernels. A polynomial kernel is a kernel whose size is a polynomial in the original parameter.

Definition 4. Let (Q, κ) be a parameterized problem over $\{0,1\}^*$. A polynomial time computable function $K : \{0,1\}^* \to \{0,1\}^*$ is a polynomial kernelization of (Q, κ) if there is a polynomial $h : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \{0,1\}^*$ we have $(x \in Q \iff K(x) \in Q)$ and $|K(x)| \leq h(\kappa(x))$. If h is a linear function then we say that (Q, κ) admits a linear kernel.

A Kernel for Vertex Cover For an example we give an informal description of a simple polynomial kernel (due to S. Buss, see [BG93]) for the vertex cover problem:

> Vertex Cover
> Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
> Question: Does G have a vertex cover of size k or less?

We describe some valid reduction rules for the Vertex Cover problem:

Rule 1: An isolated vertex u, a vertex of degree 0, cannot be in a vertex cover of optimal size. Since there are no edges associated with such a vertex, there is no benefit of including it in any vertex cover, so we may delete all the isolated vertices from a graph.

Rule 2: If u is a pendant vertex, a vertex of degree 1, there is a vertex cover of optimal size that does not contain the pendant vertex but does contain its unique neighbor v. Thus, we may delete both u and v and the edges incident on v, and decrease the parameter by 1.

Rule 3: Suppose the degree of a vertex u is more than k. Observe that u is forced to be a part of any vertex cover of size at most k. Any k-sized vertex cover cannot afford to exclude u, because if u does not belong to the vertex cover, then to cover the edges incident on u, all its neighbors must belong to the vertex cover, but this would imply a vertex cover of size strictly more than k. Thus, we may delete any vertex u such that $d(u) > k$, and for every such u, we decrease the parameter by 1.

Rule 4: Note that in any vertex cover in the resulting graph, each vertex may cover at most k edges since Rule 3 has been applied on all vertices of the graph. Thus, if the number of edges in the graph is in excess of k^2, we may safely rule out the possibility that the graph admits a vertex cover of size at most k. Therefore if the graph has more than k^2 edges after the application of Rule 3 then we terminate and say NO.

We repeat each of these rules in the sequence until they are no longer applicable. When we stop we are left with a graph where the degree of every vertex in the graph is at most k and the number of edges is at most k^2. We refer to such a graph as a *reduced* instance.

The number of edges on any reduced graph that is a yes-instance is at most k^2, because of Rule 4. Thus, we have a kernel as a bound on the number of edges, and this also implies that the number of vertices is bounded — it is useful, in this context, to know that there are no isolated vertices.

Remark: Notice that if the problem is to find a vertex cover whose vertices induce a *connected* graph, then the reduction rules described above don't apply. In particular, vertices with degree more than k cannot be deleted from the graph as we need to "remember" what vertices need to be connected. It is still true that we may have at most k vertices of degree more than k in any YES instance of the problem. Further, the graph induced on vertices of degree at most k will have at most k^2 edges in an YES instance. However, notice that the set L of vertices with degree at most k is no longer bounded by a function of k, as there could be a large number of vertices that are isolated in the graph induced on L, but may be adjacent to vertices of degree more than k, which we are unable to delete.

We shall see in section 6.4.2 that for the problem of finding a connected vertex cover, it is not just that these specific rules do not apply, but that it is infeasible, under certain complexity-theoretic assumptions, to devise *any set of* rules that leave us with a kernel of polynomial size — even though the problem appears to be a simple variant of vertex cover, which admits a simple quadratic kernel.

There are many examples of kernels that are obtained using clever reduction rules, the vertex cover problem alone is known to admit a handful of different kinds of kernelizations. For example, it has a linear vertex-kernel via either the use of linear programming in which case the problem kernel has $2k$ vertices, or the so-called *crown reduction* rules using which the kernel has at most $3k$ vertices. Details for both may be found in the book, [Nie06]. For work on a more general variant of Vertex Cover, see [CC08]. The Feedback Vertex Set problem for which we are required to check if there exist k vertices whose removal makes the graph acyclic, for instance, has a quadratic kernel [Tho09]. For the problem of editing at most k edges, that is, adding non-existent edges and deleting given edges such that the resultant graph is a disjoint union of cliques — called Cluster Editing, a quadratic kernel is known. For an example of work on Cluster Editing, see [GGHN03]. Many of these problems are known to have numerous applications. Refer to[GN07] for a survey on problems having polynomial sized kernels.

4 No Polynomial Kernels

We are now ready to examine the circumstances under which we do not expect a polynomial kernel for parameterized problem. We spend this section describing a characterization of problems that do not have polynomial-sized kernels[1] using a certain property. This turns out to be useful — showing that a problem is unlikely to admit a polynomial kernel boils down to demonstrating this property for the problem. The results in this section are based on seminal works by Bodlaender et al [BDFH09] and Fortnow and Santhanam [FS08].

4.1 Distillation Algorithms

Fortnow and Santhanam [FS08] first showed the infeasibility of compressing what they called OR-SAT. Let $\phi_i, 1 \leq i \leq t$, be instances of SAT, which is the standard language of propositional boolean formulas. OR-SAT is the following language:

[1] Under reasonable complexity theoretic assumptions, specifically under the assumption that $coNP \subseteq NP/poly$.

$$OR\text{-}SAT = \{< \phi_i, 1^n > | 1 \leq i \leq t \text{ at least one of the } \phi_i \text{ is satisfiable, and each } \phi_i \text{ is of length at most } n, \text{ i.e., uses at most } n \text{ variables.}\}$$

Any compression routine for *OR-SAT* can be thought of as an algorithm that accepts multiple instances of SAT and returns a small formula equivalent to the "or" of the input formulas. This motivates our first definition (see [BDFH09, FS08]).

A distillation algorithm for a given problem is designed to act as a "Boolean OR of problem-instances" — it receives as input a sequence of instances, and produces a yes-instance if and only if at least one of the instances in the sequences is also a yes-instance. Although the algorithm is allowed to run in time polynomial in the total length of the sequence, its output is required to be an instance whose size is polynomially bounded by the size of the maximum-size instance in its input sequence. Formally, we have the following:

Definition 5. Let $Q, Q' \subseteq \{0,1\}^*$ be classical problems. A *distillation from Q in Q'* is a polynomial time algorithm $\mathcal{D}$ that receives as inputs finite sequences $\overline{x} = (x_1, \ldots, x_t)$ with $x_i \in \{0,1\}^*$ for $i \in [t]$ and outputs a string $\mathcal{D}(\overline{x}) \in \{0,1\}^*$ such that

1. $|\mathcal{D}(\overline{x})| = (max_{i \in [t]} |x_i|)^{O(1)}$
2. $\mathcal{D}(\overline{x}) \in Q'$ if and only if for some $i \in [t] : x_i \in Q$.

If $Q' = Q$ we speak of a self-distillation. We say that Q has a distillation if there is a distillation from Q in Q' for some Q'. One of the most important theorems on which this entire lower bound theory was built is the following.

Theorem 2 ([FS08])**.** *If any NP-complete problem has a distillation algorithm then $coNP \subseteq NP/poly$.*

Let Q be an NP-complete problem with a distillation algorithm $\mathbb{A}$, and let $\overline{Q}$ denote the complement of Q. The authors show that using $\mathbb{A}$, one can design a non-deterministic Turing machine (NDTM) that, with the help of a polynomial advice, can decide Q in polynomial-time. This will show that $coNP \subseteq NP/poly$ prove the statement of the theorem. Note that when combined with Yap's theorem [Yap83] ($coNP \subseteq NP/poly \Rightarrow PH \subseteq \Sigma_3^p$), this will also imply that a distillation algorithm for a NP-complete problem implies $PH \subseteq \Sigma_3^p$. (For a detailed proof, see [FS08]).

4.2 Ruling Out Polynomial Kernels

We turn our attention to distillation-type algorithms for parameterized problems. These algorithms are similar to their classical counterparts in spirit, except that they must specify additional constraints on the size of the parameter of the output instance. Different choices at this stage lead to different definitions and implications. For instance, consider algorithms that require all the parameters in the sequence to be the same, and have the parameter of the output polynomially-bounded in k, and have no requirement on the length of the output instance. Such an algorithm is called a *composition*.

Of interest is the fact that the existence of a composition algorithm for a parameterized problem along with a polynomial kernel implies a distillation algorithm for the corresponding classical problem. The formal definition of a composition algorithm is the following:

Definition 6. [BDFH09] Let (P, κ) be a parameterized problem. A *composition of* P is a polynomial time algorithm $\mathbb{A}$ that receives as inputs finite sequences $\overline{x} = (x_1, \ldots, x_t)$ with $x_i \in \{0,1\}^*$ for $i \in [t]$, such that $\kappa(x_1) = \kappa(x_2) = \cdots = \kappa(x_t)$. Let $k := \kappa(x_1)$. The algorithm is required to output a string $\mathbb{A}(\overline{x}) \in \{0,1\}^*$ such that

1. $\kappa(\mathbb{A}(\overline{x})) = k^{O(1)}$
2. $\mathbb{A}(\overline{x}) \in P$ if and only if for some $i \in [t] : x_i \in P$.

It turns out that if the parameterized version of a NP-complete problem admits both a composition and a polynomial kernel, then it also has a distillation. This will imply that if a parameterized problem has a composition algorithm, then it has no polynomial kernel unless $coNP \subseteq NP/poly$, due to Theorem 2.

Theorem 3 ([BDFH09])**.** *Let (P, κ) be a compositional parameterized problem such that P is NP-complete. If P has a polynomial kernel, then P also has a distillation algorithm.*

5 Examples of Compositions

In this section we prove the conditional hardness of obtaining polynomial kernels for a variety of problems, and suggest a standardized framework in which these algorithms may be understood.

Algorithms that compose multiple instances of a problem into one (respecting a number of properties) have been developed for various problems ([BDFH09],[CFM09], [DLS09]).

5.1 Composition by Disjoint Union

We begin with a warm-up example; consider the following problem:

k-PATH	
Instance:	A graph G and a non-negative integer k.
Parameter:	k.
Question:	Does G have a path of length k?

This problem is shown to be in FPT with running time $2^{O(k)}n^{O(1)}$ via the technique of color coding. The algorithm randomly "colors" the vertex set with k colors, that is, it picks uniformly at random a function $c : V \to [k]$ and hopes that the vertices of the k-path we are after are colored distinctly. That is, the function restricted to at least one witness k-length path is injective. It then uses dynamic programming to actually identify the colorful path from the colored graph. The algorithm is de-randomized using a family of hash functions, and the reader is referred to [AYZ95] for more details.

We present here a composition for k-PATH. Suppose the input instances are:

$$(G_1, k), \ldots, (G_t, k)$$

A composition algorithm is trivial, it simply provides the disjoint union G' of all the graphs as the output. Notice that this consumes linear time, and leaves the parameter unchanged.

Clearly, G' has a path of length at most k if, and only if, there exists i, $1 \leq i \leq t$, such that G_i has a path of length at most k.

Informally speaking, the disjoint union strategy works whenever we look for connected structures that are required to satisfy properties that are functions of the vertices participating in the structure (and are independent of rest of the graph). Typically, the fact that the property is "local" helps the forward direction and the connectivity aids the reverse direction of the argument. Examples include k-Cycle, k-Tree or k-Out Tree in a directed graph. Thus these problems, though FPT, do not admit polynomial kernels unless $coNP \subseteq NP/poly$.

However, most composition algorithms are more complicated than this, though they seem to follow some general strategies. We begin by describing some of the strategies that have been designed and end with one, which we feel, is general enough to capture most of these.

5.2 Composition Using IDs

In general, a compositional algorithm $\mathcal{A}$ for a parameterized language (Q, κ) is required to "merge" instances $x_1, x_2, \ldots, x_t$, into a single instance x in polynomial time (more precisely, if $n_i := |x_i|$, and $n = \sum n_i$, then the algorithm's runtime is a polynomial in n). The output of the algorithm belongs to $(Q, \kappa(x))$ if and only if there exists at least one $i \in [t]$ for which $x_i \in (Q, \kappa)$. Further, the length of the output of a *composition algorithm* is unrestrained, short of having to be $n^{O(1)}$ which follows from the constraint of the algorithm to spend only $n^{O(1)}$ time.

A compositional algorithm is also equipped with the promise that $\kappa(x_i) = k$, for all $i \in [t]$. Recall that we require the parameter of the output to be a polynomial in the parameter of the inputs.

We now describe some general observations that can be applied to specific problems during the design of such algorithms. We will use $\mathcal{A}$ to refer to a composition algorithm.

The first observation is inspired by the fact that the time available to $\mathcal{A}$ for working things out is more when the number of instances fed to it is larger. We also know that a given instance can be solved completely in FPT time, so we attempt to exploit this. Suppose the membership query "$x \in (Q, \kappa)$?" can be solved in time

$$T_s = 2^{\kappa(x)^c} \cdot |x|^{O(1)}.$$

Notice that the time bound for $\mathcal{A}$ is

$$T_a = n^{O(1)} = (n_1 + \cdots + n_t)^{O(1)}.$$

Let k denote the largest parameter in the input. Observe that if $t > 2^{k^c}$, then we can afford to invest time T_s for each of the instances. Indeed, the total running time T satisfies:

$$\begin{aligned} T &\leq 2^{k^c}(n_1^{O(1)} + \cdots + n_t^{O(1)}) \\ &\leq t \cdot (n_1 + \cdots + n_t)^{O(1)} \\ &\leq t n^{O(1)}, \end{aligned}$$

which is an admissible running time for the composition algorithm. Let us take advantage of this: whenever the number of instances is larger than 2^{k^c}, and a FPT algorithm using time $2^{k^c} \cdot n^{O(1)}$

is known, we iterate through the instances, solving each one completely. If we encounter a YES-instance on the way, we stop and return that instance as the output, otherwise, (all the instances are NO-instances) we choose to return any one of them. It is easy to verify that this is a composition.

The objective of the exercise is to ensure that we may always be working, without loss of generality, with a bounded number of instances in the input to the compositional algorithm. The longer the time taken by the FPT algorithm used in the argument above, the weaker the bound. Our interests will be restricted to bounds of the form $2^{k^{O(1)}}$, because of reasons that will become clear below. In particular, all the problems we discuss in this section admit such FPT algorithms.

At this point, we recall that the parameter of the output is required to be a polynomial in the parameter of the input instances (or the maximum of the parameters in the input instances, in case they are different). This requirement can now be rephrased in a manner that will be useful to us. Note that

$$t \leq 2^{k^c} \Rightarrow k^c \geq \log t,$$

allowing us to say that $\kappa(x)$, the magnitude of the parameter of the output, is allowed to be $(\log t)^{O(1)}$.

It turns out that this is a good way of putting it, because the "$\log(t)$" allowance for new parameter generally is just enough to encode information about the individual instances in the composed instance. Usually, it turns out that $\log(t)$ objects that contribute to the parameter can make their presence felt in $2^{\log(t)}$ different ways in the composed instance, and since this is exactly equal to the number of input instances, we find them useful in finding traces of the original instances in the composed instance. This emerges as a concrete strategy when we work out specific examples.

We illustrate the use of this intuition with the example of the problem of finding a minimum weight satisfying assignment for a boolean formula. Given a boolean formula F in CNF format with at most c variables in each clause, consider the problem of finding a satisfying assignment to the formula with the minimum weight (number of variables set to 1). A natural parameterized version asks whether there is a satisfying assignment with weight at most k for a given integer parameter k. This problem is a generalization of c-hitting set (which is basically the VERTEX COVER problem when $c = 2$) and has a simple $O(c^k|F|^{O(1)})$ branching algorithm [Nie06]. For $c = 2$, it has a $2k$ variables kernel as in the case of vertex cover, and it is known that there is no polynomial sized kernel when $c > 3$ unless $coNP \subseteq NP/poly$ (see [KW]). Here, we give a simple composition for the problem when parameterized by $k + c$ where c is the maximum length of a clause. This is based on a composition described in [CFM09].

Let $F_1, F_2, \ldots F_t$ be the t instances of the problem with c being the maximum length of the formula in each of the instances and k is the (upper bound for the) weight of the assignment sought for. As described in the previous section, we can first assume that $t = c^k$ as there is a $O^*(c^k)$ algorithm for the problem. We assume (by renaming if necessary) that the variables occurring in F_i and F_j are disjoint if $i \neq j$. The composed instance will involve $2b$ new variables disjoint from the ones that already occur in any of the formulae. Denote these new variables by $y_1, y_2, \ldots y_b, z_1, z_2, \ldots z_b$, where $b = k\lceil \lg c \rceil$.

We now wish to associate with each instance, a unique identifier devised using the new variables. This is achieved by using the variables as a representation of the binary encoding (which consumes b bits) of the index of the instance (which ranges from 1 to t). More precisely, let

$d_1 d_2 \ldots d_c$ be the binary representation of d, $1 \leq d \leq t$. Then, consider the clause C_d formed by the disjunction of the following literals:

$$\{y_i \mid d_i = 1\} \cup \{\bar{y}_j \mid d_j = 0\}$$

Notice that for distinct d, the corresponding clauses C_d are distinct. Now, consider the formula F'_d obtained by replacing every clause C in F_d by $C \vee C_d$ (that is, the set of disjuncts present in C is expanded to include the disjuncts in C_d. The composed formula F is the conjunction of F'_d, for all $1 \leq d \leq t$, with the following additional clauses:

$$\wedge_{i=1}^{b}(y_i \vee z_i)(\bar{y}_i \vee \bar{z}_i).$$

In the composed formula F, the maximum length of the clause has increased by $b = k\lceil \lg c \rceil$. The additional set of clauses introduced above ensures that in any satisfying assignment of F, the weight of the y and z variables is exactly b.

Now we claim that F has a satisfying assignment of weight at most $k + b$ if and only if some F_i has a satisfying assignment of weight at most k. Suppose F_j has a satisfying assignment of weight at most k. Set the y variables in such a way that the disjunction of y variables corresponding to the index j evaluate to 0. This ensures that every F'_l for $l \neq j$ is satisfied by the y variables, and F'_j is satisfied by the original (weight at most k) satisfying assignment of F_j, resulting in a satisfying assignment of weight at most $k + b$ for F. Conversely if F has a satisfying assignment of weight at most $k + b$, as discussed the y and z variables consume a weight of k. Let y_r be the bit string obtained by concatenation of the compliments of the y variables in the satisfying assignment, and let j be the integer represented by the binary coding y_r. Then it is easy to see that F_j has a satisfying assignment of weight at most k.

5.3 Composition with Colors and IDs

We introduced, in the previous section, the notion of attaching identifiers to the problem instances to achieve composition. In some situations, this alone is not enough to engineer the entire composition algorithm. A finer distinction within each problem was introduced in [DLS09], where a *colored* version of the problem was considered. For subset problems on graphs, where one is looking for a subset of size at most k with certain properties, the colored version is generally the following: the input graph is equipped with a coloring function from the vertex set to a set of k colors, and the solution is required to be such that the coloring function is bijective when restricted to the solution. I.e. we would like the elements of the solution subset to have distinct colors. The colors turn out to be an useful aid in composition. Also, this usually leads to a hardness result for the original problem as well, using the mechanism of reductions, covered in section 6. We illustrate this situation using the example of the Red-Blue Dominating Set problem, which is the following:

Red-Blue Dominating Set (RBDS)	
Instance:	A bipartite graph $G = (T \cup N, E)$ and a non–negative integer k
Parameter:	$k + \lvert T \rvert$.
Question:	Does there exist a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has at least one neighbor in N'?

An instance of RBDS comprises of a bipartite graph $G = (T \cup N, E)$ and an integer k. We ask whether there exists a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has at least one neighbor in N'. The problem is parameterized by $k + |T|$.

In the literature, RBDS is usually known under the name "Red-Blue Dominating Set"and the sets T and N are called "blue vertices" and "red vertices", respectively. Here, we call the vertices "terminals" and "nonterminals" in order to avoid confusion with the colored version of the problem that we are going to introduce. RBDS is equivalent to Set Cover and Hitting Set and is, therefore, NP-complete [GJ90]. Our interest is in showing the hardness of polynomial kernelization for this problem — however, we do not know of a composition for it so far. We will eventually provide a way reducing to this problem from a closely related variant. The reduction is again deferred to the next section, we presently describe the variant which is shown to be compositional in [DLS09].

Consider the colored version of RBDS, denoted by Colored Red-Blue Dominating Set (Col-RBDS):

Colored Red-Blue Dominating Set (Col-RBDS)	
Instance:	A bipartite graph $G = (T \cup N, E)$ and a non–negative integer k and a function $col : N \to \{1, \ldots, k\}$.
Parameter:	$k + \|T\|$.
Question:	Does there exist a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has at least one neighbor in N' and N' has exactly one vertex of each color?

Here, the vertices of N are colored with colors chosen from $\{1, \ldots, k\}$, that is, we are additionally given a function $col : N \to \{1, \ldots, k\}$, and N' is required to contain exactly one vertex of each color. The parameter is again $k + |T|$.

The first step in showing composition for Col-RBDS is to assume that the number of problem instances of Col-RBDS in the input, a tuple of instances of Col-RBDS, to the composition algorithm is small using the trick described in Section 5.2. After this the composition for Col-RBDS described in [DLS09] assigns unique identification gadget to each of these instances. This allows them to identify the yes instance when showing that if the output of the composition algorithm is an yes instance then there exists an instance among the input tuple that is an yes instance. For a detailed description, and other examples of this strategy we refer to [DLS09].

5.4 Composition as Dynamic Programming

We described compositional algorithms for various problems. It turns out that most of these algorithms can be framed in a dynamic programming setting. We will denote the "update" of the dynamic programming by $\rho : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, which accepts two instances and returns one. We use ρ in a systematic way over the inputs $(x_1 \ldots x_t)$. It will generally be useful to have in mind the picture of a complete binary tree on t leaves.

The "table" of the dynamic programming is most simply described as a tree. We visualize the input instances as plugged in at the leaves of a complete binary tree with t vertices. For convenience, we generally assume $t = 2^l$ $(l \leq k^c)$. This makes the tree l levels deep. (Such an assumption can usually be easily justified.) We inductively compute the contents of a given

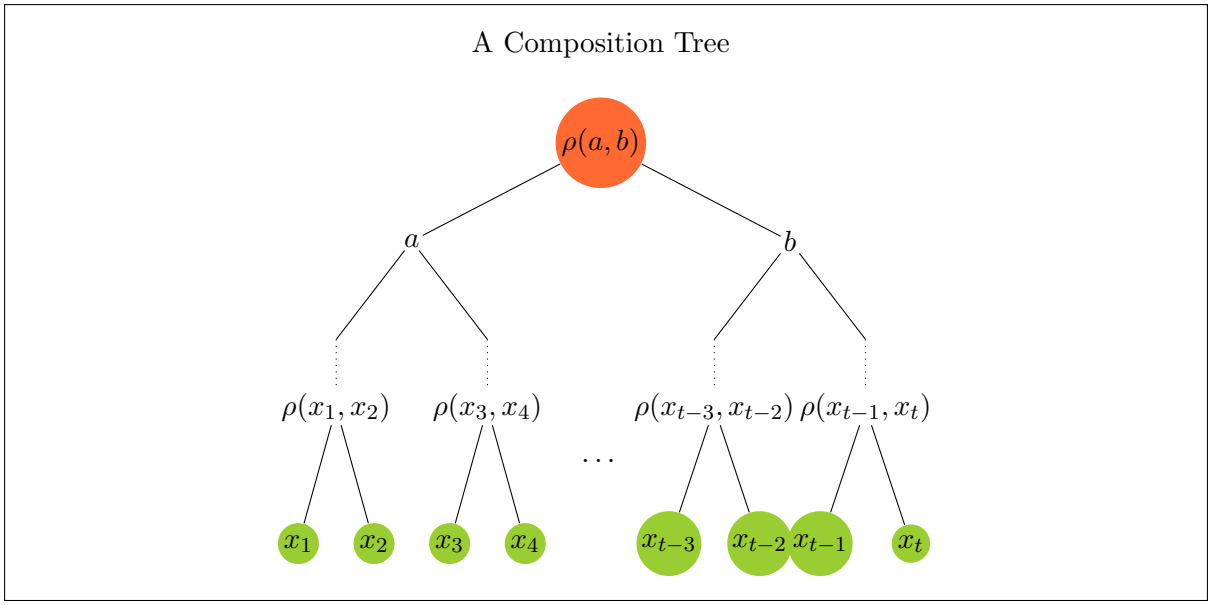


Figure 1: General Framework For Composition

node as being $\rho(a,b)$, where a and b are the instances obtained at the children of this node. The output of the composed algorithm is what is obtained at the root of this tree (as shown in the Figure 1).

In the case of the k-path problem, we can think of the 'update' function as just performing a disjoint union of the two instances it receives. Those strategies that assign a unique identifier to each instance can also be visualized in this setting by suitably 'distributing' the identifiers to the 'update' function, so that the id of an instance can be recovered by looking at the path between the root and the leaf corresponding to the instance.

We explain this framework in detail by describing the composition for the Disjoint Factors problem [BTY09].

Let $[k]$ be the set consisting of the letters $\{1,2,\ldots,k\}$. We denote by $[k]^*$ the set of words on $[k]$. A factor of a word $w_1 \cdots w_r \in [k]^*$ is a substring $w_i \cdots w_j \in [k]^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter, i.e., the factor has length at least two and $w_i = w_j$. A word w has the *disjoint factor property* if one can find disjoint factors $F_1,\ldots,F_k$ in w such that the factor F_i starts and ends by the letter i.

For example, the word 123235443513 has all the r-factors, $r \in [5]$ — but not as many disjoint factors. It has disjoint 2, 3, and 4 factors, but, for instance, the only 5-factor overlaps with with the 4-factor, and the only 1–factor overlaps with all other factors. Of course, other combinations of disjoint factors are attainable from this word, but it clearly does not have the disjoint factor property. The problem of testing whether a given string has the Disjoint Factors property is known to be NP-complete [BTY09].

Observe that the difficulty lies in the fact that the factors F_i, if they occur, need not appear in increasing order, otherwise one can detect them in $O(n)$ time, where n is the length of w.

We now introduce the parameterized problem Disjoint Factors.

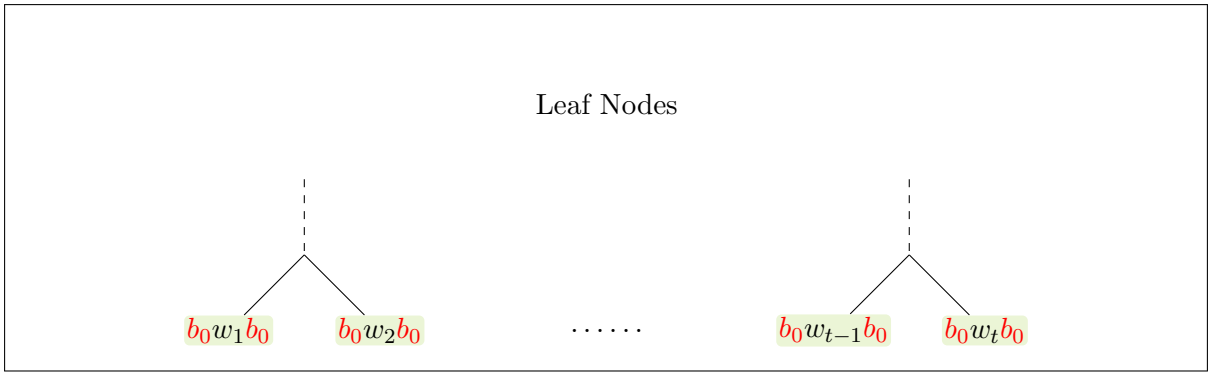


Figure 2: Composition for Disjoint Factors: Leaves

Disjoint Factors	
Instance:	A word $w \in L_k^*$.
Parameter:	$k \geq 1$.
Question:	Does w have the Disjoint Factors property?

It is immediate that p-Disjoint Factors is FPT. Since the problem can be solved in time that is linear in the length of the word given the ordering of the factors, we simply iterate over all possible orderings of factors — this gives us an algorithm with runtime $k! \cdot n$. This complexity can be improved to $2^k \cdot p(n)$ using Dynamic Programming, see [BTY09].

Let $w_1, w_2, \ldots, w_t$ be words in $[k]^*$. Towards the composition algorithm $\mathcal{A}$, we may begin by assuming that the number of instances is bounded by 2^k, since when we have more, we may use the FPT algorithm to solve each instance separately. As a matter of convention, we output the first w for which $\mathcal{A}$ succeeds in finding k disjoint factors, and in the event that none of the words have k disjoint factors, $\mathcal{A}$ returns w_t. Clearly, $\mathcal{A}$ is a composition.

When $t < 2^k$, we make the reasonable assumption that $t = 2^l$ for some l. Notice that $l \leq k$, and therefore, we observe that we may use at most l^d new letters in the composed instance, for some constant d. It turns out that we will need exactly l new letters, $b_1, \ldots, b_l$. We begin by plugging in the words at the leaves along with b_0 appended to either end:

$$\rho(\mathbb{T}_0(i)) = b_0 w_i b_0.$$

Let $\lambda(xa, by)$, where a,b are letters, and x,y are words, denote the word $xaby$ when $a \neq b$ and the word xay when $a = b$. Then we have, at the j^{th} level; $1 \leq j < l$:

$$\rho(\mathbb{T}_j(i)) = b_j \;\; \lambda(\rho(\mathbb{T}_{j-1}(2i-1)), \rho(\mathbb{T}_{j-1}(2i))) \;\; b_j.$$

At the root, we perform a simple concatenation:

$$\mathbb{T}_l(1) = \lambda(\mathbb{T}_{l-1}(1), \mathbb{T}_{l-1}(2)) = w$$

Evidently, if $\mathbb{T}_l(1))$ has the disjoint factors property, then we are forced to pull out the $1, 2, \ldots, k$ disjoint factors from exactly one of the original words.

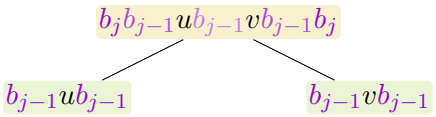
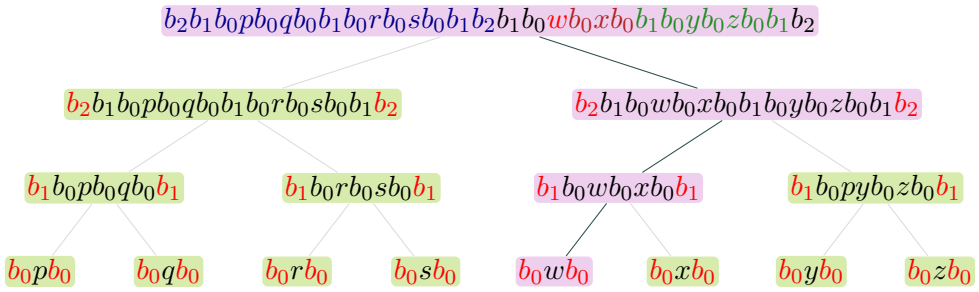


Figure 3: Disjoint Factors: ρ



An example with

$$\{p, q, r, s, w, x, y, z\}$$

as input words. Notice how a solution in the root can be traced to some unique solution in the leaves and conversely! (Warning: beware of confusing this w with the w we have been using to denote the composed instance — they are different.)

Figure 4: Disjoint Factors Composition: An Example

We argue this in the forward direction first: suppose one of the w_i's has the disjoint factors property. We want to show that the composed instance also has the disjoint factors property (over the extended alphabet $[k] \cup \{b_0, \ldots b_{l-1}\}$, observe that:

The $1, 2, \ldots, k$ disjoint factors may be obtained from w_i.

Note that w_i is a substring of one of the b_{l-1} factors. For b_{l-1}, therefore, we choose that b_{l-1} factor that does not overlap with w_i.

Further, there are exactly three b_{l-2} factors that do not overlap with w_i, and two of these overlap with the chosen b_{l-1} factor, so there is one b_{l-2} factor that does not overlap with any of the factors chosen thus far, and we use this as the b_{l-2} factor.

This process can be continued until all the b_i factors are obtained, indeed, we will always have one for every $0 \leq i \leq l-1$ by construction.

On the other hand, if the composed instance has the disjoint factors property, then we would like to derive that all the $1, 2, \ldots, k$ factors are substrings of w_i for some i. It is easy to observe that if we delete all the b_i factors from the word w, then the remaining word contains exactly one of the w_i's as a substring. For example, notice that the b_{l-1} factor overlaps with half of the w_i's, and the b_{l-2} factor overlaps with half of the remaining w_i's, and so on. Thus, once the b_i factors have been accounted for, only one possible source remains for the $1, 2, \ldots k$ factors — necessarily one of the w_i's. But then w_i must exhibit (by definition) the disjoint factors property over the restricted alphabet $[k]$, and hence we are done.

Note that the parameter of the composed instance is $k + l$, where $l \leq k$. Thus the size of the alphabet of the composed instance is at most twice the size of the original alphabet. It is easy to see that the algorithm runs in polynomial time, as the operations at every node would consume only constant time.

5.5 Summary

In this section, we have shown composition algorithms for a number of problems. By theorem 2 and 3, we now have the following:

Theorem 4. *The following problems admit composition algorithms, and hence do not admit polynomial kernels unless coNP $\subseteq$ NP/poly: k-*PATH*, MinweightSAT (parameterized by the weight and the maximum length of a clause),* DISJOINT FACTORS*,* COL-RBDS*.*

6 Transformations

In this section, we introduce the notion of transformations, which will allow us to prove results for problems that do not obviously have compositions.

6.1 Philosophy and Definition

We begin by describing what we mean by a polynomial parameter transformation [BTY09, DLS09].

Definition 7 (Polynomial parameter transformation)**.** Let (P, κ) and (Q, γ) be parameterized problems. We say that P is polynomial parameter reducible to Q, written $P \preccurlyeq_{ppt} Q$, if there exists a polynomial time computable function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$, and a polynomial $p : \mathbb{N} \to \mathbb{N}$, and for all $x \in \{0,1\}^*$ and $k \in \mathbb{N}$ if $f(x) = y$, then the following hold:

1. $x \in P$, if and only if $y \in Q$, and
2. $\gamma(y) \leq p(\kappa(x))$

We call f a polynomial parameter transformation from P to Q.

Notice the differences with the notion of fixed-parameter reductions introduced by Downey and Fellows (see [DF95b, DF95c, DF95a]). In general fixed-parameter reductions, one is allowed $f(k)p(|x|)$ time for an input instance (x, k) where f is an arbitrary function and p is a polynomial function, and the resulting parameter is allowed to be an arbitrary function of the original parameter. Here the running time allowed is only a polynomial in $|x|$ and k and the resulting parameter value is only a polynomial function of the original parameter.

Polynomial parameter transformations are used to show non-existence of polynomial sized kernels using the following theorem.

Theorem 5 ([BTY09])**.** *Let (A, κ) and (B, γ) be parameterized problems such that A is NP-complete, and $B \in NP$. Suppose that there is a polynomial parameter transformation from A to B. Then, if B has a polynomial kernel, then A has a polynomial kernel.*

As an easy corollary of Theorem 5, note that whenever (A, κ) and (B, γ) are parameterized problems (such that A is NP-complete, and $B \in NP$) and $A \preccurlyeq_{ppt} B$, if A is compositional, then B does not have a polynomial kernel unless $coNP \subseteq NP/poly$. A natural strategy to prove that a problem A is unlikely to admit a polynomial kernel, is to reduce some NP-complete problem B, for which we have a composition, to A using a polynomial parameter transformation.

6.2 (Vertex) Disjoint Cycles

Consider the following two parameterized problems.

Vertex Disjoint Cycles	
Instance:	Undirected graph $G = (V, E)$ and a non-negative integer k.
Parameter:	k.
Question:	Does G contain at least k vertex-disjoint cycles?

Edge Disjoint Cycles	
Instance:	Undirected graph $G = (V, E)$ and a non-negative integer k.
Parameter:	k.
Question:	Does G contain at least k edge-disjoint cycles?

The problem of Vertex Disjoint Cycles is strongly related to the Feedback Vertex Set (FVS) problem, wherein the question is whether there exist k vertices whose deletion makes the graph acyclic (usually studied with k as the parameter). Clearly, if a graph has more than

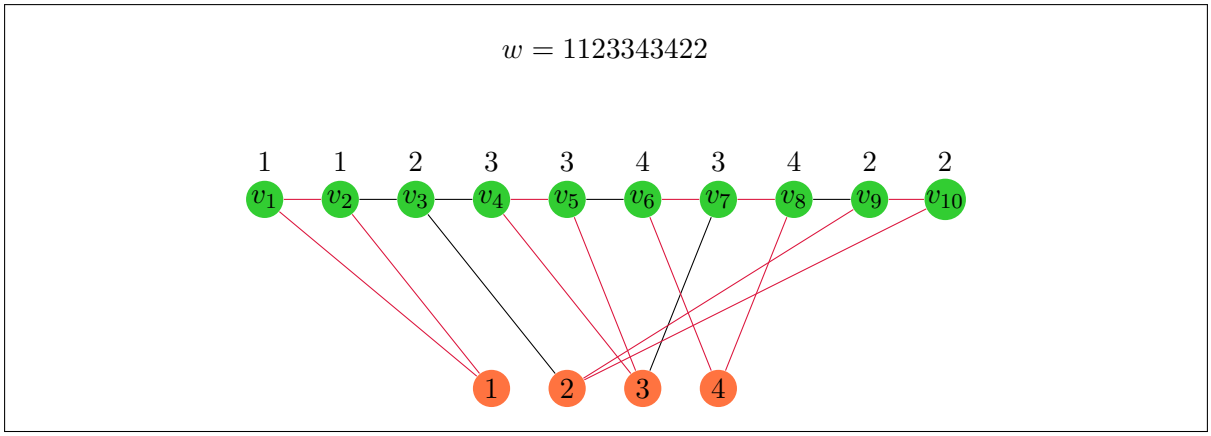


Figure 5: Disjoint Factors $\preccurlyeq_{ppt}$ Disjoint Cycles

k vertex disjoint cycles, then it cannot have a FVS of size k or less, as any FVS has to pick at least one vertex from every cycle. If there are at most k vertex disjoint cycles, the implications are less immediate, but an upper bound of $O(k \log k)$ on the size of the optimal FVS is known, due to a result by Erdős and Pósa [EP65]. For the FVS problem, there is a kernel of size $O(k^2)$ ([Tho09]) by Thomassé, who improved upon a kernel of size $O(k^3)$ ([Bod07]). The Edge Disjoint Cycles problem has a polynomial kernel (see [BTY09]).

In contrast, it is shown in [BTY09] that the Vertex Disjoint Cycles problem does not admit a polynomial kernel through a polynomial parameter transformation from Disjoint Factors. From now on, when we say disjoint cycles we mean *vertex* disjoint cycles.

Recall that the Disjoint Factors problem was the following:

Disjoint Factors	
Instance:	A word $w \in L_k^*$.
Parameter:	$k \geq 1$.
Question:	Does w have the Disjoint Factors property?

Given an input (W, k) of Disjoint Factors, with $W = w_1 \cdots w_n$, a word in $\{0,1\}^*$, we build a graph $G = (V, E)$ as follows. First, we take n vertices $v_1, \ldots, v_n$, and edges $\{v_i, v_{i+1}\}$ for $1 \leq i < n$, i.e., these vertices form a path of length n. Let P denote this subgraph of G. Then, for each $i \in L_k$, we add a vertex x_i, and make x_i incident to each vertex v_j with $w_j = i$, i.e., to each vertex representing the letter i. See Figure 5 for an illustration.

We next claim that G has k disjoint cycles if and only if (W, k) has the requested k disjoint factors. Suppose G has k disjoint cycles $c_1, \ldots, c_k$. As P is a path, each of these cycles must contain at least one vertex not on P, i.e., of the form x_j, and hence each of these cycles contains exactly one vertex x_j (as the cycles are vertex disjoint, and there are k vertices available outside P). For $1 \leq j \leq k$, the cycle c_j thus consists of x_j and a subpath of P. This subpath must start and end with a vertex incident to x_j. These both represent letters in W equal to j. Let F_j be the factor of W corresponding to the vertices on P in c_j. Now, $F_1, \ldots, F_k$ are disjoint factors, each of length at least two (as the cycles have length at least three), and F_j starts and ends with j, for all j, $1 \leq j \leq k$.

Conversely, if we have disjoint factors $F_1, \ldots, F_k$ with the properties as in the Disjoint Factors problem, we build k vertex disjoint cycles as follows: for each $j, 1 \leq j \leq k$, take the cycle consisting of x_j and the vertices corresponding to factor F_j. Thus we have shown:

Theorem 6 ([BTY09]). DISJOINT FACTORS *does not admit a polynomial kernel unless* $coNP \subseteq NP/poly$.

6.3 Red-Blue Dominating Set

Recall that in the previous section, we showed a composition for the colored variant of RBDS. Our real interest, however, is in showing the hardness of obtaining polynomial kernels for RBDS. We complete that argument here, by reducing RBDS from COL-RBDS.

Recall that in RBDS we are given a bipartite graph $G = (T \cup N, E)$ and an integer k and asked whether there exists a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has at least one neighbor in N'. We also called the vertices "terminals" and "nonterminals" in order to avoid confusion with the colored version of the problem (COL-RBDS).

In the colored version that we showed was compositional, the vertices of N are colored with colors chosen from $\{1, \ldots, k\}$, that is, we are additionally given a function $col : N \to \{1, \ldots, k\}$, and N' is required to contain exactly one vertex of each color.

Theorem 7 ([DLS09]). (1) *The unparameterized version of* COL-RBDS *is* NP*-complete.*
(2) *There is a polynomial parameter transformation from* COL-RBDS *to* RBDS.
(3) COL-RBDS *is solvable in* $2^{|T|+k} \cdot |T \cup N|^{O(1)}$ *time.*

Proof. (1) It is easy to see that COL-RBDS is in NP. To prove its NP-hardness, we reduce the NP-complete problem RBDS to COL-RBDS: Given an instance $(G = (T \cup N, E), k)$ of RBDS, we construct an instance $(G' = (T \cup N', E'), k, col)$ of COL-RBDS where the vertex set N' consists of k copies $v^1, \ldots, v^k$ of every vertex $v \in V$, one copy of each color. That is, $N' = \bigcup_{a \in \{1,\ldots,k\}} \{v^a \mid v \in N\}$, and the color of every vertex $v^a \in N_a$ is $col(v^a) = a$. The edge set E' is given by

$$E' = \bigcup_{a \in \{1,\ldots,k\}} \{\{u, v^a\} \mid u \in T \wedge a \in \{1, \ldots, k\} \wedge \{u, v\} \in E\}.$$

The correctness of this construction is immediate.

(2) Given an instance $(G = (T \cup N, E), k, col)$ of COL-RBDS, we construct an instance $(G' = (T' \cup N, E'), k)$ of RBDS.

In G', the set T' consists of all vertices from T plus k additional vertices $z_1, \ldots, z_k$. The edge set E' consists of all edges from E plus the edges

$$\{\{z_a, v\} \mid a \in \{1, \ldots, k\} \ \wedge \ v \in N \ \wedge \ col(v) = a\}.$$

The proof of the correctness of this construction is immediate. See Figure 6 for an illustration.

(3) To solve COL-RBDS in the claimed running time, we first use the reduction given in (2) from COL-RBDS to RBDS. The number $|T'|$ of terminals in the constructed instance of RBDS is $|T| + k$. Next, we transform the RBDS instance (G', k) into an instance $(\mathcal{F}, U, k)$ of SET COVER where the elements in U one-to-one correspond to the vertices in T' and the sets in $\mathcal{F}$ one-to-one correspond to the vertices in N. Since SET COVER can be solved in $O(2^{|U|} \cdot |U| \cdot |\mathcal{F}|)$ time [FKWW04, Lemma 2], statement (3) follows. □

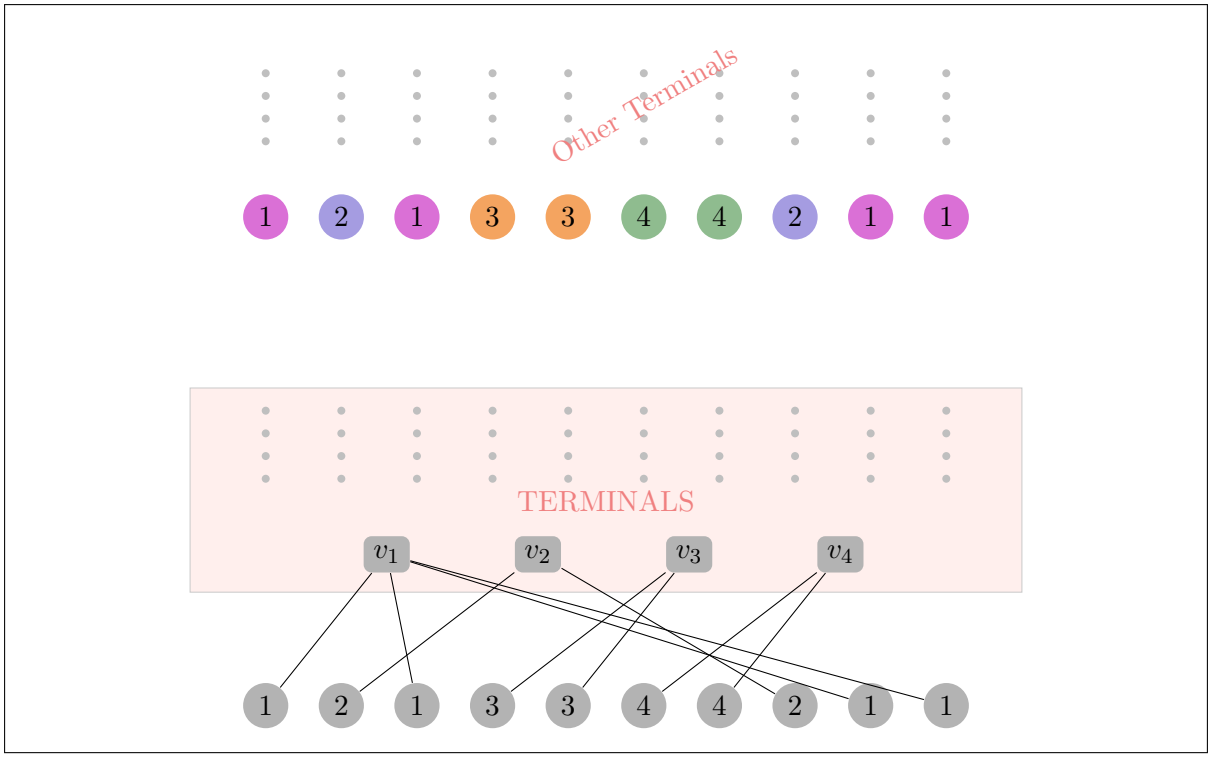


Figure 6: The polynomial parameter transformation from the colored version of RBDS to RBDS.

6.4 Reductions from RBDS

In this section, we use the fact that RBDS is compositional ([DLS09]) to give hardness results for four other problems, all of which are known to be NP-complete (see [GJ90]). The possibility of their admitting polynomial kernels is ruled out by polynomial parameter transformations from RBDS.

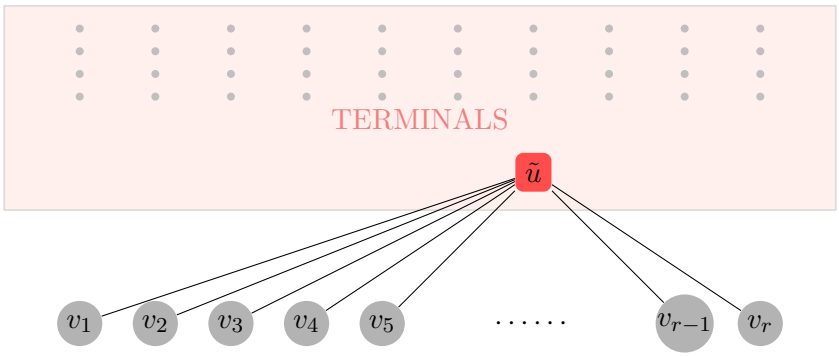
6.4.1 Steiner Tree

In Steiner Tree we are given a graph a graph $G = (T \cup N, E)$ and an integer k and asked for a vertex set $N' \subseteq N$ of size at most k such that $G[T \cup N']$ is connected. The problem is parameterized by $k + |T|$.

Let $(G = (T \cup N, E), k)$ be an instance of RBDS. To transform it into an instance $(G' = (T' \cup N, E'), k)$ of Steiner Tree, define $T' = T \cup \{\tilde{u}\}$ where $\tilde{u}$ is a new vertex and let $E' = E \cup \{\{\tilde{u}, v_i\} \mid v_i \in N\}$. See Figure 7 for an illustration. It is easy to see that every solution for Steiner Tree on (G', k) one-to-one corresponds to a solution for RBDS on (G, k).

6.4.2 Connected Vertex Cover

In ConVC we are given a graph $G = (V, E)$ and an integer k and asked for a vertex cover of size at most k that induces a connected subgraph in G. The parameter for the problem is the solution size, k.



This is how we modify an instance of RBDS to convert it into an instance of Steiner Tree. Notice that solution to the Steiner Tree problem on this modified problem must dominate all the non-terminals (if it does not, then $\tilde{u}$ is not connected to the vertex that is not dominated).

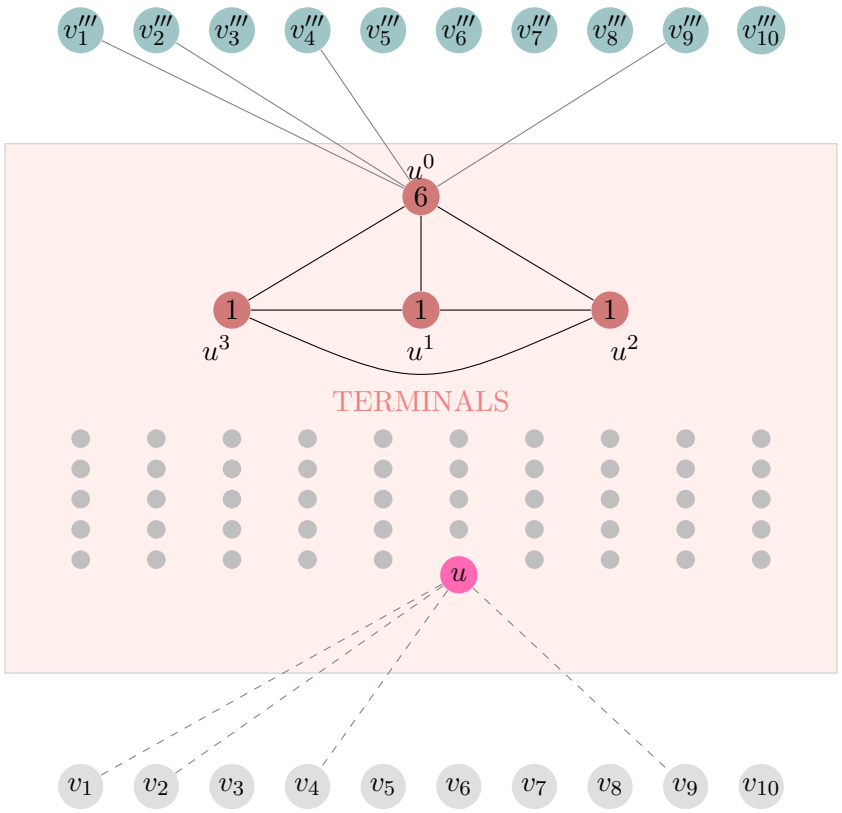
Figure 7: The polynomial parameter transformation from RBDS to Steiner Tree.

To transform (G, k) into an instance $(G'' = (V'', E''), k'')$ of ConVC, first construct the graph $G' = (T' \cup N, E')$ as described in section 6.4.1. The graph G'' is then obtained from G' by attaching a leaf to every vertex in T'. Now, G'' has a connected vertex cover of size $k'' = |T'| + k = |T| + 1 + k$ if and only if G' has a steiner tree containing k vertices from N if and only if all vertices from T can be dominated in G by k vertices from N.

6.4.3 Capacitated Vertex Cover

In CapVC we are asked to find a vertex cover on a graph where the vertices have capacities associated with them, and every vertex can cover at most as many edges as its capacity. The problem takes as input a graph $G = (V, E)$, a capacity function $cap : V \to \mathbb{N}^+$ and an integer k, and the task is to find a vertex cover C and a mapping from E to C in such a way that at most $cap(v)$ edges are mapped to every vertex $v \in C$. The parameter of this problem is k.

Next, we describe how to transform (G, k) into an instance $(G''' = (V''', E'''), cap, k''')$ of CapVC. First, for each vertex $u_i \in T$, add a clique to G''' that contains four vertices $u_i^0, u_i^1, u_i^2, u_i^3$. Second, for each vertex $v_i \in N$, add a vertex v_i''' to G'''. Finally, for each edge $\{u_i, v_j\} \in E$ with $u_i \in T$ and $v_j \in N$, add the edge $\{u_i^0, v_j'''\}$ to G'''. See Figure 8 for an illustration. The capacities of the vertices are defined as follows: For each vertex $u_i \in T$, the vertices $u_i^1, u_i^2, u_i^3 \in V'''$ have capacity 1 and the vertex $u_i^0 \in V'''$ has capacity $\deg_{G'''}(u_i^0) - 1$. Each vertex v_i''' has capacity $\deg_{G'''}(v_i''')$. Clearly, in order to cover the edges of the size-4 cliques inserted for the vertices of T, every capacitated vertex cover for G''' must contain all vertices $u_i^0, u_i^1, u_i^2, u_i^3$. Moreover, since the capacity of each vertex u_i^0 is too small to cover all edges incident to u_i^0, at least one neighbor v_j''' of u_i^0 must be selected into every capacitated vertex cover for G'''. Therefore, it is



An illustration of the reduction from RBDS to CapVC.

Figure 8: The polynomial parameter transformation from RBDS to CapVC.

not hard to see that G''' has a capacitated vertex cover of size $k''' = 4 \cdot |T| + k$ if and only if all vertices from T can be dominated in G by k vertices from N.

6.4.4 Bounded Rank Set Cover

Finally, an instance of Bounded Rank Set Cover consists of a set family $\mathcal{F}$ over a universe U where every set $S \in \mathcal{F}$ has size at most d, and a positive integer k. The task is to find a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\cup_{S \in \mathcal{F}'} S = U$. The problem is parameterized by $(k + d)$.

To transform (G, k) into an instance $(\mathcal{F}, U, k)$ of Bounded Rank Set Cover, add one element e_i to U for every vertex $u_i \in T$. For every vertex $v_j \in N$, add one set $\{e_i \mid \{u_i, v_j\} \in E\}$ to $\mathcal{F}$. The correctness of the construction is obvious, and since $|U| = |T|$, every set in $\mathcal{F}$ contains at most $d = |T|$ elements.

Thus we have the following theorem:

Theorem 8 ([DLS09]). *The problem* Steiner Tree *parameterized by* $(|T|, k)$*, and the problems* Connected Vertex Cover *and* Capacitated Vertex Cover*, both parameterized by* k*,*

and the problem Bounded Rank Set Cover *parameterized by* (k, d) *do not admit polynomial kernels unless* $coNP \subseteq NP/poly$.

6.5 Dominating Set on Degenerate Graphs

A d-degenerate graph is a graph in which every induced subgraph has a vertex of degree at most d. Bounded degenerate graphs form one of the largest class of graphs for which the dominating set problem is FPT (it is $W[2]$-hard in general graphs) and has a polynomial sized kernel. It has a kernel of size $O(k^{d^2})$ [PRS09]. To show that the Dominating Set problem in d-degenerate graphs does not have a kernel of size $poly(k, d)$ (unless $coNP \subseteq NP/poly$), we appeal to the fact that the Small Universe Hitting Set is compositional (we refer the reader to [DLS09] for details). In this problem we are given a set family $\mathcal{F}$ over a universe U with $|U| \leq d$ together with a positive integer k. The question is whether there exists a subset S in U of size at most k such that every set in $\mathcal{F}$ has a non-empty intersection with S. It can be shown that the Small Universe Hitting Set problem parameterized by the solution size k and the size $d = |U|$ of the universe does not have a kernel of size polynomial in (k, d) unless $coNP \subseteq NP/poly$:

Theorem 9 ([DLS09])**.** Small Universe Hitting Set *parameterized by solution size* k *and universe size* $|U| = d$ *does not have a polynomial kernel unless* $coNP \subseteq NP/poly$. *The* Dominating Set *problem parameterized by the solution size* k *and the size* c *of a minimum vertex cover of the input graph does not have a polynomial kernel.*

Theorem 9 has some interesting consequences. For instance, it implies that the Hitting Set problem parameterized by solution size k and the maximum size d of any set in $\mathcal{F}$ does not have a kernel of size $poly(k, d)$ unless $coNP \subseteq NP/poly$. The second part of Theorem 9 implies that the Dominating Set problem in graphs excluding a fixed graph H as a minor parameterized by $(k, |H|)$ does not have a kernel of size $poly(k, |H|)$ unless $coNP \subseteq NP/poly$. This follows from the well-known fact that every graph with a vertex cover of size c excludes the complete graph K_{c+2} as a minor. Similarly, since every graph with a vertex cover of size c is c-degenerate it follows that the Dominating Set problem in d-degenerate graphs does not have a kernel of size $poly(k, d)$ unless $coNP \subseteq NP/poly$.

Theorem 10 ([DLS09])**.** *Unless* $coNP \subseteq NP/poly$ *the problems* Hitting Set *parameterized by solution size* k *and the maximum size* d *of any set in* $\mathcal{F}$, Dominating Set in H-Minor Free Graphs *parameterized by* $(k, |H|)$, *and* Dominating Set *parameterized by solution size* k *and degeneracy* d *of the input graph do not have a polynomial kernel.*

7 Further Directions and Open Problems

Compositions, and polynomial parameter transformations are by now, known established means of showing hardness of polynomial sized kernels for parameterized problems. We outlined a number of examples in this survey. We outline here more recent developments and further directions.

7.1 Finer lower bounds for problems having polynomial kernels

In a recent development, Dell and van Melkebeek [DvM10] have obtained a strengthening of a result in [FS08] and using that they are able to show concrete lower bounds on problems that do

admit polynomial kernels. In particular they have shown that HITTING SET does not admit a kernel with $O(k^{d-\varepsilon})$ sets when parameterized by the solution size k and maximum set size d. The other highlights of the results in [DvM10] are the establishment of non existence of kernels with $O(k^{2-\epsilon})$ edges for VERTEX COVER and FEEDBACK VERTEX SET, unless $coNP \subseteq NP/poly$. These two problems have kernels having $O(k^2)$ edges (as described elsewhere in this article). In fact, using appropriate reductions, they show that any NP-hard node deletion problem based on a graph property inherited by subgraphs has such a kernel lower bound under the same assumptions.

See also [CFM09] for reductions that pay attention to the length of the composed instance. Here the authors describe a notion of "linear OR", where the length of the output is allowed to be only as long as $t \cdot (\max_i |x_i|)^{O(1)}$, where $|x_i|$ is the length of the i^{th} input, and t is the total number of instances. This is more restrictive than the usual composition, where the length of the output can be $(\sum_i |x_i|)^{O(1)}$. They show that if an NP-complete problem has a linear OR, then it doesn't even have what are called *psuedo-kernels* — kernels of the form $p(k)n^{1-\epsilon}$ where p is a polynomial and ϵ is any positive constant less than 1.

7.2 Kernels for Problems Without a Polynomial Kernel

Although a parameterized problem may not necessarily admit a polynomial kernel, it may admit many of them, with the property that the instance is in the language if and only if at least one of the kernels corresponds to an instance that is in the language. We would like as few kernels as possible (each one adds to the runtime of any algorithm that would use the kernels to solve the problem in question).

Multiple polynomial kernels are quite exciting in practice — in particular, it opens up the possibility of implementing algorithms that use parallel processing, since the kernels are independent of each other, and this potentially makes them very useful. In fact, it is perhaps worth studying multiple kernels in general, even when the problem admits a polynomial sized kernel. For example, suppose a problem has k^2 linear kernels, in addition to having a cubic kernel. A fixed-parameter tractable algorithm that tries a brute force search on each of the k^2 linear kernels will still have a better run time than a brute force search fixed-parameter tractable algorithm on the cubic kernel.

To our knowledge, the only problem for which this multiple polynomial kernels approach has been tried is the k-LEAF OUT-BRANCHING problem [FFL+09]. The authors show that the problem is unlikely to have a polynomial sized kernel while still obtaining n kernels of size $O(k^3)$ each.

It appears that we may obtain many polynomial kernels for a problem (Q, κ) if we know of another language Q' for which we already have a polynomial kernel, and Q can be re-stated as the boolean OR of many instances of Q'. For instance, consider k-PATH. We may fix an arbitrary vertex v and ask for a pointed $k-1$ path that begins at v. If we had a polynomial sized kernel for pointed k-PATH, then we would have n polynomial kernels for k-path, as we simply have to iterate over all choices of v. Unfortunately, pointed k-PATH is just as hard as k-PATH in the polynomial kernel context [CFM09], so this observation is not very useful.

Recall our observation that the idea of using multiple kernels may have applications that transcend the "no polynomial kernels" scene. It would be nice to see explorations of many polynomial kernels for other problems.

7.3 The Two-Parameter Context

Consider the problem of finding a dominating set with at most k vertices. On general graphs, the problem is well-known to be $W[2]$-hard when parameterized by the solution size. To "cope" with the hardness of the problem, we would like to restrict our attention to some subclass of graphs where we may hope to exploit a property of the subclass towards an efficient algorithm. Consider, for instance, graphs that have "small" vertex covers. These are graphs that have vertex cover of size t (or smaller) for an arbitrary but fixed t. Any such t cuts out a slice from the set of all graphs — we now attempt to describe this "restriction" as a parameter that accompanies these graphs.

Consider the problem of finding a k-sized dominating set on a graph G which has a vertex cover of size t. We may regard both k and t as parameters to the problem. Noting that $k \leq t$ (any vertex cover is also a dominating set), our notion of a "FPT" algorithm is now something that spends $f(t) \cdot p(n)$ time. Now equipped with the promise of a vertex cover of size t, do we get anywhere? The good news is that this variant is in FPT, the FPT algorithm runs in time $O(2^t)|V|^{O(1)}$. However, it is unlikely to admit a kernel of size $(k+t)^{O(1)}$ ([DLS09]).

Due to polynomial parameter transformations, the hardness result implies that Dominating Set in H-Minor Free Graphs parameterized by solution size k and $|H|$ does not have a $O((k+|H|)^{O(1)})$ kernel, and that Dominating Set parameterized by k and degeneracy d of the input graph has no $O((k+d)^{O(1)})$ kernel (unless $coNP \subseteq NP/poly$). On the positive side, the best kernel known so far for the problem of Dominating Set in Graphs With Small Vertex Cover is $O(2^t)$. We do not expect a $t^{O(1)}$ kernel here because of the hardness described. The more compelling situation arises with Dominating Set on d-degenerate graphs, where a $k^{O(d^2)}$ kernel is known ([PRS09]). While a kernel of size polynomial in k and d has been ruled out, what about kernels of size $f(d)k^{O(1)}$? Can the recent techniques and/or the results of [DvM10] be used to rule out such kernels?

Uniform Kernels. In general, for such problems with two parameters, say k and l, the size of the kernel may be a function of any one of the following kinds:

- $f(k,l)$: An arbitrary function of k and l alone.
- $p(k)^{f(l)}$: A polynomial in k with an arbitrary function of l in the exponent.
- $p(k) \cdot f(l)$: A polynomial in k with the arbitrary function of l as a multiplicative factor[2].
- $p(k)p(l)$: Polynomial in both parameters

Note that, as usual, the "arbitrary" function refers to a computable one.

It seems a bit of a trend among FPT problems with two parameters — the "purely polynomial" kernels are shown to be unlikely, and it is trivial to note that the $f(k,l)$-sized kernels exist (for problems in FPT with one parameter, we had an argument establishing that this implies a kernel — this is easily generalized to accommodate for more parameters). Typically, a $p(k)^{f(l)}$ kernel exists, and the crucial question is unanswered — may we have a kernel whose size is a polynomial in one of the parameters, with an arbitrary function of the other occurring only as

[2]This should remind us of the difference between XP and FPT!

a multiplicative factor? Unfortunately, we do not know of a "lower bound scheme" that will rule out such a possibility, and this is an important question.

An example that emphasizes this dilemma is the so-called "Small Universe Hitting Set" problem, which is the familiar Hitting Set problem except that the universe size is also a parameter:

Small Universe Hitting Set	
Input:	A set family $\mathcal{F}$ over a universe U with $\|U\| = d$, and a positive integer k.
Question:	Is there a subset $H \subseteq U$ of size at most k such that for every set $S \in \mathcal{F}, H \cap S \neq \phi$?
Parameter(s):	k, d

The best algorithm known has a runtime of $O(2^d(|\mathcal{F}| + |U|)^{O(1)})$, and the best known kernel is of size $O(2^d)$ — no $O((k+d)^{O(1)})$ kernel is expected ([DLS09]).

The $p(k) \cdot f(l)$-sized kernels have even acquired a name by now — they are called uniformly polynomial kernels, and we emphasize that a lower bound framework for ruling out such kernels would be intriguing — and useful.

Open Problem 1. *In problems that involve two parameters, say k and l, establish new methods, or use existing ones, to show that no kernels of the form $f(k) \cdot p(l)$ exist. Specific problems where the question may be posed are* Dominating Set *on d-degenerate graphs, and* Small Universe Hitting Set.

7.4 Turing Kernelization

In [FFL+09] it is shown that k-Leaf Out-Branching admits n independent kernels of size $O(k^3)$. It was not a kernel in the usual "many to one" sense, though it was kernel in the "one to many" sense. We can generalize the notion of many to one kernels to capture the kind of kernels we saw in Section 7.2 for k-Leaf Out-Branching. This brings us to the notion of Turing kernelization. In order to define this we first define the notion of *t-oracle*.

Definition 8 ([FFL+09])**.** A t-oracle for a parameterized problem Π is an oracle that takes as input (I, k) with $|I| \leq t$, $k \leq t$ and decides whether $(I, k) \in \Pi$ in $O(t)$ time.

Definition 9 ([FFL+09])**.** A parameterized problem Π is said to have *$g(k)$-sized Turing kernel* if there is an algorithm which given an input (I, k) together with a $g(k)$-oracle for Π decides whether $(I, k) \in \Pi$ in time polynomial in $|I|$ and k.

Observe that the well known notion of kernel or many to one kernel is a special case of Turing kernelization. In particular, many to one kernels are equivalent to Turing kernels where the kernelization algorithm is only allowed to make one oracle call and must return the same answer as the oracle.

Open Problem 2 ([FFL+09])**.** *Is there a framework to rule out the possibility of having polynomially many Turing kernels similar to the framework developed in [BDFH09, FS08]?*

Dell and van Melkebeek [DvM10] have shown that k-Leaf Out-Branching is unlikely to have, for any $\epsilon > 0$, $n^{1-\epsilon}$ independent kernels, each of size polynomial in k.

Open Problem 3 ([FFL+09]). *Which other problems admit a Turing kernelization like the quadratic kernels for k-Leaf Out-Branching and k-Leaf Out-Tree? Does the problem of finding a path of length at most k admit a Turing kernel (even on planar graphs)?*

Open Problem 4 ([FFL+09]). *Does there exist a problem for which we do not have a linear many-to-one kernel, but does have linear kernels from the viewpoint of Turing kernelization?*

7.5 Specific Problems

There are many problems for which polynomial kernels are not known, and a hardness result has not been established either. Given a graph G and a positive integer k, the ECC problem asks for k cliques that cover all edges of G, and the OCT problem asks for k vertices whose deletion makes the graph bipartite. Given a directed graph G, the DFVS problem asks for a FVS of size at most k. In all cases, the parameter of the problem is k.

Open Problem 5. *Does ECC admit a polynomial kernel?*

Open Problem 6. *Does OCT admit a polynomial kernel?*

Open Problem 7. *Does DFVS admit a polynomial kernel?*

7.6 Unconditional Lower Bounds

The other obvious set of questions to ask is whether some of these lower bound theorems can be improved to unconditional statements. A less ambitious goal would be to attempt proving the same results under some "well-believed" conjectures of parameterized complexity, for instance, $FPT \neq W[P]$. It is known that there exists a FPT language that indeed does not admit a polynomial kernel, unconditionally:

Theorem 11. *([BDFH09]) There is an FPT langauge $L \subseteq \Sigma \times N^+$ solvable in $O(2^k n)$ time, $n = |x| + k$, with no kernelization of size $g(k) = 2^{o(k)}$.*

7.7 An AND Conjecture?

Much of our focus has been on composition algorithms that perform some kind of a boolean "or" on their inputs. Their counterparts are algorithms that similarly perform an "and" of the inputs, however, the existence of such algorithms for NP-complete problems is not known to cause any unlikely collapse yet, and is therefore not useful for proving lower bounds. It is an interesting open problem to pursue the implications of the existence of "and" compositions for classical and parameterized languages. These kind of and-based-compositions exist for graph layout problems like Treewidth, Pathwidth, Cutwidth and other problems like Independent Set, Dominating Set when parameterized by the treewidth of the input graph. For further discussion, see [BDFH09].

7.8 Recent Developments in Upper Bounds

In this article we only talked about lower bound results, but lately there has been an explosion of research in proving upper bounds on kernel sizes for several problems. The most significant ones

include meta theorems for kernelization [BFL+09, FLST10, Kra09], use of probabilistic tools and Fourier analysis [AGK+10, GKSY10, GKMY10, GvIMY10] and non-trivial applications of combinatorial min-max results [FGMN09, FGST09, LS09, Tho09].

7.9 Concluding Remarks

The notion of kernelization is popular in practice — in many cases, it can be thought of as a precise way of stating all the heuristic-based preprocessing steps that have been popular and effective for a long time. In theory, the notion is important for more than one reason — there is an increasingly popular feeling that kernelization is *the* way of understanding fixed-parameter tractability. The theorem that establishes the equivalence of these notions is more than a syntactic equality — it encodes an entire philosophy, and immediately puts on offer a possible "right way" of viewing FPT. Given that FPT is the most fundamental class of parameterized problems, the fact that we finally have a few lower bounds for placing problems in different places *within* this class is a great source of excitement, and a non-trivial hope for a deeper understanding of problem complexity.

References

[AFN04] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier, *Polynomial-time Data Reduction for Dominating Set*, Journal of the ACM **51** (2004), no. 3, 363–384. 2

[AGK+10] Noga Alon, Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo, *Solving MAX-r-SAT Above a Tight Lower Bound*, Algorithmica, in press. (2010). 28

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick, *Color-coding*, Journal of the ACM **42** (1995), no. 4, 844–856. 8

[BDFH09] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin, *On Problems Without Polynomial Kernels*, J. Comput. Syst. Sci. **75** (2009), no. 8, 423–434. 1, 2, 3, 6, 7, 8, 26, 27

[BFL+09] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos, *(Meta) Kernelization*, FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), IEEE Computer Society, 2009, pp. 629–638. 28

[BG93] Jonathan F. Buss and Judy Goldsmith, *Nondeterminism Within P*, SIAM Journal on Computing **22** (1993), no. 3, 560–572. 5

[Bod07] Hans L. Bodlaender, *A Cubic Kernel for Feedback Vertex Set*, STACS'07: Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science (Berlin, Heidelberg), Springer-Verlag, 2007, pp. 320–331. 18

[BTY09] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo, *Kernel Bounds for Disjoint Cycles and Disjoint Paths*, Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009., Lecture Notes in Computer Science, vol. 5757, 2009, pp. 635–646. 13, 14, 16, 17, 18, 19

[CC08] Miroslav Chlebík and Janka Chlebíková, *Crown Reductions for the Minimum Weighted Vertex Cover Problem*, Discrete Appl. Math. **156** (2008), no. 3, 292–312. 6

[CFKX07] Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia, *Parametric Duality and Kernelization: Lower Bounds and Upper Bounds on Kernel Size*, SIAM J. Comput. **37** (2007), no. 4, 1077–1106. 2

[CFM09] Yijia Chen, Jörg Flum, and Moritz Müller, *Lower Bounds for Kernelizations and Other Preprocessing Procedures*, CiE '09: Proceedings of the 5th Conference on Computability in Europe (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 118–128. 8, 10, 24

[CKJ01] Jianer Chen, Iyad A. Kanj, and Weijia Jia, *Vertex cover: Further Observations and Further Improvements*, J. Algorithms **41** (2001), no. 2, 280–301. 2

[DF95a] R. G. Downey and M. R. Fellows, *Parameterized Computational Feasibility*, P. Clote, J. Remmel (eds.): Feasible Mathematics II, Boston: Birkhäuser, 1995, pp. 219–244. 17

[DF95b] Rod G. Downey and Michael R. Fellows, *Fixed-parameter Tractability and Completeness I: Basic Results*, SIAM J. Comput. **24** (1995), no. 4, 873–921. 17

[DF95c] ———, *Fixed-parameter Tractability and Completeness II: On Completeness for $W[1]$*, Theor. Comput. Sci. **141** (1995), no. 1-2, 109–131. 17

[DF99] Rod G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer, November 1999. 4

[Die00] Reinhard Diestel, *Graph Theory*, 2 ed., Springer-Verlag, New York, 2000. 3

[DLS09] Michael Dom, Daniel Lokshtanov, and Saket Saurabh, *Incompressibility Through Colors and IDs*, ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 378–389. 8, 11, 12, 16, 19, 20, 22, 23, 25, 26

[DvM10] Holger Dell and Dieter van Melkebeek, *Satisfiability Allows no Nontrivial Sparsification Unless the Polynomial-time Hierarchy Collapses*, STOC '10: Proceedings of the 42nd ACM symposium on Theory of Computing (New York, NY, USA), ACM, 2010, pp. 251–260. 23, 24, 25, 26

[EP65] P. Erdős and P. Pósa, *On Independent Circuits Contained in a Graph*, Canadian Journal of Mathematics **17** (1965), 347–352. 18

[FFL+09] Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Daniel Raible, Saket Saurabh, and Yngve Villanger, *Kernel(s) for Problems with No Kernel: On Out-Trees with Many Leaves*, Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2009, pp. 421–432. 24, 26, 27

[FG06] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer-Verlag New York Inc, 2006. 3, 4

[FGMN09] Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier, *A Generalization of Nemhauser and Trotter's Local Optimization Theorem*, Proceedings of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2009, pp. 409–420. 28

[FGST09] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Stéphan Thomassé, *A Linear Vertex Kernel for Maximum Internal Spanning Tree*, ISAAC 2009: Proceedings of the 20th International Symposium on Algorithms and Computation, Springer-Verlag, 2009, pp. 275 – 282. 28

[FKWW04] Fedor V. Fomin, Dieter Kratsch, J. Woeginger, and Gerhard J. Woeginger, *Exact (exponential) Algorithms for the Dominating Set Problem*, in Proceedings of the 30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004, Springer, 2004, pp. 245–256. 19

[FLST10] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos, *Bidimensionality and Kernels*, ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010, pp. 503–510. 28

[FS08] Lance Fortnow and Rahul Santhanam, *Infeasibility of Instance Compression and Succinct PCPs for NP*, STOC '08: Proceedings of the 40th annual ACM symposium on Theory of Computing (New York, NY, USA), ACM, 2008, pp. 133–142. 6, 7, 23, 26

[GGHN03] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier, *Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation*, In Proc. 5th CIAC, Volume 2653 of LNCS, Springer, 2003, pp. 108–119. 6

[GJ90] Michael R. Garey and David S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1990. 12, 20

[GKMY10] Gregory Gutin, Eun Jung Kim, Matthias Mnich, and Anders Yeo, *Betweenness Parameterized Above Tight Lower Bound*, J. Comput. Syst. Sci. **76** (2010), no. 8, 872–878. 28

[GKSY10] Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo, *A Probabilistic Approach to Problems Parameterized above or below Tight Bounds*, J. Comput. Syst. Sci., in press (2010). 28

[GN07] Jiong Guo and Rolf Niedermeier, *Invitation to Data Reduction and Problem Kernelization*, SIGACT News **38** (2007), no. 1, 31–45. 2, 6

[GvIMY10] Gregory Gutin, Leo van Iersel, Matthias Mnich, and Anders Yeo, *All Ternary Permutation Constraint Satisfaction Problems Parameterized above Average Have Kernels with Quadratic Numbers of Variables*, Algorithms - ESA 2010, 18th Annual European Symposium Proceedings, Part I, 2010, pp. 326–337. 28

[HNW08] Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke, *Techniques for Practical Fixed-Parameter Algorithms*, The Computer Journal **51** (2008), no. 1, 7–25. 4

[Kra09] Stefan Kratsch, *Polynomial Kernelizations for MIN F^+ Π_1 and MAX NP*, STACS'09: Proceedings of the 26th Annual Conference on Theoretical Aspects of Computer Science, 2009, pp. 601–612. 28

[KW] Stefan Kratsch and Magnus Wahlström, *Preprocessing of Min Ones Problems: A Dichotomy*, ICALP 2010: Proceedings of the 35th International Colloquium on Automata, Languages and Programming. 10

[LS09] Daniel Lokshtanov and Saket Saurabh, *Even Faster Algorithm for Set Splitting!*, Parameterized and Exact Computation: 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 2009, 2009, pp. 288–299. 28

[Nie06] Rolf Niedermeier, *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*, Oxford University Press, USA, March 2006. 2, 4, 6, 10

[PRS09] Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar, *Solving Dominating Set in Larger Classes of Graphs: FPT Algorithms and Polynomial Kernels*, Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September, Lecture Notes in Computer Science, vol. 5757, 2009, pp. 694–705. 23, 25

[Tho09] Stéphan Thomassé, *A Quadratic Kernel for Feedback Vertex Set*, SODA '09: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2009, pp. 115–119. 2, 6, 18, 28

[Wei98] Karsten Weihe, *Covering Trains by Stations or the Power of Data Reduction*, OnLine Proceedings of ALEX'98 - 1st Workshop on Algorithms and Experiments, 1998, pp. 1–8. 2

[Yap83] Chee-Keng Yap, *Some Consequences of Non-Uniform Conditions on Uniform Classes*, Theor. Comput. Sci. **26** (1983), 287–300. 7