Structural Parameterized Complexity

MSc Thesis *(Afstudeerscriptie)*

written by

Jouke E. Witteveen
(born January 28, 1988 in Leeuwarden)

under the supervision of **Dr Leen Torenvliet**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
January 28, 2015	Prof Dr Hans L. Bodlaender
	Prof Dr Harry M. Buhrman
	Dr Leen Torenvliet
	Prof Dr Ronald M. de Wolf (chair)

Institute for Logic, Language and Computation

Contents

Abstract

We revisit fixed-parameter tractability, fixed-parameter reducibility and kernelizability. The standard formulations of their definitions are enhanced for greater correctness from a structural complexity point of view. Doing so makes clear the distinction between fixed-parameter reducibility and kernelizability. Additionally, these time based definitions are augmented with space based counterparts. We then proceed to introduce an informativeness property for parameterized problems and define a measure for the computational complexity of individual instances of parameterized problems. By its construction, this measure embodies the idea that parameters capture complexity. This measure is lower bounded by the resource bounded Kolmogorov complexity, where the resource bound depends on the computational complexity of the parameterized problem.

1 Introduction

We give an informal introduction to the topics this thesis is concerned with, leaving a formal treatment to the subsequent sections. An overview of the contents of this thesis is provided, so that the reader can skip some or all of the thesis, depending on his/her (current) interests.

Computationally hard problems are an integral part of every day life. The boundary between computationally hard and computationally not hard is a popular area of exploration in computer science. Although computer science has an undeniable presence in society, much of its power is a result of advances in hardware. With computers becoming faster and cheaper, we can take on more and more computational problems. There are computational problems, however, for which making computers faster and more plentiful is of little use. While many such intrinsically intractable problems are known, a definitive separation of the intractable from the tractable is a major unsolved problem in computer science. For a large class of problems known as the **NP**-complete problems, their presumed intractability is phrased as the statement that they do not lie in a class known as **P**. A definitive proof of their intractability would settle the **P** versus **NP** problem, which has been troubling computer scientists at least since it was formulated by Cook in 1971.

In the 1990s a more detailed view to the study of tractability and intractability was introduced. This parameterized take on tractability has proven of use for both theoretical and practical computer science. Textbooks have appeared both on the classification of problems in the framework of parameterized complexity and on algorithm design using methods aimed at fixed-parameter tractability. Still, we feel that parameterized complexity has far greater potential then is reflected in the current body of research.

Most parameterized complexity classes currently studied are derived from the class of problems that are fixed-parameter tractable in a strong sense with respect to computation time. We feel that this leaves us with a rather limited complexity zoo and see an opportunity for the development of more diverse parameterized complexity classes and corresponding notions of preprocessing. In this thesis, we will make a humble beginning with such a diversification by introducing the notion of fixed-parameter tractability with respect to the working memory requirements of computations.

There are also lines of research to which parameterized complexity contributes invaluable new methods which have yet to find their way into textbooks. One such line of research is that of the distribution of complexity over the instances of problems. Given an intractable problem, one could wonder how many instances are responsible for the intractability of that problem. Using parameter values to keep track of the individual hardness of problem instances, parameterized complexity provides a framework for the analysis of such questions.

In this thesis, such questions are looked at from a structural standpoint, that is, without referring to specific computational problems. Nevertheless, specific problems are used as examples and to demonstrate results. Structural considerations have the potential to prove separation results about complexity classes and thus parameterized complexity might be used as a tool on such an endeavor too. Even for the **P** versus **NP** problem, parameterized complexity might offer new insights, or at least provide a clean formalism for the formulation

of existing insights.

This thesis first works its way to an exposition of some central notions of parameterized complexity in Section 4. While revisiting the basics, we identify some shortcomings in the customary definitions. In Section 5, the definition of fixed-parameter tractability with respect to computation time is lifted into the realm of fixed-parameter tractability with respect to the working memory requirements of computations. The main results and contributions of this thesis to parameterized complexity are in Section 6 and Section 7. Section 6 points out a link to descriptive complexity theory which reflects the interpretation of the parameter as a measure of the complexity of problem instances. Section 7 contains an investigation of what is needed from a parameter to make an arbitrary problem fixed-parameter tractable. This can be interpreted as an investigation of the limits of the usability of parameterized complexity, or as a foray into the kinds of restrictions that could be placed on parameters. For the sake of comparing our methods to classical methods from probability theory, we stray from our structural ways and present, in Section 8, an analysis of the parameter values of a specific graph problem. Compensating for this digression, a fundamentally structural approach to parameterized complexity is tried in Section 9, using a form of reducibility as the central concept. Although of very limited success, this approach does inspire one of the conjectures listed in the section on possible future research in structural parameterized complexity, Section 10.

2 History

Parameterized complexity is a young branch of computational complexity theory. We give a brief overview of its history and of its motivation. As the history of computer science is well documented [19] and references are readily available, we choose to not clutter the bibliography with references that do not relate to the subject matter of this thesis.

With the advent of several, alternative, formalizations of the notion of computability in the 1930s, a rigorous study of computability theory became possible. This was the birth of recursion theory. It was only after digital computers had materialized in the 1940s and 1950s that questions about tractability became prevalent in mathematics and computer science. The most operable formalization of the notion of computability from the point of view of questions about tractability turned out to be that of Turing, and in the early 1960s Hartmanis and Stearns put forth the idea of measuring resource usage of Turing machines as functions of the length of the input provided to them. As a result, complexity theory came into existence. Halfway the 1960s Edmonds and Cobham independently argued that any Turing machine that requires an amount of 'time' that is upper bounded by a polynomial in the length of its input should be considered to constitute an effective method. This definition was widely accepted, despite an apparent objection in the form of the time hierarchy theorem of Hartmanis and Stearns. Under the characterization of effective methods by Edmonds and Cobham, the time hierarchy theorem holds that for any given polynomial of arbitrary degree, there exists an effective method of which the running time is not upper bounded by the given polynomial. By the 1970s, the definition of the class **P** of problems solvable by an effective method

was firmly established and **P** was recognized as the complexity class representing tractability. However, by the 1990s this take on tractability had become unsatisfactory for a variety of reasons and in a series of four groundbreaking papers, Downey and Fellows introduced a new standard for tractability: fixed-parameter tractability. This new notion was formalized in the complexity class **FPT** and accompanied by a suitable notion of reducibility, facilitating the structural study of this new complexity class and its relatives [18]. We will mention some of the most important arguments in favor of fixed-parameter tractability as an approach to tractability.

The use of parameters in the specification of problems makes more fine-grained expressions of complexity possible. With parameters representing metrics or other aspects of the input to a Turing machine, it is possible give a description of the resource requirements of that machine which incorporates these aspects. Such a description is based not only on the length of the input, but also on the parameters. In particular, a parameterized analysis can distinguish complexities that are indistinguishable when expressed as a function of only the length of the input.

In practice, problems for which no tractable solution is known do not always pose problems. This is mostly due to the fact that not all possible inputs of any given length are equally likely to be encountered. In many practical situations, data tends to be cooperative, save for some relatively small troublesome part. Keeping the size of such a troublesome part fixed and looking at the complexity behavior for variable input sizes is precisely the fixed-parameter look at complexity. Thus fixed-parameter tractability promises to give a more realistic account of tractability in practice. From a theoretical perspective, we could say that parameterizing problems allows prior knowledge about values of metrics or of other aspects of problem instances to be taken into account when considering tractability. Thus fixed-parameter tractability is a way to loosen the constraints placed on tractability by the classical notion captured by **P**.

A related motivation for fixed-parameter tractability comes from the desire to understand sources of intractability. For many problems, intractability is a result of a combinatorial explosion. That is, a search space related to solving such a problem has a size that cannot be bounded by any polynomial in the length of the input to a machine that is supposed to solve the problem. Still, there may be an upper bound, however big, to the size this search space that can be expressed in terms of an aspect of the input instead of in terms of the length of the input. If the length of the input and the aspect are not too strongly correlated, such an upper bound should be thought of as confining the combinatorial explosion. After all, for some inputs of great length the aspect value may nonetheless indicate a relatively limited search space. This is the case, for example, when the input specifies a rather redundant problem instance. In such cases it is often possible to strip away the redundant parts and reduce an instance to its more or less intrinsically hard part. Here, we find a connection between parameterized complexity theory and preprocessing practice. Moreover, we notice that parameterized complexity enables an investigation of the distribution of the source of complexity, where this distribution is considered within problems.

Classically, the study of the internal structure of problems has, for classes of intractable problems, led to the notion of a natural problem. Specifically, **NP**-complete problems are called natural whenever they are p-cylinders, for

which a definition can be found in Section 7. It is an open problem whether or not all **NP**-complete problems are natural. For parameterized versions of **NP**-complete problems it is unclear which problems to call natural. In fact, such structural considerations are seemingly underdeveloped in current parameterized complexity research. Suggestions for conditions that should be satisfied by natural parameterized problems are included in Section 6, but all suggested conditions are unsatisfactory. Either they exclude problems which we would want included, or they allow parameterizations such as those constructed in Section 7, which we would want rejected. Nevertheless some parameterizations are intuitively more natural than others and research in parameterized tractability has commonly focussed on those problems that are natural in an intuitive sense. The alternative program of finding parameterized problems related to a classically intractable problem that are both natural in some sense and have favorable fixed-parameter tractability is rarely attempted. Although several textbooks on parameterized complexity have already appeared and results have found applications outside academia, the field of parameterized complexity theory is still in flux and its foundations are not yet set in stone.

3 Preliminaries

We review the small number of basic definitions from computer science that are used in this thesis. None of these definitions is unusual, so the reader familiar with computer science may skip this section.

Conceptually, complexity theory is about the difficulty of problems. In order to attempt a formal study, proper definitions are needed. One of the most common perceptions of a problem in complexity theory [3] is that of a decision problem, which is the type of problem we will define in this section and study in this thesis.

In computer science, an *alphabet* is a finite set and every finite set can serve as an alphabet. A finite sequence of characters of an alphabet is called a *string*. If Σ is an alphabet, the set of all possible strings of d characters of that alphabet is denoted by Σ^d. The following notational conventions are used throughout this thesis.

Definition 1. The natural numbers start at 1. For every natural number n, the set of all possible nonempty strings of at most n characters of an alphabet Σ is denoted by $\Sigma^{\leq n}$. The set of all nonempty finite strings of an alphabet Σ is denoted by Σ^+, where the $+$ is known as the Kleene plus operator.

$$\begin{aligned} \mathbb{N} &= \{1, 2, \ldots\}, \\ \Sigma^{\leq n} &= \bigcup_{d=1}^{n} \Sigma^d, \\ \Sigma^+ &= \bigcup_{d \in \mathbb{N}} \Sigma^d. \end{aligned}$$

Note that the unions in the above definition are disjoint. For all $n \in \mathbb{N}$, the following length norm is therefore well-defined on $\Sigma^{\leq n}$ and Σ^+.

Definition 2. The *length* of a string $x \in \Sigma^d$ is d and denoted by $|x|$.

A subset of Σ^+ is called a *language* and comes with its own norm.

Definition 3. The *size* of a language $A \subseteq \Sigma^+$ is its cardinality and denoted by $\|A\|$.

We call the empty language, $\emptyset$, and the full language, Σ^+, the *trivial* languages. Usually in computer science, Σ^+ is extended to include the empty sequence and the result of this extension is denoted by $\Sigma^\star$. However, a string of length 0 is often inconvenient and sometimes even invalidates theorems. In fact, we would at times like the length of any string x to be at least 2, so that $|x|^c$ increases monotonically as a function of c.

Note that all infinite languages have the same cardinality and there exists a bijective correspondence between any countably infinite set and any infinite language. In specific cases, this correspondence can be of a very practical nature. For instance, a language using one alphabet can be related to a language using another alphabet by *encoding* the symbols of the one alphabet in fixed-length sequences of symbols of the other. Therefore, the following convention does not limit our results.

Definition 4. Throughout this thesis Σ denotes the binary alphabet, $\{0, 1\}$, and log denotes the binary logarithm.

Another case of a practical bijection is that of a language corresponding to the Cartesian product $\Sigma^+ \times \Sigma^+$. Just mapping an element (x, y) of $\Sigma^+ \times \Sigma^+$ to the concatenation of x and y does not define a bijection between $\Sigma^+ \times \Sigma^+$ and Σ^+, as we do not know where x ends and y begins in the concatenation. A proper correspondence can be found by using an intermediate self-delimiting language. A language A is *self-delimiting* if for every $x \in A$ and $y \in \Sigma^+$ we have that the concatenation of x and y is not in A. Note that no self-delimiting language of size greater than 1 contains the empty string. A self-delimiting language for which we have a direct bijective correspondence with Σ^+ is the language where, for all $d \in \mathbb{N}$, every $x \in \Sigma^d$ is represented by the string obtained by concatenating $d - 1$ '1's, a '0' and x. If we call this language A, then we have just defined a correspondence between $\Sigma^+ \times \Sigma^+$ and $A \times \Sigma^+$. From the concatenation of the components of an element of $A \times \Sigma^+$ it is possible to recover the element itself, thus all such concatenations form a language that is in bijective correspondence to $A \times \Sigma^+$ and thus to $\Sigma^+ \times \Sigma^+$. Calling the bijective mapping from $\Sigma^+ \times \Sigma^+$ to this language f, the practical use of f is demonstrated by the fact that we have $|f(x, y)| = 2|x| + |y|$, which demonstrates that a well-behaved length norm can be defined on products of languages.

The language corresponding to $\Sigma^+ \times \Sigma^+$ just constructed serves as an example of a language that has an *interpretation* of its elements. An interpretation that is possible for all strings is given by a correspondence between $\mathbb{N}$ and Σ^+. Any string of characters of our binary alphabet can be considered to be the binary representation of a natural number: '0' corresponds to 1, '1' corresponds to 2, '00' corresponds to 3, '01' corresponds to 4 et cetera. The representation may appear somewhat strange, but, for $n \in \mathbb{N}$, it is just the customary representation of $n + 1$, dropping the leading '1'. This interpretation induces an order on Σ^+. Even when a language is associated with an interpretation different from that as natural numbers, it can be practical from a computational point of view to nevertheless use the interpretation as natural numbers.

A more general form of correspondence between languages is via language maps.

Definition 5. A *language map* from a language A to a language B is a function $f : \Sigma^+ \to \Sigma^+$ such that, for all $x \in \Sigma^+$, we have $x \in A \iff f(x) \in B$.

Sometimes, when the meaning is clear from the context, language maps are simply called maps. Language maps form the basis of many-one reductions, which we will simply call reductions in this thesis.

Definition 6. A class of language maps $\mathcal{C}$ is a class of *reductions* if we have:

1. for every language, the identity map on that language is in $\mathcal{C}$;
2. for every two maps in $\mathcal{C}$, their composition is in $\mathcal{C}$.

The language maps from a language A to a language B in $\mathcal{C}$ are denoted by $\mathcal{C}(A, B)$.

The properties of a class of reductions make sure that a class of reductions $\mathcal{C}$ defines a preorder $\preceq_{\mathcal{C}}$ on all languages: $A \preceq_{\mathcal{C}} B \iff \mathcal{C}(A, B) \neq \emptyset$. That is, the properties of a class of reductions warrant reflexivity and transitivity of the given preorder. When composition of reductions is associative, languages and reductions form a category.

Lacking from the discussion so far, but of paramount importance to computer science, is the notion of computability. If any theorem deserves the title of fundamental theorem of computer science it would be that the class of computable functions is largely invariant under the choice of a formalism for the underlying notion of computability. This theorem is the result of the Church–Turing thesis when the Turing machine, or any other equivalent formalism, is taken as the defining formalization of computability. The prominent formalization of computability in this thesis will be a form of pseudocode that we consider self-explanatory and for which we will not define an interpreter. We will call a, possibly partial, function *computable* if there is an algorithm specified in our pseudocode that computes it.

Definition 7. An *algorithm*, or *procedure* is any computable function or specification thereof according to some formalization of computability.

Thus, there are two kinds of equivalence of algorithms: an extensional one based on their behavior as a function and an intensional one based on their specification.

On inputs where a procedure does not terminate, the corresponding computable function is not defined. However, for our purposes we mostly look at resource bounded computations, in which case we may assume procedures terminate on all possible inputs. Therefore, we may assume that all resource bounded computable functions are total functions.

We will be especially interested in decision procedures.

Definition 8. An algorithm $f : \Sigma^+ \to \{\mathbf{true}, \mathbf{false}\}$ *decides* membership of a string $x \in \Sigma^+$ in a language A if we have:

$$f(x) = \begin{cases} \mathbf{true} & \text{if } x \in A \\ \mathbf{false} & \text{if } x \notin A \end{cases}.$$

An algorithm is a *decision procedure* if it decides membership of all strings $x \in \Sigma^+$ in A.

Any language for which there is a decision procedure is called *decidable* and the membership question for a string in a decidable language is called a *decision problem*. Slightly abusing terminology, we adopt the following definition.

Definition 9. A *problem* is a language that admits a decision procedure.

Computability and decidability are related and generally the term *recursive* is used to designate the overarching concept, but we will not do so here. When it comes to computability or decidability we are interested in the amount of some resource required by the computation involved. In particular, we are interested in the time and space usage of a computation. Here, time usage stands for the number of atomic computational steps that a computation is made up of. Space usage stands for the longest length of the specification of the input-dependent state encountered during a computation. That is, space usage is about the memory requirement of a computation. We will say a procedure is computable in time t to indicate that computation of the procedure terminates after no more than t atomic computational steps. Similar phrases will be used for the decidability of languages instead of computability of procedures and for space instead of time. When we focus on the behavior of resource usage more than on the specifics, we will make use of $\mathcal{O}$-notation. Let f and g be functions from an arbitrary domain A to $\mathbb{N}$. In a context where we have an implicit abstraction over a variable x in A, we say that $f(x)$ is in $\mathcal{O}(g(x))$ if there is a constant c such that for all but finitely many x in A we have $f(x) \leq cg(x)$. In this thesis, c can always be chosen so that the inequality holds for all x in A. We will often use implicit function definitions in place of f and g. Thus, for example, the phrase 'the elements (x, y) of a language A are such that $|x|$ is in $\mathcal{O}(y^2)$' comes to mean the same as the logical expression

$$\exists c : \forall (x, y) : (x, y) \in A \implies |x| \leq cy^2.$$

Even more specifically, with f mapping (x, y) to $|x|$ and g mapping (x, y) to y^2, the phrase means the same as the expression

$$\exists c : \forall (x, y) : (x, y) \in A \implies f(x, y) \leq cg(x, y).$$

Similarly, a decision procedure taking input x is said to decide a problem in space $\mathcal{O}(s(|x|))$ if there is a constant c such that for all $x \in \Sigma^+$ the space used by the decision procedure is at most $cs(|x|)$. A special phrase is used for procedures taking input x for which there is a d that does not depend on x such that the time or space usage of the procedure is in $\mathcal{O}(|x|^d)$. Such procedures are said to be computable in polynomial time or space. We want to emphasize that $\mathcal{O}$-notation is about functions, not about specific function values. Our choice of notation is motivated by the fact that for the implicit specification of functions it is convenient to be able to refer to function arguments.

As a running example of a problem, in this thesis we will look at the vertex cover problem on simple graphs.

Definition 10. A *simple graph* consists of a finite set V of which the elements are called *vertices* and a symmetric anti-reflexive binary relation E on V of which the related pairs are called *edges*.

Simple graphs can be visualized by depicting the vertices as dots and drawing lines between related vertices. Because E is anti-reflexive, no line will be drawn from a dot directly to itself.

Definition 11. Given a graph (V, E), a subset V' of V is a *vertex cover* of (V, E) if for every $(v_1, v_2) \in E$ we have $v_1 \in V'$ or $v_2 \in V'$. The *size* of a vertex cover V' is the cardinality of V'.

The *vertex cover problem* is the language of pairs of a simple graph (V, E) and a natural number k such that there is a vertex cover of size at most k in (V, E). Note that there are countably many pairs of a simple graph and a natural number, so there indeed exists a language admitting such an interpretation. It is not known whether the vertex cover problem is decidable in polynomial time. In fact, it is complete for the complexity class **NP** of problems that are computable in polynomial time for a formalization of computability that includes nondeterminism. Here, completeness is taken with respect to polynomial time reductions known as *Karp reductions*. Related to the vertex cover problem is the *minimum vertex cover problem*, which asks whether a number k is the smallest possible size of a vertex cover in a graph. For the minimum vertex cover problem it is not even known whether it is in **NP**. It is, precisely when **NP** equals the class **co-NP** of problems of which the complement is in **NP**.

4 Parameterized Complexity

A structural outline of fixed-parameter tractability is given. On the way, we identify some flaws in the textbook treatments and correct them. The interplay between several concepts that play a central role in the field of parameterized complexity theory is explored.

As with all interesting complexity classes, there is a multitude of ways in which the classes relevant for analysis in parameterized complexity research can be defined. Characterizations of the classical **NP** include those via nondeterministic Turing machines, via certificates and verifiers, and via reductions and complete problems. Similarly, we will provide several approaches to the classes most important for this thesis.

Before we do so, some attention needs to be given to the concept of a parameterized problem. For this concept, like for many in parameterized complexity theory, different definitions exist. In this thesis, we will go with the one introduced originally by Downey and Fellows [13, 14].

Definition 12. A *parameterized problem* is a subset of $\Sigma^+ \times \Sigma^+$. For $(x, k) \in \Sigma^+ \times \Sigma^+$ we call x the *instance* and k the *parameter*. The parameter is usually interpreted as a natural number.

Throughout this thesis, x and k will denote arbitrary instances and parameters of the problems in scope. For convenience we set $|x|$ to twice the number of characters in x.

The seemingly peculiar choice for the length, $|x|$, is motivated by the technical desire to have $|x|^c$ grow unbounded as a function of c, regardless of x. This trait is made possible by our choice because our choice guarantees $|x| > 1$, which even holds regardless of our alphabet. The choice does not impact our results, because we are interested in behavior that is polynomial or logarithmic in

the length of x and as our definition is linear in the number of characters such behavior is retained: it is present in terms of the number of characters precisely if it is present in terms of $|x|$. Thus the main consequence of our choice is that our mathematical expressions can become more elegant than when we would have set $|x|$ to just the number of characters in x. Furthermore, our choice has a technical interpretation. Since we are dealing with the product $\Sigma^+ \times \Sigma^+$, we may need to make x self-delimiting in any practical coding scheme. If x has n characters, this is easily possible in $2n$ characters by prepending $n-1$ '1's followed by a '0' to x.

Definition 12 was originally also used by by Flum and Grohe [17], although for their book on parameterized complexity theory [18] they used an alternative definition. In their case a parameterized problem is a subset of Σ^+ equipped with a parameterization function $\kappa : \Sigma^+ \to \mathbb{N}$ that is computable in polynomial time. The only difference with our definition is that in our case the length of the parameter is not echoed in the length of the instance. This matters in particular when the parameter is not computable from the instance in polynomial time. In such a case, the precise dependency of an algorithm's complexity on the length of the instance can be obscured by the presence of a specification of the parameter in the instance. Thus, for Flum and Grohe parameters are necessarily internal to the problem. By contrast, our definition allows parameters to capture external knowledge.

When dealing with parameterized problems, it is often desirable to look at fixed-parameter values, giving rise to slices of the problem.

Definition 13. The kth *slice* of a parameterized problem A is the problem

$$A_k = \{x \mid (x, k) \in A\}.$$

The class of parameterized problems of which the slices are in **P** is known as slice-wise **P**, or **XP** [14, 18] and comes in several guises.

Definition 14. A parameterized problem A is in the class *nonuniform* **XP** if there are functions $g, e : \Sigma^+ \to \mathbb{N}$ and a family of decision procedures, $\Phi = \{\phi_k\}_{k \in \Sigma^+}$ such that for each k the slice A_k is decided by ϕ_k in time $g(k)|x|^{e(k)}$.

If A is in nonuniform **XP** by a decidable (without resource bounds) family Φ, then it is in *uniform* **XP**. In this case, we use Φ to indicate the computable mapping $k \mapsto \phi_k$. Indeed, an indexed family is decidable precisely when the mapping of index values to their corresponding elements is a computable function.

If A is in uniform **XP** by some g, e, Φ such that g and e too are computable, then it is in *strongly uniform* **XP**. When used without modifiers, **XP** means strongly uniform **XP**.

The freedom in the choice of e in the definition of **XP** makes **XP**-algorithms impractical in general. The proper way to stretch the notion of tractability beyond **P** into the class of fixed-parameter tractable problems, or **FPT** is a further restriction.

Definition 15. When e is taken as a constant in the definitions of the various versions of **XP**, we obtain the definitions for *nonuniform* **FPT**, *uniform* **FPT**, and *strongly uniform* **FPT**.

When used without modifiers, **FPT** means strongly uniform **FPT**.

It may seem as if an even stronger notion of tractability is attained by also restricting the effect of g to be additive instead of multiplicative. It turns out, however, that this would lead to the same class **FPT**. This is a nice robustness property of **FPT**.

Lemma 1. *Let A be a parameterized problem. The following are equivalent.*

1. *A is in* **FPT**.
2. *There is a decision procedure that, for some computable function g and constant c, decides A in time $g(k)|x|^c$.*
3. *There is a decision procedure that, for some computable function g' and constant c', decides A in time $g'(k) + |x|^{c'}$.*

Proof. The equivalence $1 \iff 2$ we get because the decidable families of decision procedures for slices of A and the decision procedures for A are related by uncurrying and currying.

The implication $2 \implies 3$ follows by taking $g(k) = g'(k) + 1$ and $c = c'$, for then we have $g(k)|x|^c \geq g'(k) + |x|^{c'}$. The opposite direction, $2 \impliedby 3$, follows by taking $g'(k) = g(k)^{c+1}$ and $c' = c + 1$. As either $g'(k)$ or $|x|^{c'}$ is bigger than $g(k)|x|^c$, their sum certainly is. □

We remark that the above lemma and proof would be invalid if the empty string, which has length 0, was allowed as an instance. Unfortunately, parameterized problems are commonly defined as subsets of $\Sigma^\star \times \Sigma^\star$ [16, 18] so the empty string often *is* allowed.

A direct consequence of the definition of **FPT** is that **FPT** is a subset of **XP**. To see that it is a strict subset, we turn to the following problem.

Problem (PPAcc)**.** Parameterized polynomial time acceptance.

We define problems by a membership criterion based on an interpretation of the instance and the parameter.

Instance: x.

Parameter: (ϕ, e), where ϕ is a decision procedure and e a natural number.

Criterion: ϕ accepts x in time $|x|^e$.

We claim that this problem is in **XP** but not in **FPT**.

Lemma 2. PPAcc $\in$ **XP**.

Proof. By standard results in simulation via efficient universal Turing machines [3], we know that for some c it is possible to compute $\phi(x)$ from a specification of ϕ and x in time $\mathcal{O}(|x|^{ce})$, where the hidden constant depends on the number of states and tapes used by ϕ. These numbers can be computed from ϕ, hence PPAcc is in strongly uniform **XP**. □

Theorem 3. **FPT** $\subset$ **XP**.

Proof. It suffices to show that PPAcc is not in **FPT**. Suppose for contradiction that it is. Then, for some c there exists a g such that regardless of ϕ the slice $\text{PPAcc}_{(\phi,c+1)}$ is decidable in time $g(\phi, c+1)|x|^c$. However, by the time hierarchy theorem [3], there exists a slice with a decision procedure ϕ that runs in time $|x|^{c+1}$, but is outside $\mathcal{O}(|x|^c)$, contradicting our assumed decidability. □

An eminent problem that is not known to be tractable in the classical sense is the vertex cover problem. The vertex cover problem might not be in **P**, but definitely is in **FPT**, so tractable when parameterized. In fact, the parameterized vertex cover problem is a standard example of a fixed-parameter tractable problem.

Problem (VC)**.** Vertex cover.

Instance: G, a simple graph.

Parameter: k, a natural number.

Criterion: G contains a vertex cover of size at most k.

Membership of **FPT** follows directly from a simple algorithm by Sam Buss [6, 14, 18] which we include here as Algorithm 1. In the algorithm, the time needed by loop 1 is quadratic in $\|V\|$ and the time needed by loop 2 is determined by $\|V'\|$, which is upper bounded by a function of the parameter. Therefore, the algorithm indeed proves that VC is in **FPT**.

Algorithm 1 An **FPT**-algorithm for VC.

Input: A graph (V, E) and a parameter k
Output: The presence of a vertex cover of size k in (V, E)

```
V' ← ∅
k' ← k
for all v in V do                                              // loop 1
  d ← the degree of v
  if d > k then          // v is necessarily in every cover smaller than k
    k' ← k' − 1
  else if d > 0 then                                  // v cannot be ignored
    V' ← V' ∪ {v}
  end if
end for
E' ← E ∩ (V' × V')
if ‖E'‖ ≤ k'k then                  // a cover of size k' might be possible
  for all subsets, W, of V' of size min(k', ‖V'‖) do                // loop 2
    if W is a vertex cover of E' then
      return true
    end if
  end for
end if
return false
```

The two-part character of Algorithm 1 is a common design pattern in practical algorithms. The purpose of the first part, loop 1, is to *preprocess* the input, (V, E). The result of this preprocessing, (V', E'), represents a view of the

input that has been rid of any trivial parts. More generally, it is the goal of preprocessing to identify the intrinsically hard part of the input.

In Algorithm 1, the preprocessing is of a special kind. Because the size of the result of the preprocessing can be upper bounded by a function of the parameter, we have a clean cut in the variables the time needed in both loops depends upon. For the preprocessing, loop 1, the required time is polynomial in the size of the instance, whereas an upper bound on the time required by loop 2 can be determined solely on the basis of the parameter. This kind of preprocessors is called kernelization.

Definition 16. A language map from one parameterized problem to another, mapping (x, k) to (x', k'), is a *kernelization* if there is a constant c, and a computable, non-decreasing function $h : \mathbb{N} \to \mathbb{N}$ such that we have:

1. (x', k') is computable in time $|x|^c$;
2. $|x'| \leq h(k)$ and $k' \leq h(k)$ hold.

We call (x', k') the *kernel* of (x, k). A parameterized problem A is *kernelizable* if there exists a kernelization from A to itself.

In our case, loop 1 of Algorithm 1 constitutes a kernelization from VC to itself, showing that VC is kernelizable.

Usually the running time of the language map in the definition of kernelizations is allowed to depend on the parameter [6, 16, 18], but this is not necessary for the fundamental lemma about kernelizability.

Lemma 4. *Also equivalent to the statements in Lemma 1 is the following.*

4. A is decidable and kernelizable.

Proof. For 3 $\Longleftarrow$ 4, observe that computing the kernel followed by deciding the kernel, the size of which is bounded by a function, h, of the parameter, takes $|x|^c + g(h(k))$ time, where g is a bound on the running time of the decision procedure. As g and h are both computable, their composition is too.

The 2 $\Longrightarrow$ 4 direction is a bit more involved. In case $A = \emptyset$ or $A = \Sigma^+ \times \Sigma^+$ we can get away with a kernelization mapping all inputs to an arbitrary string. In all other cases we can take some $y \in A$ and $n \notin A$. Let ϕ be the assumed decision procedure that runs in time $g(k)|x|^c$. We define a kernelization using ϕ, y and n as follows.

Input: (x, k)
Output: a kernel of size $\max(|y|, |n|, g(k) + |k|)$
  run ϕ on (x, k) for $|x|^{c+1}$ steps
  if ϕ has terminated **then**
    return y or n according to the output of ϕ
  else
    return (x, k)
  end if

By construction, the running time of our algorithm is as required. It remains to show that whenever ϕ did not halt within $|x|^{c+1}$ steps, we have $|x| \leq g(k)$. This is true because ϕ is guaranteed to halt in $g(k)|x|^c$ steps, so in this situation we have $|x|^{c+1} < g(k)|x|^c$ and consequently $|x| < g(k)$ as desired. □

Unfortunately, kernelizations in general are no reductions, so we cannot directly extract a preorder on problems from them. Kernelizations fall short of being reductions in that the identity map is not necessarily admitted. In Section 7 we will see that it is natural for the length of instances to grow unbounded given a parameter value. Therefore, it is natural for the identity map to violate requirement 2 of Definition 16, $|x| \leq h(k)$, for some x. Nevertheless we are interested in a reduction that somehow relates to parameterized problems and in particular to **FPT** and **XP**. The proper notion is that of fpt-reductions.

Definition 17. A language map from one parameterized problem to another, mapping (x, k) to (x', k'), is an *fpt-reduction* if there are constants c, d, and computable, non-decreasing functions $g, h : \mathbb{N} \to \mathbb{N}$, such that we have:

1. (x', k') is computable in time $g(k)|x|^c$;
2. $|x'| \leq |x|^d$ and $k' \leq h(k)$ hold.

The difference between fpt-reductions and kernelizations is subtle but important. Additionally, looking at Definition 14 and Definition 15 it should be obvious that fpt-reductions are the most restrictive of several similar types of reductions, the least restrictive of which being nonuniform xp-reductions. Often, results can be readily rephrased in terms of another reduction, but we will stick to fpt-reductions in this thesis.

The definition of fpt-reduction above is not standard, but it is correct.

Theorem 5. *The language maps that are fpt-reductions are reductions. More specifically, the identity map is an fpt-reduction and the class of fpt-reductions is closed under composition.*

Proof. We will present a proof only of the last statement. Let f_1 and f_2 be fpt-reductions of signatures that admit the composition $f_2 \circ f_1$. We will subscript all auxiliary symbols from the definition according to the map they belong to.

1. The composition $f_2 \circ f_1$ is computable in time $g_1(k)|x|^{c_1} + g_2(k')|x'|^{c_2} \leq g_1(k)|x|^{c_1} + g_2(h_1(k))|x|^{d_1 c_2}$. Hence $f_2 \circ f_1$ is computable in time $g(k)|x|^c$, with $c = c_1 + d_1 c_2$ and $g(k) = g_1(k) + g_2(h_1(k))$.
2. For $f_2(f_1(x, k)) = (x'', k'')$, we find $|x''| \leq |x'|^{d_2} \leq |x|^{d_1 d_2}$ and $k'' \leq h_2(k') \leq h_2(h_1(k))$. Hence $|x''| \leq |x|^d$ and $k'' \leq h(k)$, with $d = d_1 d_2$ and $h = h_2 \circ h_1$.

□

Standard definitions [14, 18] omit the first part of requirement 2 of Definition 17, $|x'| \leq |x|^d$. However, this requirement is essential for Theorem 5. In the classical setting of Karp reductions the bound on the size of the result of the reduction was not necessary because it was implied by the bound on the computation time. With fpt-reductions, though, the computation time may depend on the parameter and the bound on the size of the result of the reduction must be made explicitly.

Just as **P** is the minimal class for Karp reductions in the sense that there are no nonempty, strict subsets of **P** that are closed under Karp reductions, **FPT** is the minimal class for fpt-reductions.

Lemma 6. *Also equivalent to the statements in Lemma 1 and Lemma 4 is the following.*

5. A is fpt-reducible to every nontrivial parameterized problem.

Proof. For 2 $\Longleftarrow$ 5, consider the singleton parameterized problem $\{y\}$ and let f be an fpt-reduction to $\{y\}$, which exists by assumption. We define a decision procedure for A as follows.

Input: (x, k)
Output: **true** or **false**, consistent with $(x, k) \in A$
  if $f(x, k) = y$ **then**
    return **true**
  else
    return **false**
  end if

Computing $f(x, k)$ is possible within the time bound of an acceptable decision procedure, as is testing for equality to the constant y. Hence, this decision procedure satisfies our requirements.

Similarly, for 2 $\Longrightarrow$ 5, let ϕ be the assumed decision procedure that runs in time $g(k)|x|^c$, let B be any nontrivial parameterized problem and take $y \in B$ and $n \notin B$. We define an fpt-reduction from A to B as follows.

Input: (x, k)
Output: a member of B if $(x, k) \in A$ and a non-member otherwise
  if $\phi(x, k)$ **then**
    return y
  else
    return n
  end if

The running time is immediately seen to be acceptable and as the output is of constant maximum size, $\max(|y|, |n|)$, requirement 2 of Definition 17 is also satisfied. □

Corollary 7. FPT *is closed under fpt-reductions.*

Note that **XP** too is closed under fpt-reductions. Because of Theorem 3 and Corollary 7 not every problem in **XP** is complete for **XP** under fpt-reductions. Therefore, it is nice to know problems that are complete for **XP**.

Lemma 8. PPAcc *is complete for* **XP** *under fpt-reductions.*

Proof. Having proved Lemma 2, only hardness of PPAcc under fpt-reductions needs to be proved.

Let A be in **XP** by some g, e, Φ. We claim that the mapping $(x, k) \mapsto (x, (\Phi(k), e(k) + \log g(k)))$ is an fpt-reduction from A to PPAcc. The time needed to compute this mapping depends only linearly on $|x|$, as does the size of the output. By computability of g, e and Φ, both the parameter dependency of the computation time and the size of $(\Phi(k), e(k) + \log g(k))$ can be upper bounded by a computable function of k. Thus all that remains to be shown is that this mapping is indeed a language map. To see that it is, recall that $\Phi(k)$ decides A_k in time $g(k)|x|^{e(k)}$. Because $g(k)|x|^{e(k)} \leq |x|^{e(k)+\log g(k)}$ holds, we

find that (x, k) is in A precisely if $\Phi(k)$ accepts x in time $|x|^{e(k)+\log g(k)}$, which is the defining criterion of PPAcc stated in terms of the output of our mapping. Hence, the mapping is indeed an fpt-reduction. □

A similar proof was given by Flum and Grohe [18] for a different problem, p-Exp-DTM-Halt. However, their definition of fpt-reduction omitted the first part of requirement 2 of Definition 17. Under our improved definition, their proof fails. The situation is made even more difficult because their definition of a parameterized problem forbids a correction of their definition of fpt-reduction. We interpret these difficulties as an indication of the correctness of our definitions and the superiority of PPAcc over p-Exp-DTM-Halt.

We note that the statements of Lemma 1, Lemma 4 and Lemma 6 can be extended even further. Other characterizations of **FPT** include one via advice to oracle machines [1,9,14], one via circuits [13,14,18], and one via model-checking problems for fragments of first-order logic [17]. As these notions will not be explored in this thesis, we refrain from including their details.

Apart from VC, many more problems are known to be in **FPT**. Extensive lists are available in the literature [14, Appendix A] and will not be included here.

5 Fixed-Parameter Space Tractability

Complementary to the analysis of fixed-parameter tractability with respect to running time is the analysis of fixed-parameter tractability with respect to memory usage. We extend our framework with classes for fixed-parameter space complexity and investigate the connection between time motivated and space motivated classes. A discussion of the practical aspects of fixed-parameter space tractability is also included.

Modern day challenges in handling big data, such as when data is too big to fit in active, fast memory, suggest an increasingly important role for the study of space complexity. We feel that classically, space tractability is best captured by the class, **L**, of problems decidable in logarithmic space [3]. In the previous section, we introduced the class **FPT** to allow a more refined analysis of time tractability than was possible through the study of **P**. A similar program is possible for space tractability. Contrary to the development of **FPT**, this program has a rather messy and scattered history. One of the earliest mentions of parameterized space tractability was a remark in the fourth of the founding papers of parameterized complexity theory by Downey and Fellows [1]. Unfortunately, the definition found there was that of slice-wise **L**, which we consider impractical for the same reason we considered **XP** impractical. The same authors were part of a group that later published a proper treatment of parameterized space complexity [9]. However, we feel that their framework of oracle machines taking advice does not properly reflect the very practical significance of fixed-parameter space tractability. In the works of Flum and Grohe [17,18], the class of fixed-parameter space tractable problems appears as **para-L**. While their textbook [18] only makes very brief mention of the class in an exercise, their foundational paper [17] contains some actual results, as well as a definition of an associated reduction. Like in the case of fpt-reductions, though, the definition that is put forward is defective.

We will structure our analysis of parameterized space complexity the same way we built up our understanding of parameterized time complexity in the previous section. The class of parameterized problems of which the slices are in **L** we will call slice-wise **L**, or **XL**. Again, different variants exist.

Definition 18. A parameterized problem A is in the class *nonuniform* **XL** if there are functions $g, e : \Sigma^+ \to \mathbb{N}$ and a family of decision procedures, $\Phi = \{\phi_k\}_{k\in\Sigma^+}$ such that for each k the slice A_k is decided by ϕ_k in space $g(k) + e(k)\log|x|$.

If A is in nonuniform **XL** by a decidable (without resource bounds) family Φ, then it is in *uniform* **XL**. In this case, we use Φ to indicate the computable mapping $k \mapsto \phi_k$.

If A is in uniform **XL** by some g, e, Φ such that g and e too are computable, then it is in *strongly uniform* **XL**. When used without modifiers, **XL** means strongly uniform **XL**.

The class of fixed-parameter space tractable problems, or **FPST** is a restriction of **XL**.

Definition 19. When e is taken as a constant in the definitions of the various versions of **XL**, we obtain the definitions for *nonuniform* **FPST**, *uniform* **FPST**, and *strongly uniform* **FPST**.

When used without modifiers, **FPST** means strongly uniform **FPST**.

The connection between the space motivated classes **XL** and **FPST**, and the time motivated classes **XP** and **FPT** reminds of the familiar [3] connection between **L** and **P**.

Theorem 9. **XL** $\subseteq$ **XP** *and* **FPST** $\subseteq$ **FPT**.

Proof. We will only prove **XL** $\subseteq$ **XP**. The proof of **FPST** $\subseteq$ **FPT** is subsumed and recovered by taking e constant.

A computation on (x, k) that terminates after having used at most $g(k) + e(k)\log|x|$ space can have been in at most

$$\mathcal{O}\Big(2^{g(k)+e(k)\log|x|}\Big) = \mathcal{O}\Big(2^{g(k)}|x|^{e(k)}\Big)$$

different configurations during the computation, where the hidden constant depends on the computation procedure. Indeed, if not, the computation would end up looping forever. Finding the terminating computation through these configurations can be done in linear time [3], hence for every **XL**-algorithm, there exists an **XP** algorithm. In other words, every problem in **XL** is in **XP** as desired. □

Summarizing our knowledge of the strongly uniform parameterized complexity classes we get the diagram in Figure 1. The diagram can be amalgamated with similar lattices for the uniform and nonuniform classes.

Unfortunately, the space hierarchy theorem [3] is not strong enough to separate **FPST** from **XL** [9] in the same way we separated **FPT** from **XP**. We remark that a further strengthening of our parameterized classes might facilitate more powerful diagonalization theorems. Concretely, we propose restricting the strongly uniform classes to those where the problems are strongly uniform

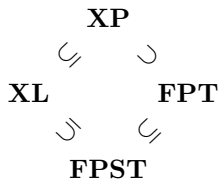

Figure 1: The containment order of selected classes of parameterized complexity.

for some *constructible* g, e. Of course, in the setting of time complexity 'constructible' would be time constructible and in that of space complexity it would be space constructible.

The classical and open question $\mathbf{L} \stackrel{?}{=} \mathbf{P}$ has an immediate equivalent in the realm of parameterized complexity theory, namely whether nonuniform **XL** equals nonuniform **XP**. Indeed, if we have $\mathbf{L} = \mathbf{P}$, then slices of a parameterized problem are in **L** if and only if they are in **P**, and vice versa. It is possible to drop the nonuniformity predicate, because given a **P**-algorithm we can effectively construct a reduction running in logarithmic space to the circuit value problem [3] and if $\mathbf{L} = \mathbf{P}$ holds there exists an **L**-algorithm for the circuit value problem. Indeed, the concatenation of the reduction and the algorithm for the circuit value problem constitutes an effective transformation of arbitrary **P**-algorithms to **L**-algorithms, approving the omission of the nonuniformity predicate. In addition, looking closely at our transformation, we see that if $\mathbf{L} = \mathbf{P}$ holds it is possible to bound the value of the constant coefficient of a logarithmic space complexity based on the exponent in the associated polynomial space complexity, showing that $\mathbf{FPST} \stackrel{?}{=} \mathbf{FPT}$ too is equivalent to $\mathbf{L} \stackrel{?}{=} \mathbf{P}$, as claimed by Flum and Grohe [18].

Cai, Chen, Downey and Fellows state that equality of **L** and **P** is furthermore equivalent to equality of **XL** and **FPT** [9]. However, if this claim holds we would conclude that equality of **L** and **P** implies equality of **FPT** and **XP**, but that is forbidden by Theorem 3. Thus they would have proven $\mathbf{L} \neq \mathbf{P}$. One shortcoming we note in their proof is the assumption that **FPT** is closed under xl-reductions, a type of reduction we will deal with shortly. Such a property of **FPT** is yet to be proven.

Just as we have fpt-reductions and their relatives for the analysis of parameterized time complexity, we have reductions for the analysis of parameterized space complexity too. Also, just as standard definitions of fpt-reduction had a shortcoming, standard definitions of reductions for parameterized space complexity [9, 15, 17] have one.

Definition 20. A language map from one parameterized problem to another, mapping (x, k) to (x', k'), is an *fpst-reduction* if there are constants c, d, and computable, non-decreasing functions $g, h : \mathbb{N} \to \mathbb{N}$, such that we have:

1. each character of (x', k') is uniformly computable in space $g(k) + c \log |x|$, independent of all other characters;
2. $|x'| \leq |x|^d$ and $k' \leq h(k)$ hold.

The shortcoming of standard definitions is once more in requirement 2 of the above reduction, of which the first part, $|x'| \leq |x|^d$, is usually unjustly omitted.

Definition 20 generalizes to related reductions for all of our parameterized space complexity classes and each of these classes is minimal for its corresponding reduction, analogous to what we found in Lemma 6. The somewhat contrived space usage limit is an established [3] way to make these reductions actual reductions. With the more intuitive requirement of computability of (x', k') in its entirety in space $g(k) + c \log |x|$, there might not be enough space to actually hold (x', k'), and an equivalent of Theorem 5 would not be possible.

Theorem 10. *The language maps that are fpst-reductions are reductions.*

Proof. Like before, it suffices to prove only that the class of fpst-reductions is closed under composition. Let f_1 and f_2 be fpst-reductions of signatures that admit the composition $f_2 \circ f_1$. We will subscript all auxiliary symbols from the definition according to the map they belong to.

1. We alter the procedure for computing f_2 so that each time a character of (x', k') influences its computation, it is computed from (x, k) by means of f_1 and an index in (x', k'). By the time bound used in Theorem 9, (x', k') has a length in $\mathcal{O}(2^{g_1(k)}|x|^{c_1})$ so the space needed to keep an index in memory is no more than $g_1(k)+c_1 \log |x|$ plus a constant. Up to a constant that is of no significance, this yields a procedure for computing $f_2 \circ f_1$ in space $2g_1(k) + 2c_1 \log |x| + g_2(k') + c_2 \log |x'| \leq 2g_1(k) + 2c_1 \log |x| + g_2(h_1(k)) + d_1 c_2 \log |x|$. Hence each character of $f_2 \circ f_1$ is computable in space $g(k) + c \log |x|$ with $c = 2c_1 + d_1 c_2$ and $g(k) = 2g_1(k) + g_2(h_1(k))$.

2. Identical to the corresponding part of the proof of Theorem 5.

□

A related characterization of tractability by kernelizations is possible in the case of fixed-parameter space tractability too.

Definition 21. A language map from one parameterized problem to another, mapping (x, k) to (x', k'), is a *space kernelization* if there is a constant c, and a computable, non-decreasing function $h : \mathbb{N} \to \mathbb{N}$ such that we have:

1. each character of (x', k') is uniformly computable in space $c \log |x|$, independent of all other characters;

2. $|x'| \leq h(k)$ and $k' \leq h(k)$ hold.

We call (x', k') the *space kernel* of (x, k). A parameterized problem A is *space kernelizable* if there exists a space kernelization from A to itself.

A more practical version of the first requirement requires (x', k') to be computable by a logarithmic space computation with a write-only means of presenting its output, such as a printer. In the setting of deterministic computation, we can use addressable write-only variables for output. This enables the use of pseudocode in our analysis: we disallow the use of a call stack, and recursion with it, produce the output in designated write-only output variables, and limit the space usage of all other variables. Although not presented in this way

Algorithm 2 A space kernelization for VC. The input consists of a number of vertices, n, and an adjacency matrix, $(X_{i,j})_{0 \leq i < j < n}$, for a simple graph of n vertices. All variables except X', which is only written to, are restricted to using $\mathcal{O}(\log n)$ space. The degree of the ith vertex in accordance with adjacency matrix X is written as $\deg(i : X)$. Conceptually, the algorithm is no different from Algorithm 1.

Input: An undirected adjacency matrix $n, (X_{i,j})_{0 \leq i < j < n}$ and a parameter k
Output: A space kernel $(n', X'), k'$ of $(n, X), k$

$n' \leftarrow 0$
$k' \leftarrow k$ // without loss of generality, we assume $k \leq n$
for all i **in** $\{0, 1, \ldots, n-1\}$ **do** // first, determine n' and k'
  if $\deg(i : X) > k$ **then**
    $k' \leftarrow k' - 1$
  else if $\deg(i : X) > 0$ **then**
    $n' \leftarrow n' + 1$
  end if
end for
if $n' \leq 2k'k$ **then** // a cover of size k' might be possible
  $i' \leftarrow 0$
  $j' \leftarrow 0$
  for all i **in** $\{0, 1, \ldots, n-1\}$ **do**
    if $0 < \deg(i : X) < k$ **then**
      for all j **in** $\{i+1, i+2, \ldots, n-1\}$ **do**
        if $0 < \deg(j : X) < k$ **then**
          $X'_{i',j'} \leftarrow X_{i,j}$ // X' is a write-only variable
          $j' \leftarrow j' + 1$
        end if
      end for
      $i' \leftarrow i' + 1$
    end if
  end for
else // construct a trivial rejecting space kernel
  $n' \leftarrow 1$
  $X'_{0,0} \leftarrow$ **true**
  $k' \leftarrow 0$
end if
return (n', X'), k'

before, space kernelizations are very real. An example is Algorithm 2, which shows that VC is space kernelizable.

Of course, space kernelizability interests us because it is related to space tractability.

Lemma 11. *A parameterized problem is in* **FPST** *if and only if it is decidable and space kernelizable.*

Proof. The proof is the same as that of Lemma 4, except that instead of running ϕ with a time bound of $|x|^{c+1}$, we run ϕ with a space bound of $(c+1)\log|x|$ and abort as soon as ϕ tries to use more space. If ϕ did not terminate, we have $(c+1)\log|x| < g(k) + c\log|x|$ and consequently $|x| < 2^{g(k)}$. The mapping $k \mapsto 2^{g(k)}$ meets the requirements on h in definition 21 of being computable and non-decreasing. □

Note that the bound on the size of the kernel that this proof provides is exponential in g, which is to be expected in light of the proof of Theorem 9. However, the bound is not very tight, as can be seen from our kernelization for VC. Both Algorithm 1 and Algorithm 2 provide a kernel of size $\mathcal{O}(k^4)$. Since the unparameterized version of VC is complete for **NP** under Karp reductions, it is unlikely that the size of a polynomial kernel [6] corresponds to the running time component g of an **FPT**-algorithm, as that would imply $\mathbf{P} = \mathbf{NP}$.

Like kernelizations in the case of time tractability, space kernelizations have a practical interpretation. Consider a data provider connected to a communication channel [11], such as a computer storage device or a database server in a network, that provides structured data, say in the form of graphs. If we ask it a graph with the intention of checking whether it has a vertex cover of some size, the applicability of space kernelization becomes apparent. Without using space kernelization there is no guaranteed limit on the amount of data that needs to be transfered as a result of the query. If, however, we notify the provider of our intention and the provider implements space kernelization, it is sufficient for the provider to yield a space kernel, thus obtaining a guarantee on the amount of data to be transfered. The important observation is that this does not move the computational burden in terms of usage of space resources onto the data provider, because the space kernelization algorithm requires only a logarithmic amount of space and the space kernel can be communicated directly, without storing it on the side of the data provider. In general, this suggests that it is beneficial for data providers to implement a rich query language, enabling specification of intended data usage and its associated opportunities for employing space kernelization. This pattern is of course already in use in many different forms. Computer storage devices are almost always addressable and many database query languages have high expressive power. Thus we can think of space kernelizations as a means of limiting data transfer and we have found a practical application of the analysis of fixed-parameter space tractability.

In the search for parameterized problems that are in **FPST** most effort is put in analyzing parameterized problems of which the unparameterized version is hard in some classical sense. We have already seen that VC is in **FPST**, which is exciting because the unparameterized vertex cover problem is complete for **NP** under Karp reductions, thus classically hard, especially with respect to space usage, as all inclusions in $\mathbf{L} \subseteq \mathbf{P} \subseteq \mathbf{NP}$ are believed to be strict. Unfortunately,

the notion of a parameterized problem being natural is not stable under logarithmic space reductions, thus the inclusion of VC in **FPST** does not directly give us the inclusion in **FPST** of all natural parameterized versions of problems in **NP**. Additionally, Flum and Grohe [17] showed that completeness for **FPST** is not a very interesting property since trivial parameterizations of problems in **L** are already complete for **FPST**. Luckily, though, there are also nontrivial parameterizations that *are* natural, as we have with VC. In the limited research on parameterized space complexity, already quite a few parameterized problems of which the parameter is undoubtedly natural are classified [9, 15]. We will focus here on stretching the notion of parameterizations being natural, adding parameters until we can prove that a problem is in **FPST**.

The longest path problem is **NP**-complete and in **FPT** when given the obvious parameterization consisting of the desired path length [6]. A more generously parameterized version is the following.

Problem $((l, d)$-PATH$)$**.** Longest path.

Instance: G, a directed graph.

Parameter: (l, d).

Criterion: G does not contain a path of length l and the maximum outdegree in G is d.

First, note that it is possible to verify whether d is the maximum outdegree in G in $\mathcal{O}(\log |G|)$ space. Next, any path of length l in a graph G with maximum outdegree d can be specified as a starting vertex plus a list of the l consecutive outbound edges. This specification takes $\mathcal{O}(\log |G| + l \log d)$ space. Conversely, in $\mathcal{O}(\log |G| + l \log d)$ space it is possible to enumerate all potential paths of length l in G. Verifying that no such potential path is present in the graph is the inverse of testing if any such potential path is present in the graph, which consists of the following checks.

1. All specified outbound edges are present in G.
2. The specification specifies an actual path, that is, it does not contain loops.

Given a specification of a path in $\mathcal{O}(l \log d)$ space, the first check can be implemented by a simple graph traversal in $\mathcal{O}(\log |G|)$ additional space, but the second is slightly more involved. A space efficient way of checking whether no loops are specified is by sequentially generating all pairs of vertices along the potential path and check whether no pair is made up of the same vertex twice. This too can be implemented in $\mathcal{O}(\log |G|)$ additional space, hence we kept our space usage within $\mathcal{O}(\log |G| + l \log d)$ and have found an **FPST**-algorithm for (l, d)-PATH.

In a similar fashion, we can find a parameterization of the connectivity problem, which is complete for **NL** [3], with which the connectivity problem ends up in **FPST**.

Problem $((l, d)$-STCON$)$**.** Connectivity.

Instance: (G, s, t), where G is a directed graph containing vertices s and t.

Parameter: (l, d).

Criterion: There is a path in G from s to t of length at most l and the maximum outdegree in G is d.

Arguably a more natural parameterization would be one that excludes the maximum outdegree, d. In that parameterization, the problem has been studied by Elberfeld, Stockhusen and Tantau [15]. Their investigation focussed on the amount of nondeterminism needed by a decision procedure as a function of the parameter and was part of studying a parameterized analogue of **NL**. Our parameterized version, (l, d)-STCon, of the connectivity problem, however, needs no nondeterminism and is in **FPST**. Like before, all paths of length at most l, starting at s can be specified in $\mathcal{O}(l \log d)$ space. Checking that a potential path contains no loops is not necessary for this problem. The only checks that need to be performed are whether the potential path is present in G and whether it ends in t. As these are possible in space $\mathcal{O}(\log |G|)$, it follows that (l, d)-STCon is in **FPST**. Of course, this is comes as no surprise considering our result with (l, d)-Path, but we do feel the indication that the path length and outdegree play a substantial role in the space complexity of the connectivity problem is noteworthy.

The final problem we will discuss here is the circuit value problem, which is **P**-complete [3]. In light of the previous two problems, we think of a circuit as a labeled directed acyclic graph, and our parameterization looks as follows.

Problem ((h, d)-CV)**.** Circuit value.

Instance: (C, x), where C is a circuit and x a suitable input to C.

Parameter: (h, d).

Criterion: C outputs **true** on x, the height of C is h and the maximum fan-out in C is d.

We claim that (h, d)-CV is in **FPST**. Since the height of the circuit equals its maximum path length, it is possible to specify each path through the circuit in $\mathcal{O}(h \log d)$ space, much like we did with the previous two problems. Moreover, we can execute a depth-first traversal of the circuit in $\mathcal{O}(h \log d)$ space and retain partial outputs for all gates along a path within that space bound. Being able to keep track of partial outputs is crucial and hinges on the fact that the operations at the gates of the circuit are associative. Associativity allows us to merge the outputs of two inputs to a gate of arbitrary fan-in into a single partial output. Thus, partial outputs are of constant size and we see that indeed we can compute the output of a circuit C given input x in $\mathcal{O}(\log |C| + h \log d)$ space by doing a depth-first traversal. Hence, (h, d)-CV is in **FPST**.

There is a difference between the role of the parameter d in the first two problems and in the last problem. In (l, d)-Path and (l, d)-STCon the parameter d can be substituted for an alternative parameter d', the maximum *indegree* in G. However, replacing the parameter d in (h, d)-CV by a parameter d' representing the maximum fan-in in C, we run into trouble. In that case, the number of gates in the circuit can be upper bounded by a function of the parameters, $|C| \leq (d')^h$, which, as we will see in Section 7, makes the problem trivially a member of **FPST**. The same complication does not occur when parameterizing the way we did, by the maximum fan-out. This tells us that the space complexity of instances of the circuit value problem relates in a significant way to their

maximum fan-out. This reminds us of a result obtained by the probabilistic method [26], in particular as a consequence of Lovśz local lemma, for the satisfiability problem [21,27]. Namely, without clauses of a formula in conjunctive normal form sharing variables, the formula is satisfiable. Of course, on such formulas the satisfiability problem is easily decided. It appears that this is part of a more general result regarding the linkage between the outdegree in a directed acyclic graph and the complexity of problems defined on them.

6 Two-Part Codes

Descriptive complexity is involved in our study of parameterized complexity. Using some original definitions this involvement opens the door for structural parameterized complexity theory.

What makes a natural problem natural is best left a matter of aesthetics, but for parameterized problems Cai, Chen, Downey and Fellows hint at a property that would make a good requirement [9].

Definition 22. A parameterized problem A is *smooth* if, for the appropriate n and all $k_1, k_2 \in \mathbb{N}^n$, we have:

$$k_1 \leq k_2 \implies A_{k_1} \subseteq A_{k_2},$$

where the first inequality is taken componentwise. A smooth parameterized problem A is said to converge to the problem

$$A_\infty = \bigcup_{k \in \Sigma^+} A_k.$$

The three problems of the previous section, (l,d)-Path, (l,d)-STCon and (h,d)-CV, as well as VC are all smooth. Informally, a parameterized problem, A, being smooth suggests that its slices approximate A_∞ from below. For (l,d)-STCon and (h,d)-CV this is certainly true. Both exhibit convergence to their classical counterparts. For the other two parameterized problems, however, the convergence would be to Σ^+. Cai, Chen, Downey and Fellows name parameterized problems of the former sort *standardized* and observe that all parameterized problems can be turned into standardized ones. This can be done by copying all or part of the parameter into the instance. For example, a standardized version of (l,d)-Path is the following.

Problem. Standardized longest path.

Instance: (G,l), where G is a directed graph.

Parameter: (k,d).

Criterion: G does not contain a path of length l, the maximum outdegree in G is d and we have $l \leq k$.

We remark that the alternative definition of parameterized problems by Flum and Grohe [18] can be thought of as forcing all parameterized problems to be standardized.

The standardized problems that receive the most study are those for which fixed-parameter tractability results are known, whereas classical tractability results for the problems they converge to are not. The parameter of such standardized problems represents a source of complexity. However, there may be different fixed-parameter tractable standardized problems that converge to the same problem that is potentially intractable in the classical sense. For the vertex cover problem, Jansen and Bodlaender [24] have provided a parameterization different from VC such that the parameterized problem is still kernelizable, thus fixed-parameter tractable. Question about the possibility of essentially different sources of complexity embodied by different fixed-parameter tractable standardized problems converging to the same potentially classically intractable problem are not prevalent in current fixed-parameter complexity research.

For all of our smooth parameterized problems there exists an upper bound to the parameter value with which an instance first occurs. These upper bounds only depend on the size of the instance. The existence of such upper bounds in practical applications of parameterized complexity theory is no surprise, because at some point the parameter dependent part of a complexity measure will start to dominate the part dependent on the instance size. Indeed, if this were not the case, the parameter had little to offer anyway.

The approximating characteristics of parameterized problems have been studied more formally in the context of *polynomial time approximation schemes* [14, 18]. Given an optimization problem, there is a strong connection between it having an efficient polynomial time approximation scheme and it being fixed-parameter tractable when parameterized by the potential value of optimal solutions.

Here, we will pursue a development of the informal notion of approximation by slices, independent of more classical approximation schemes. When provided a smooth parameterized problem A and an $x \in \Sigma^+$ that is eventually in a slice of A, we interpret the parameter values with which x is in A as capturing some upper bound on the resource complexity of x in A. Certainly, bigger parameter values grant more of the resource at hand. This view motivates a complexity driven description of the instance x by means of a two-part code. The first part of the code specifies a slice and with it a complexity, the second part specifies an index within that slice.

Definition 23. We denote the index of some $x \in \Sigma^+$ in a problem B by $\mathrm{rank}(x : B) = \|\{y \mid y \in B \wedge y \leq x\}\|$, where the inequality is the inequality induced by the standard encoding of $\mathbb{N}$ in Σ^+. The *parameter complexity* of an instance x with respect to a parameterized problem A is

$$\mathrm{pc}(x : A) = \min\big\{2|k| + |\mathrm{rank}(x : A_k)| \;\big|\; (x, k) \in A\big\}.$$

We take the minimum of the empty set to be ∞.

The constant 2 in front of the specification of the slice is there to be able to tell the two parts of the code apart. If a parameterized problem converges to Σ^+, the parameter complexity with respect to that problem is never ∞.

In light of the minimum description length principle [22], we can think of the parameter complexity as a way of selecting the most representative slice. Then, slices take the role of models and the parameter complexity is a well-understood two-part code. There is also a more practical approach to parameter complexity. Since the parameter complexity is based upon an actual way

of encoding instances, it relates to a compression method whenever the parameterized problem is decidable. For example, in the case of VC, it tells us that we can encode a graph as the size of a vertex cover in the graph followed by an encoding that is only possible for graphs with a vertex cover of the given size. Immediately, we see that for smooth parameterized problems the slice selected by the parameter complexity will always be the first slice containing x. Based on this, we can boost the compression performance by not working with the smooth parameterized problem, but a cleaned up variant.

Definition 24. The *purification* of a smooth parameterized problem A is the problem

$$A^{\wp} = \{(x, k) \mid (x, k) \in A \ \wedge \ (x, k-1) \notin A\}.$$

In the purification of a problem, instances only occur with their lowest parameter value. An important fact about purifications is that if a smooth parameterized problem is in one of our parameterized complexity classes, its purification is too. This is a consequence of the fact that our parameterized complexity classes are closed under finite applications of set-theoretic operations, including complementation.

Now, for VC, the encoding of an instance x associated with $\mathrm{pc}(x : \mathrm{VC}^{\wp})$ consists of the size of a *minimum* vertex cover followed by an encoding that is only possible for graphs with a *minimum* vertex cover of the given size. In situations where vertex covers play a central role, this is a highly efficient encoding scheme. Note, though, that this encoding scheme internalizes all computational difficulty in the instances. Any smooth parameterized problem becomes computationally trivial when defined for inputs following the encoding associated with the problem. For example, given a graph encoded using the size of its minimum vertex cover, it is trivial to determine if it contains a vertex cover of any given size.

There is a relationship between parameter complexity and Kolmogorov complexity, the latter of which we will indicate by $\mathrm{C}^{t,s}$, where t and s represent time and space bounds respectively [26].

Lemma 12. *For every decidable parameterized problem A there exists a constant c and bounds t, s such that for all x we have*

$$\mathrm{C}^{t,s}(x) \leq \mathrm{pc}(x : A) + c.$$

Proof. We may assume $\mathrm{pc}(x : A) < \infty$. Let (k, i) be an encoding of x according to its parameter complexity with respect to A. That is, $|(k, i)| = \mathrm{pc}(x : A)$ and x is the ith element of A_k. By decidability of A, it is possible to generate all members of A_k in order. In particular, we can find the ith element of A_k. As functions of (k, i), the time and space needed for this reconstruction of x depend solely on the time and space needed by a decision procedure for A.

The claim follows, with the additive constant c depending on the above procedure, which in turn depends on A. □

This relationship can be put to use in parameterized problems where we have some upper bounding property of the parameter complexity. This upper bounding property comes in the form of a slice-wise density characterisation that relates to slices being sparse, but is a little more general.

Definition 25. A parameterized problem A is *informative* if there exists a function $f : \Sigma^+ \to \mathbb{N}$ such that for every parameter $k \in \Sigma^+$ and constant $c \in \mathbb{N}$ we have

$$\|(A_k \cap \Sigma^{\leq cf(k)})\| \leq \|\Sigma^{\leq c(f(k)-1)}\|. \tag{1}$$

More specifically, we then say that A is *f-informative*.

Note that for an informative smooth parameterized problem A, the above limit is not required to hold for A_∞. Relating to the works of Sipser, Li and Vitányi [26] we note that a parameterized problem is informative if its slices are meager with a uniform rate of convergence. In any informative parameterized problem, the initial segments of each slice get increasingly empty as they grow in size. This should be thought of as instance sizes outgrowing parameter values. From this perspective, informative parameterized problems make sense because the additional resources provided by a parameter do not scale with the size of the instances and are thus only expected to aid decision for a limited subset of elements. In general, thinking about the distribution on the instance size that a parameter value induces may provide insight, even when a parameterized problem is not informative.

Since finite unions of meager problems are again meager, a parameterized problem is informative if and only if its purification is informative. This is potentially helpful in proving parameterized problems informative. Many parameterized problems are informative and a proof can often be given via a combinatorial argument. We will do so for VC.

Lemma 13. *Under standard encodings of labeled graphs,* VC*, and thus* $\text{VC}^\wp$ *too, is informative.*

Proof. For readability, we will not track all constants, additive or multiplicative, through this proof.

Standard, bijective, encoding by means of an adjacency matrix of labeled simple graphs with n vertices shows that there are $2^{\mathcal{O}(n^2)}$ different such graphs. We want to know how many of these graphs have a vertex cover of size k. An upper bound on this number is provided by looking at an alternative encoding of graphs with n vertices and a vertex cover of size k. These graphs can be encoded by giving k vertices that make up a vertex cover followed by a reduced adjacency matrix of size $k \times n$. A specification of all the edges in this way is possible by definition of a vertex cover. The k vertices of the vertex cover can be specified within length $k \log n$. Every graph of n vertices with a vertex cover of size k thus admits a specification, although not uniquely, of length $\mathcal{O}(kn)$. Proving that (1) holds for VC is now possible by giving a function f so that for all constants c the number of graphs having a vertex cover of size k and a size $cf(k)$ specification under the standard encoding is less than $2^{c(f(k)-1)}$. Ignoring some constants, we may thus set $n = \sqrt{cf(k)} = \sqrt{c}\sqrt{f(k)}$ and by the alternative coding we get an upper bound to the number of graphs of $2^{k\sqrt{c}\sqrt{f(n)}}$. Focussing on the exponents and moving the factors depending on c to one side, this leaves us to find a function f so that, regardless of c, we have

$$k\sqrt{f(n)} \leq \sqrt{c}(f(k) - 1).$$

The function $f(k) = k^2 + 2$ behaves as desired and we may conclude that VC is $\mathcal{O}(k^2)$-informative. □

Informative parameterized problems provide a bridge between classical complexity theory and parameterized complexity theory. We have seen parameterized problems that are in **FPST** and **FPT**, but converging to classical problems in **NP**. If every such parameterized problem is informative, the additional resources granted by the parameter are apparently essential for slice-wise tractability and aid only a limited number of instances. In other words, there are instances of the classical problem requiring arbitrary additional resources, showing that the classical problem is indeed intractable. Of course, this line of reasoning asks for all parameterized problems in some class to be informative, which is a highly nontrivial question. The following general result is a consequence of Lemma 12, but we point out that the lemma requires the parameterized problem to be decidable, so the result is only usable for strongly uniform parameterized complexity classes.

Theorem 14. *For any decidable f-informative parameterized problem A converging to Σ^+, random strings make hard instances. Here, a hard instance is an instance x that only occurs with parameter values k for which $|x|$ is in $\mathcal{O}(|k|f(k))$.*

Proof. Let x be a string that cannot be compressed by more than a given constant. By Lemma 12, we get that this means that there is a constant r such that $|x| - r \leq \mathrm{pc}(x : A)$ holds. Suppose that x is in A_k. We prove that $|x|$ is in $\mathcal{O}(|k|f(k))$ with the hidden constant depending on r. By equation 1, the second part of the two-part code compresses strings of length $cf(k)$ by at least c characters. Since x cannot be compressed by more than r characters and the first part of the two-part code has length $2|k|$, the length of x is upper bounded by $(r + 2|k|)f(k)$. □

By the incompressibility theorem [22,26] there are strings of any length that cannot be compressed by more than a given constant. Thus there are infinitely many hard instances for any informative parameterized problem.

It would be more attractive if we could define hardness of an instance x as x only occurring with parameter values k for which $|x|$ is in $\mathcal{O}(f(k))$. This strengthening of Theorem 14 may well be possible, at least for a particular class of parameterized problems. A good candidate for such a class is the class of smooth parameterized problems. On these problems we can push the envelope of compression efficiency by looking at compressibility with respect to the purification of problems. Nevertheless, the definition used does a fair job capturing hardness. In algorithms, the dependency on the parameter is often of a big enough magnitude for the additional factor $|k|$ to be irrelevant.

The restriction on the convergence of A to Σ^+ in Theorem 14 could be loosened. Indeed, with minor modifications the same proof holds for parameterized problems converging to any exponentially dense problem. As this is not the major value of the theorem, we valued clarity more than generality in this matter.

The inspiration for Theorem 14 is a paper by Buhrman and Orponen [7]. In that paper similar statements are proven based on instance complexity [20, 28], which is upper bounded up to a constant by Kolmogorov complexity. A major difference between their work and ours is that our construction makes no mention of any particular resource or complexity class. Instead, we extract our result from equation 1 which is fundamentally of a parameterized nature

and held by parameterized problems without a fixed reference to any particular complexity class.

7 Arbitrary Problems Made Tractable

When used as a tool to identify hardness inside problems, parameterized complexity theory extends existing research. Much of the existing structural complexity theory has a place in structural parameterized complexity theory and new light is shed on old conjectures.

Given enough parametric leeway, every decidable problem becomes tractable with respect to any resource. For example, if A is a decidable problem, then the parameterized problem $\{(x, |x|) \mid x \in A\}$ is in **FPT** and **FPST** since the time and space needed to decide any instance can be expressed as functions of the time and space needed by a decision procedure for A, which are functions of the parameter exclusively. This yields the following theorem.

Theorem 15. *Any decidable problem is in* **FPT** *and in* **FPST** *when parameterized as A so that there is a finite upper bound on $\max\{|x| \mid x \in A_k\}$ that is computable as a function of k.*

Proof. We will prove the statement for **FPT**. Proving the statement for **FPST** goes in an entirely similar way.

Let ϕ be a decision procedure for the original problem that decides potential members of length at most n in time $t(n)$ and let $n(k)$ be the computable finite upper bound to $\max\{|x| \mid x \in A_k\}$. The following algorithm is a decision procedure for A.

Input: (x, k)
Output: **true** or **false**, consistent with $(x, k) \in A$
  run ϕ on x for $t(n(k))$ steps
  if ϕ has terminated **then**
    return the output of ϕ
  else
    return false
  end if

We claim that this is an **FPT**-algorithm, which is a nontrivial claim since t and n need not be time constructible. The running time of the algorithm has two constituents. The first is the time spent by computing $t(n(k))$, which is expressible as a function of k. The second is the time spent by simulating ϕ for $t(n(k))$ steps, which is also expressible as a function of k. Hence the entire running time is expressible as a function of k. By Lemma 1 we thus get that A is in **FPT** by setting g to the function obtained and taking c equal to zero. □

It is precisely this theorem that made us consider parameterizing circuits by their height and maximum fan-in unacceptable in Section 5. However, given a constant height and maximum fan-out, without restricting the fan-in, there are infinitely many circuits that realize these constant constraints. Thus it is impossible to compute an upper bound to size of a circuit from its height and maximum fan-out.

Theorem 15 pertains to classes of strongly uniform fixed-parameter tractability. A slight relaxation gives a similar result for classes of uniform fixed-parameter tractability.

Theorem 16. *Any decidable problem is in uniform* **FPT** *and in uniform* **FPST** *when parameterized so that all slices of the parameterized problem are finite.*

Proof. Let A be the parameterized problem of which all slices are finite. Define $n(k)$ as $\max\{|x| \mid x \in A_k\}$, which is finite for all values of k. The proof is now identical to the proof of Theorem 15, although n, and consequently g too, is not necessarily computable. The version of Lemma 1 about uniform **FPT**, where g need not be computable, states precisely what one would expect. □

In Theorem 15 and Theorem 16 all slices of the parameterized problem at hand were finite. Complementary versions of the theorems are available when all slices are cofinite, looking at upper bounds for $\max\{|x| \mid x \notin A_k\}$.

For some problems Theorem 16 is as general as it gets, as no infinite subsets of these problems are in **P** or **L**. As a characterization of problems, this property plays a role in classical structural complexity theory [4]. We will include a few relevant definitions here.

Definition 26. An algorithm that outputs either **true**, **false** or $\perp$ is a *partial decision procedure*. A partial decision procedure is said to *decide* the elements on which it outputs either **true** or **false** and to be *inconclusive* otherwise.

A partial decision procedure is *consistent* with a problem B if it correctly decides the elements of B on which it is not inconclusive.

Definition 27. An infinite subset C of Σ^+ is a *complexity core* for a problem B if no partial decision procedure that is consistent with B and runs in polynomial time decides infinitely many elements of C.

Definition 28. A problem B is **P**-*immune* if B is a complexity core for itself, and **P**-*bi-immune* if, additionally, $\Sigma^+ \setminus B$ is a complexity core for B too.

We augment these definitions in a straightforward manner to not only cater for time tractability, but for space tractability as well.

Definition 29. An infinite subset C of Σ^+ is a *space complexity core* for a problem B if no partial decision procedure that is consistent with B and runs in logarithmic space decides infinitely many elements of C.

Definition 30. A problem B is **L**-*immune* if B is a space complexity core for itself, and **L**-*bi-immune* if, additionally, $\Sigma^+ \setminus B$ is a space complexity core for B too.

The decidable problems on which the **FPT** part of Theorem 16 is the best we can do are precisely the **P**-immune problems and for the **FPST** part it are precisely the **L**-immune problems. However, not all problems outside **P** are **P**-immune and likewise for **L**. Thus there is room for a generalization of Theorem 16 in the sense that we can make a statement about the parameterized tractability of certain parameterized problems of which not all slices are finite. For this, we will make use of resource bounded instance complexity [20, 28].

Definition 31. Fix an encoding of algorithms in some formalism [3, 22, 26] and denote the encodings of partial decision procedures consistent with a problem B and running in time t and space s by $\mathrm{M}_B^{t,s}$. The t, s-*bounded instance complexity* of an instance x with respect to a problem B is

$$\mathrm{ic}^{t,s}(x : B) = \min\{|\phi| \mid \phi \in \mathrm{M}_B^{t,s} \wedge \phi(x) \neq \bot\}.$$

When t or s are omitted, they are taken to be ∞.

Our generalization will be about problems parameterized by resource bounded instance complexity. For a problem B, the parameterized problem obtained by parameterizing B by $\mathrm{ic}^{t,s}$ is the problem $\{(x, k) \mid x \in B \wedge \mathrm{ic}^{t,s}(x : B) \leq k\}$. We note that this problem is smooth and converges to B.

Theorem 17. *Let t be any polynomial and s the logarithm of any polynomial. Any decidable problem is in nonuniform* **FPT** *and in nonuniform* **FPST** *when parameterized by* ic^t *and* ic^s *respectively.*

Proof. We will prove the statement for nonuniform **FPST**. Proving the statement for nonuniform **FPT** goes in an entirely similar way.

Let B be a decidable problem and A the result of parameterizing B by ic^s. We define the following sets, dependent on the parameter value k:

$$B_{\mathrm{ic}\leq k} = \{x \mid \mathrm{ic}^s(x : B) \leq k\},$$
$$M^{\leq k} = \mathrm{M}_B^s \cap \Sigma^{\leq k}.$$

With these definitions in place, we find that we have $A_k = B \cap B_{\mathrm{ic}\leq k}$. Moreover,

$$x \in B_{\mathrm{ic}\leq k} \iff \exists \phi \in M^{\leq k} : \phi(x) \neq \bot \tag{2}$$

holds. This motivates the following decision procedure for A.

Input: (x, k)
Output: **true** or **false**, consistent with $(x, k) \in A$
  for all ϕ **in** $M^{\leq k}$ **do** // while clearing all workspace after every iteration
    if $\phi(x) \neq \bot$ **then**
      return the output of ϕ
    end if
  end for
  return false

For a fixed k, this procedure can be implemented by combining all implementations of the procedures in $M^{\leq k}$ so that they are executed sequentially on empty workspace. This is possible because $M^{\leq k}$ is finite, but nonuniform because in general it is impossible to *know* $M^{\leq k}$. The space requirement of this implementation equals s plus an overhead depending on s, required to clear all workspace. Notably, the space needed by our implementation is logarithmic in $|x|$. In the case of nonuniform **FPT**, the running time, t, increases by a multiplicative constant depending on $\|M^{\leq k}\|$.

It remains to show that the output of our procedure is correct. Immediately, by (2) we get that our procedure is correct on $B_{\mathrm{ic}\leq k}$. As we have $A_k = B \cap B_{\mathrm{ic}\leq k}$, the procedure should output **false** outside $B_{\mathrm{ic}\leq k}$, which it does. Hence, for every value of k the above describes a construction of a decision procedure for A_k that runs in logarithmic space, proving that A is in **FPST**. □

For many interesting problems, parameterizing by resource bounded instance complexity yields parameterized problems that have infinite slices. This shows that Theorem 17 is not implied by Theorem 16. As an example of such problems, we look at problems that showcase a certain degree of redundancy.

Definition 32. Let a problem B and a bijective language map, an *isomorphism*, from Σ^+ to $\Sigma^+ \times \Sigma^+$ mapping B to $B \times \Sigma^+$ be given.

The problem is a *p-cylinder* if the isomorphism and its inverse are computable in polynomial time.

The problem is an *l-cylinder* if the isomorphism and its inverse are computable in logarithmic space.

We will denote the composition of an isomorphism f with projection on the first component by f_1. Thus, if f maps x to (y, z), then f_1 maps x to y.

Cylinders are related to paddable problems [2, 4]. Conjectures by Berman and Hartmanis [5] and by Hartmanis [23] state that all **NP**-complete problems are paddable in polynomial time and logarithmic space respectively. Equivalently, these conjectures hold that all **NP**-complete problems are p-cylinders and l-cylinders. In support of these conjectures many natural **NP**-complete problems have been proven to be p-cylinders and l-cylinders. This is relevant because in such problems there are levels of resource bounded instance complexity that are reached infinitely often. For a proof, consider an **NP**-complete problem B that is a p-cylinder or an l-cylinder by an isomorphism f and fix some $y \in B$. A partial decision procedure consistent with B that decides infinitely many elements is now possible: on input x, output **true** if we have $f_1(x) = y$ and output $\perp$ otherwise. Because f_1 maps infinitely many elements in B to y, this shows that there are infinitely many elements in B with a resource bounded instance complexity less than the length of the partial decision procedure just described.

An alternate take on Theorem 17 that internalizes the above proof that p-cylinders and l-cylinders, parameterized by resource bounded instance complexity have infinite slices uses cylinders exclusively. For a problem B that is a cylinder by an isomorphism f we can define the cylinder parameterization of B by f.

Problem. Cylinder parameterization of B by f.

Instance: x.

Parameter: k.

Criterion: We have $x \in B$ and $f_1(x) \leq k$.

This parameterization of B, too, is smooth and converges to B. Since not all problems are cylinders, the following theorem is not as broadly applicable as Theorem 17. On the other hand, the proven tractability is of the strongly uniform kind, adding to the theorems attractiveness.

Theorem 18. *If a decidable problem B is a p-cylinder (l-cylinder) by an isomorphism f, then the cylinder parameterization of B by f is in* **FPT** *(***FPST***).*

Proof. We will prove that the cylinder parameterization of B by f is kernelizable. As in the proof of Lemma 4, we may assume we have an element n outside the cylinder parameterization at our disposal. By the restrictions on f, the following constitutes a kernelization.

Input: (x, k)
Output: a kernel of size $\max\{|n|, |(k,k)|\}$
  if $f_1(x) \leq k$ **then**
    return $(f_1(x), k)$
  else
    return n
  end if

The theorem now follows from Lemma 4 (Lemma 11). □

It is known [2] that p-cylinders exist outside **P** and we have seen that the padding parameterization of such problems has slices that are infinite. It is also known that problems outside **P** have complexity cores [4]. As the intersection of a complexity core with any slice of a fixed-parameter tractable problem has to be finite, we can try to construct a complexity core by selecting finitely many elements from the slices of the purification of a padding parameterization. Let B be a p-cylinder by an isomorphism f and outside **P**. Fixing an arbitrary $z \in \Sigma^+$, consider the subset

$$C_z = \{f^{-1}(y, z) \mid y \in B\}$$

of B. We note that this subset is infinite, yet it contains at most one element of each slice of the purification of the cylinder parameterization of B by f. In line with the proof of Theorem 18 the properties of f ensure that C is complete under Karp reductions for the same complexity classes as B. If the Berman–Hartmanis conjecture [5] holds and B was **NP**-complete, this would mean that C is again a p-cylinder and thus [4] not **P**-immune. In other words, C cannot be a complexity core. Furthermore, as C is again a p-cylinder we can repeat the process to find a strict subset of C which is again **NP**-complete.

This scheme raises some questions. One question it raises is whether something similar is possible with l-cylinders and space complexity cores. For a start, an analogue of the Berman–Hartmanis conjecture for logarithmic space reductions is available [23]. Another question relates to the densities of the consecutive **NP**-complete problems. The density of $B \backslash C$ may be significally greater than that of C, which would tell us something structural about what makes a problem **NP**-complete.

In this context, we note that by Theorem 14 every decidable exponentially dense informative parameterized problem has infinitely many different slices. For problems that are tractable in a parameterized setting this is important, because if such problems have only finitely many different slices, they are tractable in a classical setting as well. To wit, the classical problem would be determined by a finite union of tractable slices of the parameterized problem. This observation can be thought of as a dual to Theorem 15 and Theorem 16. The observation even has a nice corollary when combined with **NP**-complete problems parameterized by resource bounded instance complexity.

Corollary 19. *Let B be an* **NP***-complete problem. We have* $\mathbf{P} = \mathbf{NP}$ *if and only if for some polynomial t the parameterization of B by* ic^t *has only finitely many different slices.*

We have found several ways of associating tractable parameterized problems with arbitrary problems. For problems in **NP** there is another one, which

is noteworthy because it has slices that are potentially infinite, contrary to the parameterized problems of Theorem 15 and Theorem 16. This parameterization goes via VC. Let B be in **NP** and f_B a logarithmic space reduction from B to VC, meaning that for all x we have $x \in B$ if and only if $f_B(x) \in \text{VC}$. The parameterization via which B is in **FPT** and **FPST** that is based on VC is the following.

Problem. VC parameterization of B.

Instance: x.

Parameter: k.

Criterion: $(f_B(x), k) \in \text{VC}$.

Because f_B can be calculated within the desired resource bounds, this parameterization defines a tractable parameterized problem. If f_B is surjective, the slices of this problem are infinite. Like VC, the VC parameterization of B is smooth. Nevertheless, we are not inclined to call it natural for almost any problem in **NP**. In characterizing tractability, we note that although by Lemma 4 and Lemma 11 we know that the VC parameterization of a problem in **NP** is kernelizable and space kernelizable, there is no immediate strong size bound on the kernels. For VC, we found that kernels of a size polynomial in the parameter value were possible. However, we can only straightforwardly propagate this result to the VC parameterization of problems for which there also is a suitable reduction back to VC. As a result, we find that the VC parameterizations of **NP**-complete problems have polynomial kernels [6], but no such result for the VC parameterizations of other problems in **NP**.

8 The Parameter Distribution of VC

We use VC *to showcase how complexity can be captured in parameters. When parameters capture complexity, the distribution of parameters doubles as a distribution of complexity. Facts about the distribution of parameters can be proven both by using parameterized complexity theory and by using classical probability theory.*

The classical vertex cover problem is complete for **NP** under Karp reductions and logarithmic space reductions. On a graph (V, E) a brute force search through all subsets of V can be performed in time $\mathcal{O}(2^{\|V\|})$, with $\|V\|$ being in $\mathcal{O}(\sqrt{|(V,E)|})$ under standard encodings of labeled graphs. As a function of just $|(V, E)|$ this bound on the running time is about as good as it gets for the vertex cover problem. Invoking the parameter in VC, we can give a more concise bound. In Algorithm 1 we used a brute force search only on a subgraph, the kernel, with at most $2k^2$ vertices, where the kernel was established in time $\mathcal{O}(\|V\|^2)$. This yields a total running time in $\mathcal{O}(\|V\|^2 + 2^{2k^2})$. Note that this bound on the running time is polynomial in $|(V, E)|$ when we limit the input to having values of k that are in $\mathcal{O}(\sqrt{\log |(V, E)|})$.

With the added detail that the parameter brings to expressions of running time, there are search techniques other than brute force which become interesting. Particularly relevant to VC is the bounded search tree technique [14, 18], with which it is possible to decide elements of VC in time $\mathcal{O}(2^k\|V\|)$. Here,

already for values of k in $\mathcal{O}(\log |(V,E)|)$ the running time is polynomial in $|(V,E)|$. We remark that under standard encodings of labeled graphs we have that $\mathcal{O}(\log |(V,E)|)$ equals $\mathcal{O}(\log \|V\|)$ and a running time that is polynomial in $|(V,E)|$ is also polynomial in $\|V\|$.

Combining the kernelization of Algorithm 1 with the bounded search tree technique, we acquire a decision procedure for VC that runs in time $\mathcal{O}(\|V\|^2 + 2^k 2k^2)$. More generally, given a kernelization that, for some constant c, computes a kernel $((V',E'),k')$ of $((V,E),k)$ in time $|(V,E)|^c$, combining the kernelization with the bounded search tree technique provides a decision procedure for VC that runs in time $\mathcal{O}(|(V,E)|^c + 2^{k'}\|V'\|)$. This sets a boundary for the size of kernels for VC we might expect. If a logarithmically sized kernel is possible for VC, the above running time is polynomial in $|(V,E)|$ and we would get $\mathbf{P} = \mathbf{NP}$. The reason for this is that for members of $\mathrm{VC}^{\wp}$ the value of the parameter k is at most $\|V\|$, so $\mathcal{O}(\log k)$ is included in $\mathcal{O}(\log \|V\|)$. Of course, as problems in **NP** are all solvable in exponential time a logarithmically sized kernel for a parameterized version of any **NP**-complete problem, where the parameter value is bounded by a polynomial of the instance size, can be used to prove $\mathbf{P} = \mathbf{NP}$. Conversely, logarithmically sized kernels are possible a fortiori for such parameterized problems when we have $\mathbf{P} = \mathbf{NP}$, as in that case constant sized kernels would be possible by deciding the instances already in the kernelization algorithm. Nevertheless, for VC the smallest kernels currently known have a number of vertices that is linearly bounded by the parameter value [24]. A kernel with $k' \leq k$ and $\|V'\| \leq 2k'$ is given by Flum and Grohe [18]. For some constant c, the kernelization corresponding to that kernel provides a decision procedure for VC that runs in time $\mathcal{O}(|(V,E)|^c + 2^k k)$. We note that a kernel where the number of vertices is linearly bounded by the parameter value should not be considered a kernel of linear size, as under standard encodings the size of a graph is quadratic in the number of its vertices.

In Section 4 we introduced kernelizations as reductions of elements to their intrinsically hard part. This suggests a critical change in hardness of instances around the parameter value for which kernelizations becomes compressing. For VC, a kernel with at most $2k$ vertices is smaller than the original instance (V,E) whenever $k < \frac{\|V\|}{2}$. If this kernel is optimal, graphs where a minimum vertex covers over half of the vertices would somehow be significantly harder than graphs with smaller minimum vertex covers. This prediction of a *phase transition* around $k = \frac{\|V\|}{2}$ contrasts with the phase transition predicted, using the same argument, but a different kernelization, by Downey and Fellows [14] around $k = \sqrt{\|V\|}$.

Both our observation that elements of VC having a parameter value in $\mathcal{O}(|(V,E)|)$ can be decided in polynomial time and the search for critical parameter values representing hardness thresholds raise interest in the distribution of parameter values among members of parameterized problems. Even if a parameter does not capture resource complexity very tightly or our understanding of the distribution of parameter values is limited, we could still obtain useful information on the resource usage in decision procedures.

We will investigate the distribution of parameter values among members of VC. Using results from Section 6 we obtain a correlation between $|(V,E)|$ and parameter values.

Theorem 20. *For every constant r, at least a fraction of $1-2^{-r}$ of the members*

of VC *is so that* $|(V, E)|$ *is in* $\mathcal{O}(|k|k^2)$ *with a hidden constant depending linearly on* r *and on nothing else.*

Proof. By a simple counting argument known as the incompressibility theorem, we know that for any fixed length n and constant r at most 2^{n-r} strings of length n can be compressed by at least r bits. As a result, at least a fraction of $1 - 2^{-r}$ of all strings cannot be compressed by more than r bits. Together with Lemma 13 and Theorem 14 this proves the theorem. □

Because $|(V, E)|$ is in $\mathcal{O}(\|V\|^2)$ this theorem shows that for the vast majority of elements of VC there is an almost linear correspondence between $\|V\|$ and k. We will proceed by showing that for random graphs where the possible edges have an independent, identical probability of occurring, so called Erdős–Rényi random graphs, the expected size of a minimum vertex cover is indeed linear in the number of vertices.

Because the size of a minimum vertex cover cannot be greater than the number of vertices, it suffices to prove that the expected size of a minimum vertex cover is at least linear in the number of vertices. To do so, we analyze a function that produces a lower bound to the size of a minimum vertex cover. This function is based on a relationship between the minimum vertex cover problem and the maximum matching problem that is folklore. As an algorithm, this function takes on the form of Algorithm 3. By removing both vertices, but

Algorithm 3 A randomized algorithm providing a lower bound to the size of any vertex cover of a given graph.

Input: A graph (V, E)
Output: A lower bound on the size of a vertex cover of (V, E)

```
M ← 0
while ‖E‖ > 0 do
    select an edge (v1, v2) ∈ E uniformly at random
    remove v1 and v2 from (V, E)
    M ← M + 1          // at least one of v1, v2 is present in any vertex cover
end while
return M
```

accounting for only one in each iteration, the algorithm settles a lower bound that is within a factor 2 of the actual size of a minimum vertex cover. While slightly more optimal results may be obtained by using a heuristic instead of randomness in the selection of edges, the approach of Algorithm 3 turns out to be easier to analyze. The algorithm has a recursive flavor which can be expressed in the equality

$$M((V, E)) = \begin{cases} M((V, E)^-) + 1 & \text{if } \|E\| \neq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $(V, E)^-$ is obtained from (V, E) by removing the endpoints of a randomly selected edge. If (V, E) is a random graph following a certain distribution, we are interested in the distribution of the associated random graph $(V, E)^-$. Let $G(n, p)$ denote the probability distribution of random graphs with n vertices where every potential edge has an independent probability p of occurring. This is the Erdős–Rényi model.

Lemma 21. *If a random graph (V, E) has the distribution $G(n, p)$, then, for some p^-, the associated random graph $(V, E)^-$ has the distribution $G(n-2, p^-)$.*

Proof. First, observe that $(V, E)^-$ is obtained by removing two vertices from (V, E), thus if (V, E) had n vertices, $(V, E)^-$ indeed has $n - 2$. Second, as the edge is selected uniformly at random, all potential edges have the same a priori probability $p/\binom{n}{2}$ of getting selected. Consequently, the posterior probability p^- of any potential edge occurring in $(V, E)^-$ is independent of and equal to that of all other potential edges in $(V, E)^-$. □

The probability p^- equals the probability of an edge occurring anywhere other than where we selected an edge, provided the selection of an edge was possible. In other words, we can calculate p^- as the fraction of $\binom{n}{2} - 1$ that is expected to contain an edge:

$$\begin{aligned} p^- &= \frac{\mathbb{E}\big(\|E\| - 1 \;\big|\; \|E\| \neq 0\big)}{\binom{n}{2} - 1} \\ &= \frac{\mathbb{E}\big(\|E\|\big)/\mathbb{P}\big(\|E\| \neq 0\big) - 1}{\binom{n}{2} - 1} \\ &= \frac{p\binom{n}{2}/\mathbb{P}\big(\|E\| \neq 0\big) - 1}{\binom{n}{2} - 1}. \end{aligned} \tag{4}$$

Note that p^- is defined whenever we have $n > 2$ and $\mathbb{P}\big(\|E\| \neq 0\big) \neq 0$. In case n equals 2, the random graph $(V, E)^-$ does not have any vertices, so the value of p^- is irrelevant to us. Additionally, note that p^- is 0 only if p is 0.

Based on (3) we find an expression for the expected value of M for a random graph (V, E),

$$\mathbb{E}\big(M((V, E))\big) = \mathbb{P}\big(\|E\| \neq 0\big)\left(1 + \mathbb{E}\big(M((V, E)^-) \mid \|E\| \neq 0\big)\right).$$

In Lemma 21 we saw that if, for some number n and probability p, a random graph (V, E) has the distribution $G(n, p)$, the random graph $(V, E)^-$ has the distribution $G(n - 2, p^-)$. This shows that the conditioning on $\|E\| \neq 0$ does not affect the distribution of $(V, E)^-$.

Definition 33. To aid readability we will use the following notation, where (V, E) is a random graph having the distribution $G(n, p)$.

$$\begin{aligned} E_n(p) &= \mathbb{E}\big(M((V, E))\big) \\ P_n(p) &= \mathbb{P}\big(\|E\| \neq 0\big) \end{aligned}$$

With these definitions we thus have

$$E_n(p) = P_n(p)\left(1 + E_{n-2}(p^-)\right),$$

and moreover, by (4), we have

$$E_n(p) = P_n(p)\left(1 + E_{n-2}\left(\frac{p\binom{n}{2}/P_n(p) - 1}{\binom{n}{2} - 1}\right)\right).$$

Dropping the dependency on p for a moment, we read off the equation $E_n = P_n + P_nP_{n-2} + P_nP_{n-2}P_{n-4} + \ldots$, where we have $P_0 = P_1 = 0$. This already suggests

an efficient way to compute the expected value by an iterative computation, at iteration i computing the value of P_{n-2i}. For this we use the equation

$$P_n(p) = 1 - (1-p)^{\binom{n}{2}}.$$

An algorithm computing $E_n(p)$ in this way is Algorithm 4.

Algorithm 4 An algorithm computing the expected value of the output of Algorithm 3 when the input to that algorithm is distributed according to the Erdős–Rényi model.

Input: Erdős–Rényi random graph parameters (n, p)
Output: $E_n(p)$
  if $n < 2$ **or** $p = 0$ **then**
    return 0
  end if
  $P \leftarrow 1$
  $E \leftarrow 0$
  loop
    $P \leftarrow P\left(1 - (1-p)^{\binom{n}{2}}\right)$
    $E \leftarrow E + P$
    if $n < 4$ **then**
      return E
    end if
    $p \leftarrow \frac{p\binom{n}{2}/P_n(p) - 1}{\binom{n}{2} - 1}$
    $n \leftarrow n - 2$
  end loop

Throughout the iterations of the algorithm, the value of p changes monotonically non-increasing. We will construct a lower bound to $E_n(p)$ by letting p decrease a bit faster. To this end, consider $q = 1 - p$ and $q^- = 1 - p^-$. From (4) we find

$$\begin{aligned} q^- &= 1 - \frac{(1-q)\binom{n}{2}/P_n(1-q) - 1}{\binom{n}{2} - 1} \\ &= \frac{\binom{n}{2}}{\binom{n}{2} - 1}\left(1 - \frac{1-q}{P_n(1-q)}\right) \\ &= \frac{\binom{n}{2}}{\binom{n}{2} - 1}\frac{q - q^{\binom{n}{2}}}{1 - q^{\binom{n}{2}}} \\ &= q\frac{\binom{n}{2}}{\binom{n}{2} - 1}\frac{1 - q^{\binom{n}{2}-1}}{1 - q^{\binom{n}{2}}}, \end{aligned} \tag{5}$$

which means that q^- can be obtained by multiplying q with some factor. Observe that the $\left(1 - q^{\binom{n}{2}-1}\right)/\left(1 - q^{\binom{n}{2}}\right)$ factor in (5) is always less than 1 and serves as a means to make sure q^- is less than 1, thus a proper probability. By dropping the factor and terminating Algorithm 4 whenever q becomes 1 or higher, we get an approximation, $\hat{E}_n(p)$, from below of $E_n(p)$. The advantage of dropping the factor is that we can now readily calculate the value of q, and

thus of p too, at iteration i. It is

$$q\prod_{j=0}^{i-1}\frac{\binom{n-2j}{2}}{\binom{n-2j}{2}-1} = q\prod_{j=0}^{i-1}\frac{(n-2j-1)(n-2j)}{(n-2j-2)(n-2j+1)}$$
$$= q\frac{(n-2i+1)\ n}{(n-2i)(n+1)}.$$

A convenient formula, from which we can get the iteration i at which the value of q equals 1. That is, at iteration

$$\frac{n+1}{2}\frac{(1-q)n}{(1-q)n+1} = \frac{n+1}{2}\frac{pn}{pn+1} \tag{6}$$

the value of q will become 1 and we should stop our approximation. This motivates the formula

$$\hat{E}_n(p) = \sum_{i=1}^{\lceil\frac{n+1}{2}\frac{pn}{pn+1}\rceil-1}\prod_{j=0}^{i} P_{n-2j}\left(1-(1-p)\frac{(n-2j+1)\ n}{(n-2j)(n+1)}\right)$$
$$= \sum_{i=1}^{\lceil\frac{n+1}{2}\frac{pn}{pn+1}\rceil-1}\prod_{j=0}^{i} 1-\left((1-p)\frac{(n-2j+1)\ n}{(n-2j)(n+1)}\right)^{\binom{n-2j}{2}}. \tag{7}$$

With this formula, we can prove a desirable property of $E_n(p)$.

Lemma 22. *Given a probability p, for all sufficiently large n we have*

$$E_n(p) \geq \frac{n}{4}.$$

Proof. Consider choosing n large enough so that we have $p \geq \frac{1}{n+2}$. For such n we have $E_n(p) \geq \hat{E}_n\left(\frac{1}{n+2}\right)$. From (7) we get

$$\hat{E}_n\left(\frac{1}{n+2}\right) = \sum_{i=1}^{\lceil\frac{n}{4}\rceil-1}\prod_{j=0}^{i} 1-\left(\frac{(n-2j+1)n}{(n-2j)(n+2)}\right)^{\binom{n-2j}{2}}.$$

The quadratic growth of the exponent makes the last term converge to 0 as $n-2j$ gets larger, which means the factors of the product all converge to 1 in n. Therefore, the entire expression converges to $\frac{n}{4}$ in n. As Figure 2 illustrates, this convergence is in fact fast. Additionally the figure shows that with a slightly larger value of n, namely so that we have $p \geq \frac{1}{n}$, for $p \leq \frac{1}{12}$ the value of $E_n(p)$ is in fact over $\frac{n}{4}$. Of course, for a fixed n, increasing p will only increase the value of $E_n(p)$. Thus with $n \geq \max\{12, \frac{1}{p}\}$ we have $E_n(p) \geq \frac{n}{4}$, which proves the lemma.

□

Given the reasonably fast convergence of factors in (7) to 1 for a fixed p, we find that a convenient estimate of $E_n(p)$ is already given by (6).

In relation to the vertex cover problem, Lemma 22 leads to the following theorem.

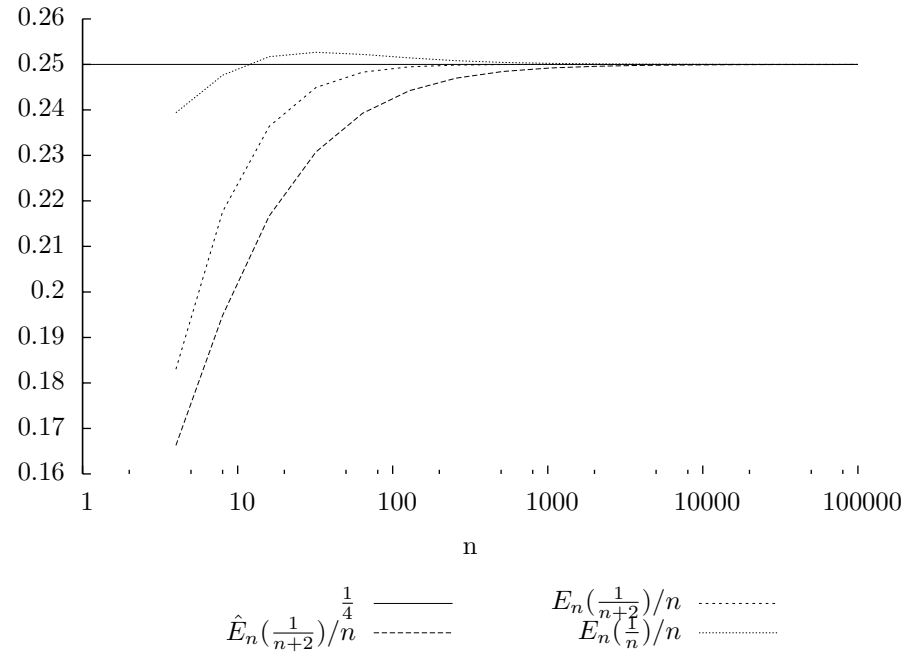

Figure 2: The behavior of $E_n(p)$ and $\hat{E}_n(p)$ as a function of n when p is the reciprocal of n.

Theorem 23. *The expected size of a minimum vertex cover of an Erdős–Rényi (n, p) random graph is at least $\frac{n}{4}$, provided $np \geq 1$ and $n \geq 12$.*

Proof. We constructed Algorithm 3 as a lower bound to the size of a minimum vertex cover. Accordingly, $E_n(p)$ is a lower bound to the expected size of a minimum vertex cover of a random graph that has the distribution $G(n, p)$. By Lemma 22, under the provisions of this theorem the value of $E_n(p)$ is in turn lower bounded by $\frac{n}{4}$. □

Relating this theorem to Theorem 20, we might rephrase this one as saying that for VC we expect $\|V\|$ to be in $\mathcal{O}(k)$. Both theorems state that typical sets [11] of graphs are hard in the sense that there is a function upper bounding the number of vertices of a graph in a typical set from the size of its minimal vertex cover. The difference between the theorems is that the typical sets in Theorem 20 refer to the universal distribution [26], whereas the typical sets in Theorem 23 refer to the Erdős–Rényi distribution.

Closing this section, we present two corollaries of Theorem 23 that demonstrate why we consider graphs (V, E) where the size k of a minimum vertex cover is so that $\|V\|$ is in $\mathcal{O}(k)$ hard.

Corollary 24. *Under the Erdős–Rényi distribution of graphs, the expected running time of Algorithm 1 is exponential in the length of the graph on its input.*

Corollary 25. *Under the Erdős–Rényi distribution of graphs, the expected size of a kernel produced by Algorithm 2 is polynomial in the length of the graph on*

its input. The corresponding **FPST***-algorithm thus has an expected space usage that is polynomial in the length of the graph on its input.*

9 Autoreducibility and XP

Information about complexity can often be found as properties of reductions. We will survey some approaches to classical autoreducibility questions in the context of fpt-reductions.

The study of autoreducibility in complexity theory was motivated by an interest in the internal manifestation of complexity in complete problems [8]. We have seen that parameters can be used to represent some of the complexity of instances. Therefore, we expect connections between the study of autoreducibility and that of parameterized complexity. Our focus will be on a flavor of autoreducibility suited to the setting of fpt-reductions.

Definition 34. A parameterized problem A is *autoreducible* if there is an fpt-reduction f from A to itself such that, for all (x, k), we have $f(x, k) \neq (x, k)$.

Historically, autoreducibility found its way to complexity theory via recursion theory. Complexity theory can, at least in part, be thought of as a resource bounded counterpart of recursion theory. With our focus set on reductions, the most natural approach to recursion theory for us to turn to is one via category theory [10, 12]. In the category theory formalization of recursion theory, computable functions take center stage. This is possible because computable functions satisfy the necessary constraints to maps in the setting of category theory, namely:

- the identity map is a computable function;
- computable functions are composable;
- the composition of computable functions is associative.

Note that the same constraints are satisfied by fpt-reductions: the first two because we demanded them of reductions to begin with, the last one because composition of fpt-reductions is identical to that of computable functions in general. Thus parameterized problems and fpt-reductions together have the structure of a category.

Central to recursion theory is the existence of a universal computable function. In the context of this section, the most convenient definition representing universality of the appropriate kind is by means of a Turing problem, which is a slight variation on what is known as a Turing object in the category theory formalization of recursion theory.

Definition 35. A problem A is a *Turing problem* for a problem B with respect to a class of maps $\mathcal{C}$ if there is a map u in $\mathcal{C}(A \times A, B)$ such that for every map f in $\mathcal{C}(A, B)$ there exists an $a_f \in A$ so that, for all x, we have $f(x) = u(a_f, x)$.

We call u *universal* and a_f the *index* of f for u.

Apart from universality, this definition embodies the main ingredient of proofs by diagonalization in their most abstract form as shown by Lawvere [25]. In our setting, his fixed point theorem takes on the following form.

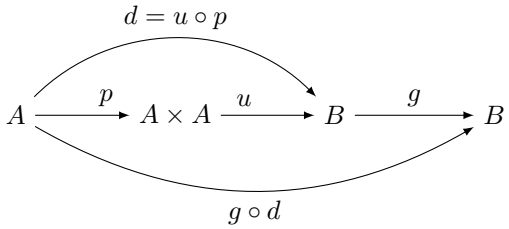

Figure 3: A commutative diagram depicting the relations between the maps involved in the proof of Theorem 26.

Theorem 26. *If A is a Turing problem for a problem B with respect to a class of maps $\mathcal{C}$, then for every g in $\mathcal{C}(B, B)$ there exists an x for which we have $x = g(x)$.*

Proof. Let p be the map $x \mapsto (x, x)$ and d, for diagonal, the composition of p and u as depicted in Figure 3. Because A is a Turing problem, the map $g \circ d$ has an index $a_{g \circ d}$ for u. Using that index we have

$$d(a_{g \circ d}) = u(p(a_{g \circ d})) = u(a_{g \circ d}, a_{g \circ d}) = g(d(a_{g \circ d})),$$

showing that setting $x = d(a_{g \circ d})$ gives us $x = g(x)$ as desired. □

As a result of the existence of a Turing problem with respect to the class of computable functions, every computable function has a fixed point. Unfortunately, this fixed point is often the outcome of a divergent computation. In other words this fixed point is potentially the point 'undefined'. However, in recursion theory computable functions have infinitely many different indices for any universal map and an index always can be chosen different from 'undefined'. This fact can be used to prove that for a Turing problem A, every map f from A to itself has a fixed point in the sense that, for all x, y, we have $u(f(x), y) = u(x, y)$, which is known as Kleene's second recursion theorem [12].

Moving from recursion theory to parameterized complexity theory, from computable functions to fpt-reductions, we get the following corollary of Theorem 26.

Corollary 27. *If there is a Turing problem for a parameterized problem B with respect to fpt-reductions, then B is not autoreducible.*

In fact, the proof of Theorem 26 provides us a construction of a fixed point of an arbitrary fpt-reduction from B to itself. The price paid for the constructible nature of the corollary is that its converse is not necessarily true. Thus, we set out to find a Turing problem with respect to fpt-reductions.

In the proof of Theorem 3 we used a diagonalization technique to separate **XP** from **FPT**. As **XP** itself thus withstands a certain degree of diagonalization, we consider **XP** a good candidate for harbouring a Turing problem. Looking at PPAcc, which is complete for **XP** under fpt-reductions, we find that **P** and **FPT** are, in a way, embedded. By definition of **XP**, the slices of PPAcc are all in **P**. Remarkably, though, the converse is also true: every problem in **P** is a slice in PPAcc. Indeed, if ϕ is a decision procedure for some problem in **P** and it

decides any x in time $|x|^e$, then that problem forms the slice $\text{PPA}_{\text{CC}(\phi,e)}$. Being closed under ordinary set-theoretic operations, the slices of PPACC thus form the field of sets known as **P**. Moreover, the decidable families of slices of PPACC with an upper bound to the value of the second component of the parameter are precisely the parameterized problems in **FPT**. This nicely reiterates the separation of **XP** from **FPT** by a diagonalization argument.

Despite these ways in which PPACC represents **P** and **FPT**, it is not a Turing problem. This can be shown similarly to the proof of Lemma 8, which stated that PPACC is complete for **XP** under fpt-reductions.

Lemma 28. PPACC *is autoreducible.*

Proof. Let PPACC be in **XP** by some g, e, Φ, which exist by Lemma 2. As seen in the proof of Lemma 8, the mapping $(x, k) \mapsto (x, (\Phi(k), e(k) + \log g(k)))$ is an fpt-reduction from PPACC to itself. By stretching the second component of the parameter, our mapping remains an fpt-reduction and we can force it to not have a fixed point:

$$(x, k) \mapsto (x, (\Phi(k), k + e(k) + \log g(k))).$$

This mapping is an fpt-reduction from PPACC to itself such that no (x, k) is mapped to itself, thus PPACC is autoreducible. □

Combined with Corollary 27 we get that PPACC admits no Turing problem, and in particular we get the following.

Corollary 29. PPACC *is not a Turing problem for itself with respect to fpt-reductions.*

Still, there are noteworthy structural properties of PPACC that might enable some form of diagonalization. For instance, PPACC exhibits smooth-like behavior, as for all ϕ, e_1, e_2 we have

$$e_1 \leq e_2 \implies \text{PPA}_{\text{CC}(\phi,e_1)} \subseteq \text{PPA}_{\text{CC}(\phi,e_2)}.$$

This is reflected in the observation that some form of index, in the sense of the definition of Turing problem, exists for functions f that run in polynomial time. To wit, for all ϕ and sufficiently large e we have

$$(x, (\phi \circ f, e)) \in \text{PPACC} \iff (f(x), (\phi, e)) \in \text{PPACC}.$$

Inspired by these observations we consider different notions of autoreducibility. The fpt-reduction of Lemma 28 only acts on the parameter, so we consider fixed instances, that is, if an fpt-reduction maps (x, k) to (x', k'), then (x, k) is a fixed instance if $x = x'$. Not all fpt-reductions from PPACC to itself have fixed instances. To see that this is the case, let m map 1 to 1 and all other inputs x to $x-1$, and consider the mapping $(x, (\phi, e)) \mapsto (x+1, (\phi \circ m, e+1))$. This mapping is an fpt-reduction without fixed instances. To be precise, we might need to add a time counter to ϕ to make sure the mapping does not allow it more computation time. All fpt-reductions in the subclass of fpt-reductions consisting of those reductions that do not alter the parameter value do have fixed instances. One just needs to consider a slice with only one member, which necessarily has

to be mapped to itself. Reductions in this subclass are in a way uniform autoreductions on all problems in **P** at once, and we feel that this is so restrictive that our result is of little value. The proper balance might be struck by the following weakening of the notion of autoreducibility.

Definition 36. A parameterized problem A is *weakly autoreducible* if there is an fpt-reduction f from A to itself such that, for all (x, k), we have $f(x, k) = (x', k')$ with $x' \neq x$ and $k' \leq k$.

In this definition, the order on the parameter might be specific for a parameterized problem. For PPAcc, natural candidates are the componentwise order and, considering the smooth-like property of PPAcc, the order determined solely by the first component, ϕ. We do not know whether PPAcc is weakly autoreducible.

To conclude this section, we remark that the results of this section spawn interest in the isomorphism problem for **XP**-complete problems. If all parameterized problems that are complete for **XP** under fpt-reductions are isomorphic by some fpt-reduction, all are autoreducible and none would be a Turing problem for any other.

10 Open Problems

We point out some possible lines of future research in structural parameterized complexity and state some conjectures. We make no statement about the difficulty of proving any of our conjectures.

Continuing where we finished the previous section, we state a conjecture in the same fashion as the Berman–Hartmanis conjecture. The observations on the relation of PPAcc to **FPT** in Section 9 and the structure of the proof of its completeness for **XP** with respect to fpt-reductions, Lemma 8, lead us to consider the following.

Conjecture 1. *Between every two problems that are complete for* **XP** *under fpt-reductions there exists a bijective fpt-reduction of which the inverse is an fpt-reduction too. In other words, all* **XP***-complete problems are fpt-isomorphic.*

If true, this conjecture pulls any potential treatment of complexity in terms of category theory via Turing problems into complexity classes more intractable than **XP**. We remark that if we would allow so much intractability that we no longer require any resource bounds for computations, we end up in a setting that is more akin to recursion theory than to complexity theory. Indeed, the Curry–Howard–Lambek correspondence already suggests a study of recursion theory and proof theory by means of Turing problems. In this regard, we propose a parameterized treatment of recursion theory and proof theory. For some consequence relation, $\vdash$, such as provability according to some proof system, the set of derivable formulae and their derivations, say $\{(\alpha, k) \mid k \vdash \alpha\}$, is in fact a parameterized problem. Moreover, PPAcc is precisely a problem of this kind and we see that different consequence relations may lead to problems that are complete for different complexity classes. Although PPAcc is not a Turing problem, there might very well be another problem of this kind that is a Turing problem. This parameterized road to recursion theory is hardly

possible under the alternative definition of parameterized problems by Flum and Grohe [18], as for almost every interesting consequence relation, derivations are not computable from consequents. Such is the case also in PPAcc, where the parameter cannot be considered a property of the instance.

In light of the computability of parameter values from instances, we note a track of parameterized complexity that appears to have not been explored yet, and which is related to one proposed by Flum and Grohe. Flum and Grohe [18] mention that fixed-parameter tractability might be too liberal as a definition of tractability and that there might be a need for what they call bounded fixed-parameter tractability. This variant of fixed-parameter tractability places limits on the growth rate of the parameter dependent g in the definition of **FPT**. Orthogonal to adding constraints to **FPT** this way is the possibility of limiting the resources available for the computation of parameter values from instance values. In this thesis we have not placed any such limits and in fact allowed parameter values that were not at all computable from instance values. Especially when it comes to strengthening Theorem 14, additional constraints of this sort might prove useful.

In Section 5 the question $\mathbf{L} \stackrel{?}{=} \mathbf{P}$ was related to inclusions of classes of parameterized complexity. We found that every inclusion relation possible between **FPT** and **XL** except $\mathbf{FPT} \subset \mathbf{XL}$ is strong enough to prove $\mathbf{L} \neq \mathbf{P}$ and from the inclusion $\mathbf{FPT} \subset \mathbf{XL}$ it is not possible to answer the question $\mathbf{L} \stackrel{?}{=} \mathbf{P}$. Thus, the following is weaker than to conjecture $\mathbf{L} = \mathbf{P}$, which is not commonly believed to hold.

Conjecture 2. $\mathbf{FPT} \subseteq \mathbf{XL}$.

Apart from the admission of equality, this is the strongest proposition possible about the inclusion relation between **FPT** and **XP** that does not settle $\mathbf{L} \stackrel{?}{=} \mathbf{P}$. The conjecture would simplify Figure 1 to $\mathbf{FPST} \subseteq \mathbf{FPT} \subseteq \mathbf{XL} \subseteq \mathbf{XP}$, with the remark that the inclusion of **FPT** in **XP** is known to be strict.

Looking more on the level of problems than on that of complexity classes, parameterized complexity theory might be used to distinguish sources of complexity. We wonder if it is possible for a problem to have multiple sources of complexity, each not strong enough to make the problem intractable by itself. Parameterized complexity offers a way to formalize this. We propose to say that an **NP**-complete problem has two sources of complexity if there are smooth parameterized problems A and B in **FPT**, both converging to the **NP**-complete problem, such that, for all x, neither is $\mathrm{pc}(x : A)$ in $\mathcal{O}(\mathrm{pc}(x : B))$, nor is $\mathrm{pc}(x : B)$ in $\mathcal{O}(\mathrm{pc}(x : A))$. We do not know whether an **NP**-complete problem exists that has two sources of complexity according to this definition.

The behavior of the parameter complexity might serve as an even more general indicator of complexity. Although Lemma 4 and Lemma 11 show that being kernelizable and being fixed-parameter tractable is the same, we know of no theorems relating fixed-parameter tractability to the size of kernels. Presumably, the notion of complexity that arises by considering the sizes of kernels is a different notion of complexity than plain fixed-parameter complexity. Unfortunately, a size-bound on kernels need not indicate anything interesting. If a parameterized problem A has kernels of exponential size, then $A' = \{(x, 2^k) \mid (x, k) \in A\}$ has kernels of polynomial size. Intuitively though, the parameterized problems are very similar and the two parameterizations do not capture different sources

of complexity. Nevertheless, the difference between A and A' is reflected in the parameter complexity with respect to each of the problems, motivating the following.

Conjecture 3. *Let A and B be parameterized problems. If, as a function of instances x, we have that $\mathrm{pc}(x : A)$ is in $\mathcal{O}(\mathrm{pc}(x : B))$ and B has kernels of polynomial size, then A has kernels of polynomial size.*

We note that by Lemma 12, considerations such as the above conjecture indicate the applicability of methods from descriptive complexity theory to computational complexity theory. As somewhat of a converse, parameterized complexity theory may be of use in areas related to descriptive complexity theory. This form of reciprocity can be found when comparing the probabilistic method [26], the incompressibility method [26] and Theorem 14. All three are incarnations of the same argument.

The probabilistic method and the incompressibility method have been used in proving Lovász local lemma for the satisfiability problem [21, 27]. We expect Lovász local lemma to have a manifestation in parameterized complexity as well. For the satisfiability problem, Lovász local lemma holds that there is a critical value of the ratio between the number of clauses and the number of variables in a formula in conjunctive normal form separating hard instances from easy instances. For the study of such phase transitions, parameterized complexity might be well suited [14]. This suggests a parameterization of the satisfiability problem where the parameter is related to the aforementioned ratio. However Lovász local lemma will then be recovered for this parameterized problem, the result will likely be applicable to a more general class of parameterized problems. As mentioned at the end of Section 5, a statement on the importance of the outdegree for the complexity of problems defined on directed acyclic graphs is a result we should look out for.

11 Conclusion

We have interpreted parameters in parameterized problems as a representation of the complexity of instances of those problems. This allowed us to look at the distribution of complexity inside problems. In particular, we found that randomness of specification and computational hardness are related for instances of parameterized problems that are, as we called it, informative. The effectiveness of our method was demonstrated by comparing our result applied to the vertex cover problem with a similar result obtained in a more traditional fashion.

At various points, structural properties of problems in **NP** were shown to be related to structural properties of associated problems in **FPT**. Also, by extending fixed-parameter tractability from the time domain to the space domain, we could relate the question $\mathbf{L} \stackrel{?}{=} \mathbf{P}$ to the relation between the parameterized complexity classes **FPT** and **XL**. This indicates that structural parameterized complexity may be a viable framework for proving separation results, not only for parameterized complexity classes.

Acknowledgement

Producing this thesis was made possible to a great extend by the availability of an oracle in the form of my supervisor, Leen Torenvliet. He is a very special kind of oracle. For one thing he answered many questions before I could think of asking them. What I am most grateful for, though, is everything he taught me about the practice of research and the workings of academia. Not only is he a wise oracle, he is also a great mentor.

I am thankful to Harry Buhrman for stimulating discussions that have led to Section 7. This thesis also benefited from comments from Peter Spreij on an early version of Section 8.

Lastly, I want to thank my grandmother for showing her support through financing some of my books.

References

[1] Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. *Annals of Pure and Applied Logic*, 73(3):235–276, 1995.

[2] Eric W. Allender. Isomorphisms and 1-L reductions. *Journal of Computer and System Sciences*, 36(3):336–350, 1988.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, first edition, 2009.

[4] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity II*. Springer, 1990.

[5] Leonard Berman and Juris Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[6] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[7] Harry Buhrman and Pekka Orponen. Random strings make hard instances. *Journal of Computer and System Sciences*, 53(2):261–266, 1996.

[8] Harry Buhrman and Leen Torenvliet. On the structure of complete sets. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 118–133, 1994.

[9] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997.

[10] Robin Cockett and Pieter Hofstra. Introduction to turing categories. *Annals of Pure and Applied Logic*, 156(2–3):183–209, 2008.

[11] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

[12] Robert A. Di Paola and Alex Heller. Dominical categories: Recursion theory without elements. *The Journal of Symbolic Logic*, 52(3):594–635, 1987.

[13] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[14] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

[15] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space complexity of parameterized problems. In *Parameterized and Exact Computation*, volume 7535 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2012.

[16] Michael R. Fellows. Parameterized complexity: the main ideas and some research frontiers. In *Proceedings of ISAAC*, volume 2223 of *Lecture Notes in Computer Science*, pages 291–307. Springer, 2001.

[17] Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Information and Computation*, 187(2):291–319, 2003.

[18] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[19] Lance Fortnow and Steve Homer. A short history of computational complexity. *Bulletin of the EATCS*, 80:95–133, 2003.

[20] Lance Fortnow and Martin Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161(1-2):123–140, 1996.

[21] Heidi Gebauer, Robin A. Moser, Dominik Scheder, and Emo Welzl. The Lovász local lemma and satisfiability. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 30–54. Springer, 2009.

[22] Peter D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, 2007.

[23] Juris Hartmanis. On log-tape isomorphisms of complete sets. *Theoretical Computer Science*, 7(3):273–286, 1978.

[24] Bart M.P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited. *Theory of Computing Systems*, 53(2):263–299, 2013.

[25] F. William Lawvere. Diagonal arguments and cartesian closed categories. In *Category Theory, Homology Theory and their Applications II*, volume 92 of *Lecture Notes in Mathematics*, pages 134–145. Springer, 1969.

[26] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

[27] Jochen Messner and Thomas Thierauf. A Kolmogorov complexity proof of the Lovász local lemma for satisfiability. In *Computing and Combinatorics*, volume 6842 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2011.

[28] Pekka Orponen, Ker-i Ko, Uwe Schöning, and Osamu Watanabe. Instance complexity. *Journal of the ACM*, 41(1):96–121, 1994.