

2.1 The Basic Definition

Our concerns are (decision) problems with two or more inputs. Thus we will be considering languages $L \subseteq \Sigma^* \times \Sigma^*$. We refer to such languages as *parameterized languages*. If $\langle x, k \rangle$ is in a parameterized language L , we call k the parameter.¹ Usually the parameter will be a positive integer, but it might be a graph or algebraic structure. However, in the interest of readability and with no loss of generality, we will usually identify the domain of the parameter as the natural numbers (in *unary*) \mathbb{N} and hence consider languages $L \subseteq \Sigma^* \times \mathbb{N}$. For a fixed k , we call $L_k = \{\langle x, k \rangle : \langle x, k \rangle \in L\}$ the k th *slice* of L .

As we have seen in the introduction, our main idea is to study languages that are tractable “by the slice.” As the reader will recall, being tractable by the slice meant that there is a constant c , independent of k , such that for all k , membership of L_k can be determined in time $O(|x|^c)$. There are various levels of non-uniformity and of non-computability available for such a definition, but for almost all of this book, the reader can take the following as the *definition* of fixed-parameter tractability.

Definition 2.1.1 (The basic definition) We say that a parameterized language L is (strongly uniformly) *fixed-parameter tractable* (FTP) iff there exists an algorithm Φ and a constant c and a computable function f such that, for all x, k , $\Phi(\langle x, k \rangle)$ runs in time at most $f(k)|x|^c$ and

$$\langle x, k \rangle \in L \quad \text{iff} \quad \Phi(\langle x, k \rangle) = 1.$$

¹There is another tradition here suggested by Flum and Grohe [312] that the parameter be a *function* $\kappa : \Sigma \times \Sigma \rightarrow \Sigma^*$. We believe that the original definition is explicit enough and certainly appropriate in practical applications. We only mention this fact for the reader who looks at material in the literature using this notation.

The reader may be alerted by the presence of “computable function f ” in Definition 2.1.1. Isn’t this supposed to be a theory designed to address complexity in practical computation? Definition 2.1.1 allows for functions like $f(k) = 2^{2^{\cdot^{\cdot^{\cdot^k \text{ times}}}}}$, or even worse! Of course, it could well be argued that for polynomial time similarly horrible polynomial-time running times could happen, and hence this is also an issue in the definition of P.

As it turns out, generally if we use the elementary toolkit described in the first chapters, c is usually small (1, 2 or maybe 3) and the constant $f(k)$ is usually a manageable functions of k , like 2^k .

For example, consider VERTEX COVER parameterized by k , defined as follows.

VERTEX COVER

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Does G have a vertex cover of size $\leq k$? (A vertex cover of a graph G is a collection of vertices V' of G such that for all edges $v_1 v_2$ of G either $v_1 \in V'$ or $v_2 \in V'$.)

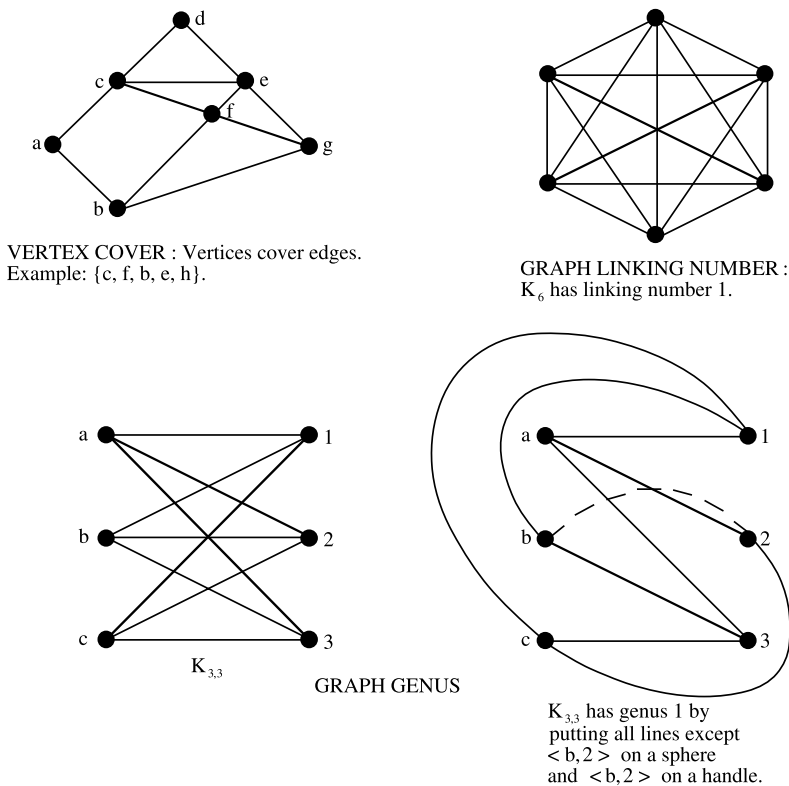
Then, as we soon see, VERTEX COVER, parameterized by k , can be solved in time $2^k |G|$. This running time can be written in a more convenient form.

Definition 2.1.2 (The O^* notation) If a parameterized algorithm has running time $f(k)|x|^c$, we will write that the algorithm has a running time of $O^*(f(k))$, i.e. ignoring the polynomial part and concentrating on the exponential part.

In the case of VERTEX COVER as above, we would say we can solve it in time $O^*(2^k)$. Some authors (such as Flum and Grohe [312]) choose to write p -VERTEX COVER to emphasize that we are dealing with the problem as a parameterized one.

2.2 The Other Flavors of Parameterized Tractability

Now, even though at this stage it might be somewhat mysterious, we feel that it is important to point out that there really are other flavors of parameterized tractability. These other definitions involve the use of non-computability, first in terms of the constant, and secondly as a non-uniformity in the algorithm itself. The use of non-computability in complexity is not new, with classes like P/Poly being uncountable. However, we will try to motivate these classes with examples. Before we give some precise definitions, we invite the reader to consider the following examples to add to the one we have seen so far, VERTEX COVER.

**Fig. 2.1** Examples of FPT problems**Example 2.2.1** GRAPH GENUS*Instance:* A graph $G = (V, E)$.*Parameter:* A positive integer k .*Question:* Does G have genus k ? (That is, can G be embedded with no edges crossing on a surface with k handles?)**Example 2.2.2** GRAPH LINKING NUMBER*Instance:* A graph $G = (V, E)$.*Parameter:* A positive integer k .*Question:* Can G be embedded into 3-space such that the maximum size of a collection of topologically linked disjoint cycles is bounded by k ?

In Fig. 2.1 we give some examples to illustrate the problems above. The fact that K_6 has linking number 1 is due to Sachs [597] and Conway and Gordon [162]. If we consider the classical versions of the problems above where k is not fixed, then they are all NP-hard. Each of the above problems exhibits some form of param-

terized tractability. As we mentioned above, in the next section, we will look at a simple technique called the method of bounded search trees that can solve VERTEX COVER using a single algorithm Φ running in time $2^k|G|$ for each k . Fellows and Langston [297, 299] introduced a method which allowed them to use the deep results of Robertson and Seymour [588, 589] to construct a single algorithm Φ which accepts $\langle G, k \rangle$ as input instances of GRAPH GENUS such that Φ determines if G has genus k in time $O(|G|^3)$. This running time was improved by Mohar [533] to $O(|G|)$. Finally, Fellows and Langston [297, 299] used the Robertson–Seymour methods to prove that for each k there is an algorithm Ψ_k which runs in time $O(|G|^3)$ and which determines if the graph G has linking number k . (We will look at the proofs of these results and the techniques used in Chap. 17.)

Notice the differences between the three varieties of fixed-parameter tractability. They are all $O(|G|^c)$ for some c slice-wise but:

- In the case of VERTEX COVER, we have a single *known* algorithm which works for all k , and, moreover, we can compute the constant and hence the exact running time for each k . This is the behavior we gave in Definition 2.1.1 and called it there *strongly uniform fixed-parameter tractability*.
- In Example 2.2.1, for GRAPH GENUS, we still have a single algorithm Φ for all k , but this time we have *no* way of computing the constant in the running time. We merely know that for each k , the running time of Φ on input $\langle G, k \rangle$ is $O(|G|^3)$. This behavior is called *uniform fixed-parameter tractability*.
- In Example 2.2.2, for GRAPH LINKING NUMBER, all we know is the exponent of the running time for the algorithms. For each k , we have a different (and unknown) algorithm running in $O(|G|^3)$ with unknown constants. This behavior is called *non-uniform fixed-parameter tractability*.

Considerations such as the examples above naturally lead us to the definitions below.

Definition 2.2.1 (Uniform and non-uniform FPT) Let A be a parameterized problem.

- (i) We say that A is *uniformly fixed-parameter tractable* if there is an algorithm Φ , a constant c , and an arbitrary function $f : \mathbb{N} \mapsto \mathbb{N}$ such that: the running time of $\Phi(\langle x, k \rangle)$ is at most $f(k)|x|^c$,
- (ii) We say that A is *non-uniformly fixed-parameter tractable* if there is a constant c , a function $f : \mathbb{N} \mapsto \mathbb{N}$, and a collection of procedures $\{\Phi_k : k \in \mathbb{N}\}$ such that for each $k \in \mathbb{N}$, and the running time of $\Phi_k(\langle x, k \rangle)$ is $f(k)|x|^c$ and $\langle x, k \rangle \in A$ iff $\Phi_k(\langle x, k \rangle) = 1$.

In [242, 247], Downey and Fellows give proofs constructing languages showing that the definitions above generate three distinct classes of computable parameterized languages. The diagonalization arguments used construct artificial languages to provably separate the classes. At present, we are not aware of any natural problem that is provably (say) in the class of problems that are uniformly but not strong

uniformly fixed-parameter tractable. We remark, however, that there are examples of natural problems such as GRAPH LINKING NUMBER which can at present only be classified as non-uniformly tractable. Nevertheless, in practice, it seems that the most important types of parameterized tractability are the two varieties of uniform tractability.

The reader may well ask why we bothered to introduce these apparently exotic classes. One of the reasons concerns lower bounds for algorithms. That is, later in Chaps. 29 and 30, we will prove results about lower bounds on the power of certain kinds of parameterized algorithms. These lower bounds require some complexity assumptions² which need the more general notions of FPT to be used. Additionally, the uniform and non-uniform versions come to center stage in the chapter where we look at applications of the Robertson–Seymour methodology, Chap. 17. For the present the readers should file the notions above in the backs of their minds.

Therefore until Chaps. 17 and 30, the reader can assume that unless otherwise specified, fixed-parameter tractability will mean *strongly uniform fixed-parameter tractability*.

We remark that most natural FPT problems which have been studied so far seem to exhibit a general migration toward strongly uniform tractability; that is, once a problem has been identified as FPT, then with more precise combinatorics, we eventually show the problem to be strongly uniformly FPT.

One reason for this general process of improvement is that often the initial classification of the problem as FPT came from the application of some general result pertaining to a class of problems containing the one at hand. Later, more uniformity, together with better constants and algorithms, can be obtained after studying the particular combinatorics of the problem considered alone. It is an open question whether there are natural examples of languages which are FPT but provably not strongly uniformly FPT.

Parameterized complexity is orthogonal to classical complexity; that is, the parameterized complexity of a problem can bear no relationship to the classical complexity of the problem. For instance, let L be *any* computable language. Perhaps L is not even elementary recursive and the time complexity for L might dwarf Ackermann's function. But consider the parameterized language $L' = \{\langle x, x \rangle : x \in L\}$. Classically, L' has the same complexity as L , but as a parameterized problem, L' is in FPT, with a constant time algorithm. Later, we will see examples where the parameterized versions are hard but classical versions are easy.

2.3 Exercises

Exercise 2.3.1 List 20 aspects of a graph you might regard as a parameter.

Exercise 2.3.2 Recall that GRAPH 3 COLORING is NP-complete. Use this to prove that GRAPH k COLORING (i.e. $\langle G, k \rangle \in L$ iff G is k -colorable) is FPT iff $P = NP$.

²Like $W[1] \neq \text{FPT}$, as we will later define.

Exercise 2.3.3 The notion of parameterization is not restricted to P . Give a definition of a language $L \subseteq \Sigma^* \times \mathbb{N}$ being in parameterized LOGSPACE.

Exercise 2.3.4 (Cai, Chen, Downey, and Fellows [122])

1. We say that a parameterized language L is (uniformly) *eventually* in $\text{DTIME}(n^d)$ iff there is an algorithm Φ such that, for each k , there exists $m = m(k) \in \mathbb{N}$ such that, for all x with $|x| > n$, $\langle x, k \rangle \in L$ iff $\Phi(\langle x, k \rangle) = 1$ and Φ runs in time $|x|^d$ on input $\langle x, k \rangle$. Prove that L is in uniform FPT iff there is a d such that L is uniformly eventually in $\text{DTIME}(n^d)$. Moreover, show that we can replace “uniform” by “strongly uniform” if the function $k \mapsto m(k)$ is computable.
2. (This is called the “advice view”.) $L \in \text{FPT}$ (non-uniform) iff there is a polynomial-time oracle Turing machine Φ and a function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x and k

$$\langle x, k \rangle \in L \quad \text{iff} \quad \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

3. Furthermore, $L \in \text{FPT}$ (*strongly uniform*) iff there is a polynomial-time oracle Turing machine Φ and a *computable* function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x and k

$$\langle x, k \rangle \in L \quad \text{iff} \quad \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

Exercise 2.3.5 (Challenging) Prove that the following problems are in FPT.

- (i) VERTEX COVER.
- (ii) PLANAR INDEPENDENT SET.

2.4 Historical Notes

Ever since the discovery of the fact that many natural problems are seemingly intractable, authors have looked for feasible partial solutions. We refer the reader to Garey and Johnson [337], particularly Chap. 4. In that chapter, Garey and Johnson look at what they call “Analyzing Subproblems”. The reason for Garey and Johnson’s interest in subproblems is encapsulated in the following quote:

“If a general problem is NP-complete, we know that an exponential time algorithm will be needed (unless $P = NP$), but there are a variety of ways which the time complexity of an algorithm can be “exponential,” some of which might be preferable to others.” (Garey and Johnson [337, p. 106].)

In retrospect, it is obvious that many authors have devised algorithms which demonstrate the fact that the parameterized problem is FPT. For instance, as we will see in Chap. 9, all pseudo-polynomial-time algorithms actually demonstrate the fact that the relevant problem is FPT. To our knowledge, the first author to explicitly note the fact that as k varied, problems such as DOMINATING SET seemed to take time $\Omega(n^{f(k)})$ with $f(k) \rightarrow \infty$ was Ken Regan in some comments in [577]. There was also a reference by Moshe Vardi [655], who suggested that classical complexity was the wrong notion for databases, since the queries were very small compared to

the database. In particular, Vardi pointed out that the input for database-query evaluation consists of two components, query and database. For first-order queries, query evaluation is PSPACE-complete, and for fixpoint query it is EXPTIME-complete, but, if you fix the query, the complexity goes down to LOGSPACE and PTIME correspondingly. In particular, the size of the database was not the right complexity for database-query complexity and the size of parameter counted. Also in the 1980s were the papers Vardi and Wolper [656] and Lichtenstein and A. Pnueli [491] who pointed out that the input for LTL model checking consists of two components, formula and transition system. LTL model checking is PSPACE-complete, but if you fix the formula, the complexity goes down to LOGSPACE.

So people in the database community were very aware that fixing a parameter makes an intractable problem tractable. In retrospect the key is that they did miss the big difference between query evaluation and model checking. In query evaluation the dependence on the formula is exponential, while in model checking it is multiplicative. Indeed, as was shown later, model checking is FPT and query evaluation is likely not FPT.

The first paper to address the “asymptotic” fixed-parameter complexity of parameterized problems (i.e. the behavior as $k \rightarrow \infty$), was Abrahamson, Ellis, Fellows and Mata [6]. Roughly speaking, those authors looked at the comparison between languages that were non-uniformly FPT and those that were “P-complete” or “dual P-complete” by the slice. There are a number of severe limitations and other more technical problems with the Abrahamson et al. approach. Those authors certainly could not address comparisons of the parameterized complexities of, say, VERTEX COVER and DOMINATING SET, nor parameterized versions of problems outside of NP. The basic definitions of this chapter were given in Downey and Fellows [241, 243, 244].

We will give more detailed historical remarks concerning the issue of fixed-parameter tractability versus intractability in Part II, where they will be more in context. We will also give historical comments concerning the various techniques examined in this book in the relevant chapters and sections.

Finally, recent histories of the early years of parameterized complexity, and the preceding work on Fellows and Langston can be found in Downey [237] and Langston [479]. Other related materials about the early years of parameterized complexity can be found in the festschrift volume Bodlaender, Downey, Fomin and Marx [85].

2.5 Summary

This chapter introduced the main definition of FPT, and introduces two less uniform variations which are important in some later chapters; particularly those describing methods based on graph minor techniques (Chap. 19) and for lower bound arguments (Chap. 30).

Fundamentals of Parameterized Complexity

Downey, R.; Fellows, M.R.

2013, XXX, 763 p. 83 illus., Hardcover

ISBN: 978-1-4471-5558-4