
*Algorithms and Theory of
Computation Handbook, Second
Edition*

CRC PRESS
Boca Raton Ann Arbor London Tokyo



Contributors



Contents

1	Parameterized Algorithms	1
	<i>Rodney G. Downey and Catherine McCartin</i> School of Mathematical and Computing Sciences, Victoria University, New Zealand; Institute of Information Sciences and Technology, Massey University, New Zealand	
1.1	Introduction	1
1.2	The Main Idea	2
1.3	Practical FPT Algorithms	5
1.3.1	Kernelization	5
1.3.2	Depth-bounded Search Trees	11
1.3.3	Interleaving	14
1.3.4	Iterative Compression	14
1.4	Not-Quite-Practical FPT Algorithms	16
1.4.1	Color-coding	17
1.4.2	Bounded Width Metrics	18
1.4.3	Algorithmic Meta-theorems	21
1.5	Parameterized Approximation Algorithms	25
1.6	Conclusions	26



1

Parameterized Algorithms

Rodney G. Downey and Catherine McCartin

School of Mathematical and Computing Sciences, Victoria University, New Zealand; Institute of Information Sciences and Technology, Massey University, New Zealand

CONTENTS

1.1	Introduction	1
1.2	The Main Idea	2
1.3	Practical FPT Algorithms	5
1.4	Not-Quite-Practical FPT Algorithms	16
1.5	Parameterized Approximation Algorithms	25
1.6	Conclusions	25

1.1 Introduction

Since the advent of classical complexity theory in the early 1970's, the twin notions of *NP*-hardness and *NP*-completeness have been accepted as concrete measures of computational intractability. However, a verdict of *NP*-hardness does not do away with the need for solving hard computational problems, since the bulk of these problems are of great theoretical and practical importance.

The field of parameterized complexity theory and parameterized computation has developed rapidly over the past twenty years as a robust approach to dealing with hard computational problems arising from applications in diverse areas of science and industry. The parameterized paradigm augments classical complexity theory and computation, providing, on the one hand, systematic and practical algorithm design techniques for hard problems, and, on the other hand, more finely-grained complexity analysis and stronger computational lower bounds for natural computational problems.

The theory is based on the simple observation that many hard computational problems have certain aspects of their input, or expected solution, that vary only within a moderate range, at least for instances that are of practical importance. By exploiting such small associated parameters, many classically intractable problems can be efficiently solved.

Apparent parameterized intractability is established via a completeness program, which parallels the traditional paradigm, but allows for stratification of problems into

a far more richly-structured hierarchy of complexity classes.

A number of approaches have been proposed to deal with the central issue of computational intractability, including polynomial time approximation, randomization and heuristic algorithms. The parameterized paradigm is orthogonal to each of these earlier approaches, yet a range of fundamental connections has begun to emerge.

The aim of this chapter is to survey the current state of the art in the field of parameterized complexity, canvassing main techniques and important results. We concentrate on the distinctive algorithmic techniques that have emerged in the field, in particular those that lead to practical and useful algorithms for classically intractable problems.

1.2 The Main Idea

It is generally accepted that solving an *NP*-hard problem will necessarily entail a combinatorial explosion of the search space. However, it is *not* necessarily the case that all instances of an *NP*-hard problem are equally hard to solve, hardness sometimes depends on the particular structure of a given instance, or of the expected solution. Instances of *NP*-hard problems arising from “real life” often exhibit more regular structure than the general problem description might, at first, suggest.

For example, suppose that we are concerned with solving computational problems to do with relational databases. Typically, a real life database will be huge, and the queries made to it will be relatively small. Moreover, real life queries will be questions that *people* actually ask. Hence, such queries will tend to be of low logical complexity. Thus, an algorithm that works very efficiently for small formulae with low logical depth might well be perfectly acceptable in practice. Alternatively, suppose that we are concerned with computational problems where the focus is to recognize a particular substructure in the problem input. If the size of the expected substructure is small, then an algorithm that works very efficiently for small solutions may be acceptable in practice.

The main idea of parameterized complexity is to develop a framework that addresses complexity issues in this situation, where we know in advance that certain parameters of the problem at hand are likely to be bounded, and that this might significantly affect the complexity.

The basic insight that underpins parameterized complexity and parameterized computation arose from consideration of two well-known *NP*-complete problems for simple undirected graphs.

A *vertex cover* of $G = (V, E)$ is a set of vertices $V' \subseteq V$ that covers all edges: that is $V' = \{v_1, \dots, v_k\}$ is a vertex cover for G iff, for every edge $(u, v) \in E$, either $u \in V'$ or $v \in V'$. A *dominating set* of $G = (V, E)$ is a set of vertices $V' \subseteq V$ that covers all vertices: that is $V' = \{v_1, \dots, v_k\}$ is a dominating set for G iff, for every vertex $v \in V$,

either $v \in V'$ or there is some $u \in V'$ such that $(u, v) \in E$.

VERTEX COVER

Instance: A graph $G = (V, E)$ and a positive integer k .

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

DOMINATING SET

Instance: A graph $G = (V, E)$ and a positive integer k .

Question: Does G have a dominating set of size at most k ?

Although both of these problems are, classically, *NP*-complete, the parameter k contributes to the complexity of these two problems in two qualitatively different ways.

DOMINATING SET Essentially the only known algorithm for this problem is to try all possibilities. The brute force algorithm of trying all k -subsets runs in time $O(n^{k+1})$ (we use n to denote $|V|$ and m to denote $|E|$.)

VERTEX COVER After many rounds of improvement, there is now an algorithm running in time $O(1.286^k + kn)$ ([18]) for determining if a graph $G = (V, E)$ has a vertex cover of size k . This has been implemented and is practical for n of unlimited size and k up to around 400 [47, 27].

The table below shows the contrast between these two kinds of complexity.

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

The Ratio $\frac{n^{k+1}}{2^k n}$ for Various Values of n and k .

These observations are formalized in the framework of parameterized complexity theory [25]. In classical complexity, a decision problem is specified by two items of information:

1. The input to the problem.
2. The question to be answered.

In parameterized complexity, there are three parts to a problem specification:

1. The input to the problem.
2. The aspects of the input that constitute the parameter.

3. The question to be answered.

The notion of *fixed-parameter tractability* is the central concept of the theory. Intuitively, a problem is fixed-parameter tractable (FPT) if we can somehow confine any “bad” complexity behaviour to some limited aspect of the problem, the parameter.

More formally, we consider a *parameterized language* to be a subset $L \subseteq \Sigma^* \times \Sigma^*$. If L is a parameterized language and $(I, k) \in L$ then we refer to I as the *main part* and k as the *parameter*.

DEFINITION 1.1 Fixed Parameter Tractability (FPT)

A parameterized language $L \subseteq \Sigma^* \times \Sigma^*$ is fixed-parameter tractable if there is an algorithm (or a k -indexed collection of algorithms) that correctly decides, for input $(I, k) \in \Sigma^* \times \Sigma^*$, whether $(I, k) \in L$ in time $f(k) \cdot n^c$, where n is the size of the main part of the input I , k is the parameter, c is a constant (independent of both n and k), and f is an arbitrary function.

Usually, the parameter k will be a positive integer, but it could be, for instance, a graph or an algebraic structure, or a combination of integer values bounding various aspects of the problem. The parameter will often bound the *size* of some part of the input instance or the solution. Alternatively, it can bound the *complexity* of the input instance in some well-defined sense. For example, in Sections 1.4.2 and 1.4.3 we introduce *width metrics* for graphs which precisely capture various notions of complexity in graphs. A single classical problem can often be parameterized in several natural ways, each leading to a separate parameterized problem.

Following naturally from the concept of fixed-parameter tractability are appropriate notions of parameterized problem reduction.

Apparent parameterized *intractability* is established via a completeness program. The main sequence of parameterized complexity classes is

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[t] \dots \subseteq W[P] \subseteq AW[P] \subseteq XP$$

This sequence is commonly termed the W -hierarchy. The complexity class $W[1]$ is the parameterized analog of NP . The defining complete problem for $W[1]$ is given here.

SHORT NON-DETERMINISTIC TURING MACHINE ACCEPTANCE

Instance: A nondeterministic Turing machine M and a string s .

Parameter: A positive integer k .

Question: Does M have a computation path accepting s in at most k steps?

In the same sense that NP -completeness of the $q(n)$ -STEP NON-DETERMINISTIC TURING MACHINE ACCEPTANCE, where $q(n)$ is a polynomial in the size of the input, provides us with very strong evidence that no NP -complete problem is likely to be solvable in polynomial time, $W[1]$ -completeness of SHORT NON-DETERMINISTIC TURING MACHINE ACCEPTANCE provides us with very strong evidence that no

$W[1]$ -complete problem is likely to be fixed-parameter tractable. It is conjectured that all of the containments here are proper, but all that is currently known is that FPT is a proper subset of XP .

If we compare classical and parameterized complexity it is evident that the framework provided by parameterized complexity theory allows for more finely-grained complexity analysis of computational problems. We can consider many different parameterizations of a single classical problem, each of which leads to either a tractable, or (likely) intractable, version in the parameterized setting. In this survey we concentrate on natural parameterized versions of NP -hard problems.

A collection of distinctive techniques has been developed for fixed-parameter tractable (FPT) algorithm design. Some of these techniques rely on deep mathematical results and give us general algorithms, often non-constructive, pertaining to large classes of problems. Others are simple, yet widely applicable, algorithmic strategies that systematically exploit the particular combinatorics of the problem at hand. There are now numerous examples where fixed-parameter techniques in this second category have led to solution of natural parameterized versions of NP -hard problems, both efficiently and optimally, in practice. We introduce some of these techniques in the next section.

1.3 Practical FPT Algorithms

In this section we introduce the main practical techniques that have emerged in the field of FPT algorithm design. We focus first on two simple algorithmic strategies that are not part of the usual toolkit of *polynomial* algorithm design, but which have led, in many important cases, to practical and useful algorithms for natural parameterized versions of NP -hard problems. Almost exclusively, these two techniques, (i) Kernelization and (ii) Depth-Bounded Search Trees, have formed the backbone of practical FPT algorithm design. In Section 1.3.3 we show how these two techniques can be profitably combined using the concept of *interleaving*. We also introduce *iterative compression*, a relatively new technique that has been successfully applied to a range of parameterized minimization problems, where the parameter is the size of the solution set.

1.3.1 Kernelization

Kernelization is based on an old idea, that of pre-processing, or *reducing*, the input data of a computational problem. It often makes sense to try to eliminate those parts of the input data that are relatively easy to cope with, shrinking the given instance to some “hard core” that must be dealt with using a computationally expensive algorithm. In fact, this is the basis of many heuristic algorithms for NP -hard problems, in a variety of areas, that seem to work reasonably well in practice. In other words,

it is something that many practitioners, faced with a real-world *NP*-hard problem, already do.

A compelling example of the effectiveness of data reduction, for a classically-posed *NP*-hard problem, is given by Weihe [48]. He considered the following problem in the context of the European railroad network: given a set of trains, select a set of stations such that every train meets at least one of those stations and such that the number of selected stations is minimum. Weihe modeled this problem as a path cover by vertices in an undirected graph. Here, we formulate the problem as domination of one set of vertices by another in a bipartite graph.

TRAIN COVERING BY STATIONS

Instance: A bipartite graph $G = (V_S \cup V_T, E)$, where the set of vertices V_S represents railway stations and the set of vertices V_T represents trains. E contains an edge $(s, t), s \in V_S, t \in V_T$, iff the train t stops at the station s .

Problem: Find a minimum set $V' \subseteq V_S$ such that V' covers V_T , that is, for every vertex $t \in V_T$, there is some $s \in V'$ such that $(s, t) \in E$.

Weihe employed two simple data reduction rules for this problem. For our problem formulation they translate to the following:

REDUCTION RULE TCS1:

Let $N(t)$ denote the neighbours of t in V_S . If $N(t) \subseteq N(t')$ then remove t' and all adjacent edges of t' from G . If there is a station that covers t , then this station also covers t' .

REDUCTION RULE TCS2:

Let $N(s)$ denote the neighbours of s in V_T . If $N(s) \subseteq N(s')$ then remove s and all adjacent edges of s from G . If there is a train covered by s , then this train is also covered by s' .

In practice, exhaustive application of these two simple data reduction rules allowed for the problem to be solved in minutes, for a graph modeling the whole European train schedule, consisting of around $1.6 \cdot 10^5$ vertices and $1.6 \cdot 10^6$ edges.

This impressive performance begs the question: Why should data reduction be of more concrete use in the parameterized paradigm? The answer comes from the observation that a data reduction scheme for a parameterized problem can give often give upper bounds on the size of the reduced instance in terms *solely of the parameter*. Once such a reduction scheme is established, a trivial FPT algorithm manifests as a brute-force search of the reduced instance. Thus, in the parameterized context, data reduction can often lead directly to an FPT algorithm to solve the problem. This contrasts with the classical context, where data reduction can clearly lead to a useful heuristic, but without any *provable performance guarantee*.

To illustrate the kernelization concept, we start with a simple data reduction scheme for the standard parameterized version of the *NP*-hard VERTEX COVER problem (introduced in Section 1.2.) As for subsequent examples given in this section, we paraphrase the treatment given in [31].

K-VERTEX COVER

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

Parameter: A positive integer k .

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

Vertices with no adjacent edges are irrelevant, both to the problem instance and to any solution. This leads to :

REDUCTION RULE VC1:

Remove all isolated vertices.

In order to cover an edge in E , one of its endpoints must be in the solution set. If one of these endpoints is a degree one vertex, then the other endpoint has the potential to cover more edges than the degree one vertex, leading to:

REDUCTION RULE VC2:

For any degree one vertex v , add its single neighbour u to the solution set and remove u and all of its incident edges from the graph.

The reduced instance thus consists of both a smaller graph and a smaller parameter, $(G, k) \rightarrow (G', k - 1)$.

These two data reduction rules are applicable in any problem solving context. However, in the parameterized setting, where we are looking only for a small solution, with size bounded by parameter k , we can do more. Sam Buss [16] originally observed that, for a simple graph G , any vertex of degree greater than k must belong to every k -element vertex cover of G (otherwise all the neighbours of the vertex must be included, and there are more than k of these).

REDUCTION RULE VC3:

If there is a vertex v of degree at least $k + 1$, add v to the solution set and remove v and all of its incident edges from the graph.

The reduced instance again consists of both a smaller graph and a smaller parameter, $(G, k) \rightarrow (G', k - 1)$.

After exhaustively applying these three rules, we have a new, reduced, instance, (G', k') , where no vertex in the reduced graph has degree greater than k' , or less than two. Thus, any vertex remaining can cover at most k' edges in this reduced instance. The solution set can be no larger than k' , forcing the reduced graph to either contain no more than k'^2 edges, and no more than k'^2 vertices, or, otherwise, to be a NO instance.

Thus, in a polynomial amount of time, we have reached a situation where we can either declare our original instance to be a NO instance, or, by means of a brute force search of the reduced instance, in time $O(k'^{2k'})$, $k' \leq k$, decide whether our original instance admits a vertex cover of size at most k .

The important point is that the reduced instance can either be immediately declared a NO instance or, otherwise, has size bounded by a *function of the parameter*.

We formalise this idea in terms of a *reduction to a problem kernel*, or *kernelization*.

DEFINITION 1.2 Kernelization

Let $L \subseteq \Sigma^* \times \Sigma^*$ be a parameterized language. Let \mathcal{L} be the corresponding parameterized problem, that is, \mathcal{L} consists of input pairs (I, k) , where I is the main part of the input and k is the parameter. A reduction to a problem kernel, or kernelization, comprises replacing an instance (I, k) by a reduced instance (I', k') , called a problem kernel, such that

- (i) $k' \leq k$,
- (ii) $|I'| \leq g(k)$, for some function g depending only on k , and
- (iii) $(I, k) \in L$ if and only if $(I', k') \in L$.

The reduction from (I, k) to (I', k') must be computable in time polynomial in $|I|$.

The kernelization for K -VERTEX COVER described above uses rules that examine only *local* substructures of the input (a vertex and its neighbourhood.) For a range of problems, this approach proves adequate for producing a reasonably-sized problem kernel. Another possibility is to consider the *global* properties of a problem instance.

Chen *et al.* [18] have used this second approach in exploiting a well-known theorem of Nemhauser and Trotter [35] to construct a problem kernel for VERTEX COVER having only at most $2k$ vertices. This seems to be the best that one could hope for, since a problem kernel of size $(2 - \varepsilon) \cdot k$, with constant $\varepsilon > 0$, would imply a factor $2 - \varepsilon$ polynomial-time approximation algorithm for VERTEX COVER. The existence of such an algorithm is a long-standing open question in the area of approximation algorithms for *NP*-hard problems.

THEOREM 1.1 Nemhauser and Trotter (1975)

For an n -vertex graph $G = (V, E)$ with m edges, we can compute two disjoint sets $C' \subseteq V$ and $V' \subseteq V$, in $O(\sqrt{n} \cdot m)$ time, such that the following three properties hold:

- (i) There is a minimum size vertex cover of G that contains C' .
- (ii) A minimum vertex cover for the induced subgraph $G[V']$ has size at least $|V'|/2$.
- (iii) If $D \subseteq V'$ is a vertex cover of the induced subgraph $G[V']$, then $C = D \cup C'$ is a vertex cover of G .

THEOREM 1.2 Chen et al. (2001)

Let $(G = (V, E), k)$ be an instance of K -VERTEX COVER. In $O(k \cdot |V| + k^3)$ time we can reduce this instance to a problem kernel $(G = (V', E'), k')$ with $|V'| \leq 2k$.

The kernelization begins by applying the three reduction rules described above,

VC1, VC2 and VC3, to produce a reduced instance (G', k') , where G' contains at most $O(k^2)$ vertices and edges, and $k' \leq k$. This reduction takes $O(k \cdot |V|)$ time.

For the resulting reduced instance (G', k') we compute the two sets C' and V' as described in Theorem 1.1. Determining the two sets C' and V' involves computation of a maximum matching on a graph constructed from G' and can be achieved in time $O(\sqrt{k^2} \cdot k^2) = O(k^3)$.

The set C' contains vertices that have to be in the vertex cover, so we define a new parameter $k'' = k' - |C'|$. Due to Theorem 1.1, we know that if $|V'| > 2k''$ then there is no vertex cover of size k for the original graph G . Otherwise, we let the induced subgraph $G[V']$ be the problem kernel, having size at most $2k'' \leq 2k$. By Theorem 1.1, the remaining vertices for a minimum vertex cover of G can be found by searching for a minimum vertex cover in $G[V']$.

Recently, a third alternative to both local and global data reduction schemes has been explored. In this third case, local rules are generalized to examine *arbitrarily large substructures*. Continuing with our running example, κ -VERTEX COVER, we show that the local rule VC2, which entails the deletion of any degree-1 vertex and the admission of its sole neighbour into the vertex cover, can be generalized to the *crown reduction rule*.

A *crown* in a graph $G = (V, E)$ consists of an independent set $I \subseteq V$ (no two vertices in I are connected by an edge) and a set H containing all vertices in V adjacent to I . $I \cup H$ is a crown in G iff there exists a size $|H|$ maximum matching in the bipartite graph induced by the edges between I and H , that is, every vertex of H is matched. It is clear that degree-1 vertices in V , coupled with their sole neighbours, can be viewed as the most simple crowns in G .

If we find a crown $I \cup H$ in G , then we need at least $|H|$ vertices to cover all edges in the crown. Since all edges in the crown can be covered by admitting at most $|H|$ vertices into the vertex cover, there is a minimum size vertex cover that contains all vertices in H and no vertices in I . These observations lead to the following reduction rule.

REDUCTION RULE CR:

For any crown $I \cup H$ in G , add the set of vertices H to the solution set and remove $I \cup H$ and all of the incident edges of $I \cup H$ from G .

The reduced instance thus consists of a smaller graph and a smaller parameter, $(G, k) \rightarrow (G', k - |H|)$. For both instance and parameter the reduction may be significant.

We are now faced with two issues. How to find crowns efficiently? and how to bound the size of the problem kernel that eventuates?

In [2] it is shown that finding a crown in a graph G can be achieved in polynomial time by computing maximum matchings in G . The size of the reduced instance that results is upper-bounded via the following theorem.

THEOREM 1.3 Abu-Khzam, Collins, Fellows, Langston, Suters,

Symons (2004)

A graph that is crown-free and has a vertex cover of size at most k can contain at most $3k$ vertices.

Another strategy for employment of crown reductions makes use of the following lemma from [21].

LEMMA 1.1 Chor, Fellows, Juedes (2004)

If a graph $G = (V, E)$ has an independent set $V' \subset V$ such that $|N(V')| < |V'|$, then a crown $I \cup H$ with $I \subseteq V'$ can be found in G in time $O(n + m)$.

The following simple crown kernelization algorithm, given in [46], uses this strategy to produce either a correct NO answer, or a problem kernel of size at most $4k$, for K-VERTEX-COVER.

We start by computing a maximal matching M in G . If $|V(M)| > 2k$ then we output NO. Since we have to pick at least one vertex for each edge in the matching, the vertex cover for this graph is greater than k . Otherwise, if $|V(M)| \leq 2k$, then there are two possibilities:

If $|V(G) - V(M)| > 2k$ then, by Lemma 1.1 we can find a crown $I \cup H$ in G in time $O(n + m)$. Since M is a maximal matching it must be the case that $V(G) - V(M)$ is an independent set in G . If we assume that G does not contain any isolated vertices then each vertex in $V(G) - V(M)$ must be adjacent to some vertex in $V(M)$. The reduced instance is $(G[V - (I \cup H)], k - |H|)$.

If $|V(G) - V(M)| \leq 2k$ then $|V(G)| = |V(M)| + |V(G) - V(M)| \leq 2k + 2k = 4k$ so G is the required problem kernel of size at most $4k$.

The three kernelizations given here for K-VERTEX-COVER make for a compelling argument in support of data reduction in the parameterized context. In comparison with polynomial approximation, kernelization equals the conjectured best possible result for this particular problem. However, since K-VERTEX COVER is considered to be the “success story” of parameterized computation, it is fair to ask whether or not the program works so well in general. There are by now a plethora of kernelization algorithms in the literature, covering applications in widely diverse areas of science and industry. Many of these yield sufficiently small problem kernels to be of concrete practical use. In this regard, the benchmark is a *linear* kernel, where the size of the fully reduced instance is a (small) linear function of the parameter. However, we note the following two caveats regarding kernelization.

For some problems obtaining a problem kernel is trivial. The following example is given in [46]. We consider the K-DOMINATING SET problem for cubic graphs, where all vertices have degree three. No vertex in such a graph can dominate more than four vertices, itself and three neighbours. Thus, we can safely answer NO whenever the input graph has more than $4k$ vertices. Note that this problem kernel of $4k$ vertices and at most $O(12k)$ edges is obtained without the application of *any* reduction rule at all. However, by the same argument we see that no cubic graph has a dominating set of size *less than* $n/4$. Thus, for any non-trivial problem instance, we

have $k \geq n/4$ and $4k \geq n$. The bound obtained for the size of the kernel is at least as large as the size of the instance itself.

In this case, a more sensible problem to consider arises from the idea of *bounding above the guarantee*, first introduced in [33]. Given a cubic graph G and parameter k , it makes more sense to ask if there is a dominating set of size $n/4 + k$ for G . Now, the parameter contributes to the problem in a non-trivial fashion, since it has become a bound on the distance of the solution from some guaranteed minimum.

All of the problem kernels that we have introduced here are commonly called linear kernels, since the number of vertices in the fully reduced instance is a linear function of the parameter. A more accurate description is to say that they are *polynomial kernels*, since the number of graph edges in the reduced instance may be quadratic in the size of the parameter. Recent results [10] suggest that some parameterized problems likely won't admit any polynomial kernel (i.e any problem kernel whose size is an arbitrary polynomial function of the parameter), under reasonable complexity-theoretic hypotheses, even though they can be shown to be FPT using some of the not-quite-practical FPT methods we introduce later in Section 1.4. This suggests that, for such problems to have *practical* FPT algorithms, these must be of rather unusual types.

1.3.2 Depth-bounded Search Trees

Many parameterized problems can be solved by the construction of a search tree whose *depth* depends only upon the parameter. The total size of the tree will necessarily be an exponential function of the parameter, to keep the size of the tree manageable the trick is to find efficient *branching rules* to successively apply to each node in the search tree.

Continuing with our running example, consider the following simple algorithm for the K-VERTEX COVER problem.

We construct a binary tree of height k . We begin by labelling the root of the tree with the empty set and the graph $G = (V, E)$. Now we pick any edge $(u, v) \in E$. In any vertex cover of G we must have either u or v , in order to cover the edge (u, v) , so we create children of the root node corresponding to these two possibilities. The first child is labeled with $\{u\}$ and $G - u$, the second with $\{v\}$ and $G - v$. The set of vertices labeling a node represents a possible vertex cover, and the graph labeling a node represents what remains to be covered in G . In the case of the first child we have determined that u will be in our possible vertex cover, so we delete u from G , together with all its incident edges, as these are all now covered by a vertex in our possible vertex cover.

In general, for a node labeled with a set S of vertices and subgraph H of G , we arbitrarily choose an edge $(u, v) \in E(H)$ and create the two child nodes labeled, respectively, $S \cup \{u\}$, $H - u$, and $S \cup \{v\}$, $H - v$. At each level in the search tree the size of the vertex sets that label nodes will increase by one. Any node that is labeled with a subgraph having no edges must also be labeled with a vertex set that covers all edges in G . Thus, if we create a node at height at most k in the tree that

is labeled with a subgraph having no edges, then a vertex cover of size at most k has been found.

There is no need to explore the tree beyond height k , so this algorithm runs in time $O(2^k \cdot n)$.

In many cases, it is possible to significantly improve the $f(k)$, the function of the parameter that contributes exponentially to the running time, by shrinking the search tree. In the case of K -VERTEX COVER, Balasubramanian et al [7] observed that, if G has no vertex of degree three or more, then G consists of a collection of cycles. If such a G is sufficiently large, then this graph cannot have a size k vertex cover. Thus, at the expense of an additive constant factor (to be invoked when we encounter any subgraph in the search tree containing only vertices of degree at most two), we need consider only graphs containing vertices of degree three or greater.

We again construct a binary tree of height at most k . We begin by labelling the root of the tree with the empty set and the graph G . Now we pick any vertex $v \in V$ of degree three or greater. In any vertex cover of G we must have either v or *all of its neighbours*, so we create children of the root node corresponding to these two possibilities. The first child is labeled with $\{v\}$ and $G - v$, the second with $\{w_1, w_2, \dots, w_p\}$, the neighbours of v , and $G - \{w_1, w_2, \dots, w_p\}$. In the case of the first child, we are still looking for a size $k - 1$ vertex cover, but in the case of the second child we need only look for a vertex cover of size $k - p$, where p is at least 3. Thus, the bound on the size of the search tree is now somewhat smaller than 2^k .

Using a recurrence relation to determine a bound on the number of nodes in this new search tree, it can be shown that this algorithm runs in time $O(5^{k/4} \cdot n)$.

A third search tree algorithm for K -VERTEX COVER, given in [31] uses the following three branching rules:

BRANCHING RULE VC1:

If there is a degree one vertex v in G , with single neighbour u , then there is a minimum size cover that contains u (by the argument given for rule VC2 in Section 1.3.1.) Thus, we create a single child node labeled with $\{u\}$ and $G - u$.

BRANCHING RULE VC2:

If there is a degree two vertex v in G , with neighbours w_1 and w_2 , then either both w_1 and w_2 are in a minimum size cover, or v together with *all other neighbours* of w_1 and w_2 are in a minimum size cover.

To see that this rule is correct, assume that there is a minimum size cover containing v and only one of its neighbours. Replacing v with its second neighbour would then also yield a minimum size cover and this is the cover that will be constructed in the first branching case. Thus, if there is a cover that is smaller than that containing both w_1 and w_2 then this cover must contain v and neither w_1 nor w_2 . This implies that all other neighbours of w_1 and w_2 must be in this cover.

BRANCHING RULE VC3:

If there is a degree three vertex v in G , then either v or all of its neighbours are in a

minimum size cover.

Using a recurrence relation, it can be shown that if there is a solution of size at most k then the size of the corresponding search tree has size bounded above by $O(1.47^k)$.

The basic method of finding efficient branching rules is to look for a *structure* in the problem input which gives rise to only a few alternatives, one of which must lead to an acceptable solution, if such a solution exists. In all of the examples given here for κ -VERTEX COVER, this structure consists of a vertex and its one or two hop neighbourhood. The “smallest” search tree found so far for κ -VERTEX COVER has size $O(1.286^k)$ [18] and is achieved by more complex case analysis than that described here, although the structures identified still consist simply of small local neighbourhoods in the input graph.

We now briefly canvas two quite different examples of such structures which give rise to efficient branching rules for two unrelated parameterized problems. Space limitations mean that most of the problem details are left out, the intention is merely to demonstrate the nature of the possibilities for search tree algorithms.

For the CLOSEST STRING problem [32], we are given a set $S = \{s_1, s_2, \dots, s_k\}$ of k length- l strings over an alphabet Σ , and the task is to find a string whose Hamming distance is at most d from each of the $s_i \in S$. The structure that we identify is a *candidate string*, \hat{s} . At the root node of the search tree \hat{s} is simply one of the input strings. If any other string $s_i \in S$ differs from \hat{s} in more than $2d$ positions, then there is no solution for the problem. At each step we look for an input string s_i that differs from \hat{s} in more than d positions but less than $2d$ positions. Choosing $d + 1$ of these positions, we branch into $d + 1$ subcases, in each subcase modifying one position in \hat{s} to match s_i .

For the MAXIMUM AGREEMENT FOREST problem [30], we are given two phylogenetic X -trees, T_1 and T_2 , each an unrooted binary tree with leaves labeled by a common set of species X and (unlabeled) interior nodes having degree exactly three. The (labeled) topologies of T_1 and T_2 may differ. The task is to find at most k edges that can be cut from T_1 so that the resulting forest “agrees with” both T_1 and T_2 . One structure that we can identify here is called a *minimum incompatible quartet*, essentially a set of four leaf labels, $Q = \{l_1, l_2, l_3, l_4\}$, that gives rise to two different topologies in the restriction of each of the trees to those leaves labeled by Q . Any solution for the problem can be obtained by cutting at least one of a certain set of four edges induced by Q in T_1 . At each step we branch into four subcases, in each subcase cutting one of these edges.

Finally, we note that search trees inherently allow for a parallel implementation: when branching into subcases, each branch can be further explored with no reference to other branches. This has proven of concrete use in practice for VERTEX COVER [20]. Along with the idea introduced in the next section, this is one of two powerful arguments in support of the use of the depth-bounded search tree approach to obtain *really practical* FPT algorithms.

1.3.3 Interleaving

It is often possible to combine the two methods outlined above. For example, for the K -VERTEX COVER problem, we can first reduce any instance to a problem kernel and then apply a search tree method to the kernel itself.

Niedermeier and Rossmanith [37] have developed the technique of *interleaving* depth-bounded search trees and kernelization. They show that applying kernelization repeatedly during the course of a search tree algorithm can significantly improve the overall time complexity in many cases.

Suppose we take any fixed-parameter algorithm that satisfies the following conditions: The algorithm works by first reducing an instance to a problem kernel, and then applying a depth-bounded search tree method to the kernel. Reducing any given instance to a problem kernel takes at most $P(|I|)$ steps and results in a kernel of size at most $q(k)$, where both P and q are polynomially bounded. The expansion of a node in the search tree takes $R(|I|)$ steps, where R is also bounded by some polynomial. The size of the search tree is bounded by $O(\alpha^k)$. The overall time complexity of such an algorithm running on instance (I, k) is

$$O(P(|I|) + R(q(k))\alpha^k).$$

The idea developed in [37] is basically to apply kernelization at any step of the search tree algorithm where this will result in a significantly smaller problem instance. To expand a node in the search tree labelled by instance (I, k) we first check whether or not $|I| > c \cdot q(k)$, where $c \geq 1$ is a constant whose optimal value will depend on the implementation details of the algorithm. If $|I| > c \cdot q(k)$ then we apply the kernelization procedure to obtain a new instance (I', k') , with $|I'| \leq q(k)$, which is then expanded in place of (I, k) . A careful analysis of this approach shows that the overall time complexity is reduced to

$$O(P|I| + \alpha^k).$$

This really does make a difference. In [38] the 3-HITTING SET problem is given as an example. An instance (I, k) of this problem can be reduced to a kernel of size k^3 in time $O(|I|)$, and the problem can be solved by employing a search tree of size $2 \cdot 27^k$. Compare a running time of $O(2 \cdot 27^k \cdot k^3 + |I|)$ (without interleaving) with a running time of $O(2 \cdot 27^k + |I|)$ (with interleaving).

Note that, although the techniques of *kernelization* and *depth-bounded search tree* are simple algorithmic strategies, they are not part of the classical toolkit of polynomial-time algorithm design since they both involve costs that are *exponential in the parameter*.

1.3.4 Iterative Compression

Iterative compression is a relatively new technique for obtaining FPT algorithms, appearing first in a paper by Reed, Smith and Vetta in 2004 [41]. Although currently only a small number of results are known, it seems to be applicable to a range of parameterized minimization problems, where the parameter is the size of the solution set. Most of the currently known iterative compression algorithms solve *feedback set*

problems in graphs, problems where the task is to destroy certain cycles in the graph by deleting at most k vertices or edges. In particular, the K -GRAPH BIPARTISATION problem, where the task is to find a set of at most k vertices whose deletion destroys all odd-length cycles, has been shown to be FPT by means of iterative compression [41]. This has been a long-standing open problem in parameterized complexity theory.

To illustrate the concept, we again paraphrase the treatment given in [31]. The central idea is to employ a *compression routine*.

DEFINITION 1.3 Compression Routine

A *compression routine* is an algorithm that, given a problem instance I and a solution of size k , either calculates a smaller solution or proves that the given solution is of minimum size.

Using such a routine we can find an optimal solution for a parameterized problem by inductively building up the problem structure and iteratively compressing intermediate solutions. The idea is that, if the compression routine is an FPT algorithm, then so is the whole algorithm. In general, the compression routine will have running time exponential in the size of the solution provided to it, it is therefore important that each intermediate solution considered has size bounded by some $k' = f(k)$, where k is the parameter value for the original problem.

The employment of an FPT compression routine in this manner will work effectively for any parameterized minimization problem which is *monotone* in the sense that NO instances are closed under element addition. That is, given a problem instance (I, k) that is a NO instance, any problem instance (I', k) with $I \subseteq I'$ is also a NO instance. If a problem is monotone in this sense then we can immediately answer NO if we encounter an intermediate solution that cannot be compressed to meet the original parameter bound. Note that many minimization problems are not monotone in this sense. For example, a NO instance $(G = (V, E), k)$ for K -DOMINATING SET can be changed to a YES instance by means of the addition of a single vertex that is adjacent to all vertices in V .

The manner by which the problem structure is inductively produced will be normally be straightforward, the trick is in finding an efficient compression routine. Continuing with our running example, we now describe an algorithm for K -VERTEX COVER based on iterative compression.

Given a problem instance $(G = (V, E), k)$, we build the graph G vertex by vertex. We start with an initial set of vertices $V' = \emptyset$ and an initial solution $C = \emptyset$. At each step, we add a new vertex v to both V' and C , $V' \leftarrow V' \cup \{v\}$, $C \leftarrow C \cup \{v\}$. We then call the compression routine on the pair $(G[V'], C)$, where $G[V']$ is the subgraph induced by V' in G , to obtain a new solution C' . If $|C'| > k$ then we output NO, otherwise we set $C \leftarrow C'$.

If we successfully complete the n th step where $V' = V$, we output C with $|C| \leq k$. Note that C will be an optimal solution for G .

The compression routine takes a graph G and a vertex cover C for G and returns

a smaller vertex cover for G if there is one, otherwise, it returns C unchanged. Each time the compression routine is used it is provided with an intermediate solution of size at most $k + 1$.

The implementation of the compression routine proceeds as follows. We consider a smaller vertex cover C' as a *modification* of the larger vertex cover C . This modification retains some vertices $Y \subseteq C$ while the other vertices $S = C \setminus Y$ are replaced with $|S| - 1$ new vertices from $V \setminus C$.

The idea is to try by brute force all $2^{|C|}$ partitions of C into such sets Y and S . For each such partition, the vertices from Y along with all of their adjacent edges are deleted. In the resulting instance $G' = G[V \setminus Y]$, it remains to find an optimal vertex cover that is disjoint from S . Since we have decided to take no vertex from S into the vertex cover, we have to take that endpoint of each edge that is not in S . At least one endpoint of each edge in G' is in S , since S is a vertex cover for G' . If both endpoints of some edge in G' are in S , then this choice of S cannot lead to a vertex cover C' with $S \cap C' = \emptyset$. We can quickly find an optimal vertex cover for G' that is disjoint from S by taking every vertex that is not in S and has degree greater than zero. Together with Y , this gives a new vertex cover C' for G . For each choice of Y and S , this can be done in time $O(m)$, leading to $O(2^{|C|}m) = O(2^k m)$ time overall for one call of the compression routine. With at most n iterations of the compression algorithm, we get an algorithm for K -VERTEX COVER running in time $O(2^k mn)$.

Finally, we note that the employment of compression routines is not restricted to the mode detailed here. For example, we could start with a suboptimal solution, perhaps provided by some type of parameterized approximation algorithm as detailed in Section 1.5, and then repeatedly compress this solution until it is either “good enough” or we are not willing to invest more calculation time.

1.4 Not-Quite-Practical FPT Algorithms

In this section we introduce two techniques that lead to “not-quite-practical” FPT algorithms, color-coding and dynamic programming on bounded width graph decompositions. Both of these techniques have potential for practical application and have been extensively studied from a theoretical point of view. However, in contrast to the methods introduced in the previous section, these approaches have so far lead to only isolated practical implementations and experimental results.

We also introduce a series of *algorithmic meta-theorems*. These are based on results from descriptive complexity theory and topological graph theory and provide us with general FPT algorithms, sometimes non-constructive, pertaining to large classes of problems. We view these theorems not as an end in themselves, but as being useful “signposts” in the search for practically efficient fixed-parameter algorithms.

1.4.1 Color-coding

This technique is useful for problems that involve finding small subgraphs in a graph, such as paths and cycles. Introduced by Alon et al. [6], it can be used to derive seemingly efficient randomized FPT algorithms for several subgraph isomorphism problems.

We formulate a parameterized version of the SUBGRAPH ISOMORPHISM problem as follows:

K-SUBGRAPH ISOMORPHISM

Instance: $G = (V, E)$ and a graph $H = (V^H, E^H)$ with $|V^H| = k$.

Parameter: A positive integer k .

Question: Is H isomorphic to a subgraph in G ?

The idea is that, in order to find the desired set of vertices V' in G , isomorphic to H , we randomly color all the vertices of G with k colors and expect that, with some high degree of probability, all vertices in V' will obtain different colors. In some special cases of the SUBGRAPH ISOMORPHISM problem, dependent on the nature of H , this will simplify the task of finding V' .

If we color G uniformly at random with k colors, a set of k distinct vertices will obtain different colors with probability $(k!)/k^k$. This probability is lower-bounded by e^{-k} , so we need to repeat the process e^k times to have probability one of obtaining the required coloring.

We can derandomize this kind of algorithm using *hashing*, but at the cost of extending the running time. We need a list of colorings of the vertices in G such that, for *each* subset $V' \subseteq V$ with $|V'| = k$ there is at least one coloring in the list by which all vertices in V' obtain different colors. Formally, we require a k -perfect family of hash functions from $\{1, 2, \dots, |V|\}$, the set of vertices in G , onto $\{1, 2, \dots, k\}$, the set of colors.

DEFINITION 1.4 k -Perfect Hash Functions

A k -perfect family of hash functions is a family \mathcal{H} of functions from $\{1, 2, \dots, n\}$ onto $\{1, 2, \dots, k\}$ such that, for each $S \subset \{1, 2, \dots, n\}$ with $|S| = k$, there exists an $h \in \mathcal{H}$ such that h is bijective when restricted to S .

By a variety of sophisticated methods, Alon et al. [6] have proved the following:

THEOREM 1.4 Alon et al. (1995)

Families of k -perfect hash functions from $\{1, 2, \dots, n\}$ onto $\{1, 2, \dots, k\}$ can be constructed which consist of $2^{O(k)} \cdot \log n$ hash functions. For such a hash function, h , the value $h(i)$, $1 \leq i \leq n$, can be computed in linear time.

We can color G using each of the hash functions from our k -perfect family in turn. If the desired set of vertices V' exists in G , then, for at least one of these colorings, all vertices in V' will obtain different colors as we require.

We now give a very simple example of this technique. The subgraph that we will look for is a cycle of length k . We use k colors. If a k -cycle exists in the graph, then there must be a coloring that assigns a different color to each vertex in the cycle.

For each colouring h , we check every ordering c_1, c_2, \dots, c_k of the k colours to decide whether or not it *realizes* a k -cycle. We first construct a directed graph G' as follows:

For each edge $(u, v) \in E$, if $h(u) = c_i$ and $h(v) = c_{i+1(\text{mod } k)}$ for some i , then replace (u, v) with arc $\langle u, v \rangle$, otherwise delete (u, v) .

In G' , for each v with $h(v) = c_1$, we use a breadth first search to check for a cycle C from v to v of length k .

A deterministic algorithm will need to check $2^{O(k)} \cdot \log |V|$ colorings, and, for each of these, $k!$ orderings. We can decide if an ordering of colors realizes the k -cycle in time $O(k \cdot |V|^2)$. Thus, our algorithm is FPT, but, arguably, not practically efficient. The main drawback is that the $2^{O(k)} \cdot \log |V|$ bound on the size of the family of hash functions hides a large constant in the $O(k)$ exponent.

More interesting examples of applications of color-coding to subgraph isomorphism problems, based on dynamic programming, can be found in [6].

1.4.2 Bounded Width Metrics

Faced with intractable graph problems, many authors have turned to study of various restricted classes of graphs for which such problems can be solved efficiently. A number of graph *width metrics* naturally arise in this context which restrict the inherent complexity of a graph in various senses.

The idea here is that a useful width metric should admit efficient algorithms for many (generally) intractable problems on the class of graphs for which the width is small. This leads to consideration of these measures from a parameterized point of view. The corresponding naturally parameterized problem has the following form:

Let $w(G)$ denote any measure of graph width.

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

Parameter: A positive integer k .

Question: Is $w(G) \leq k$?

One of the most successful measures in this context is the notion of *treewidth* which arose from the seminal work of Robertson and Seymour on graph minors and immersions [42, 43, 44]. Treewidth measures, in a precisely defined way, how *tree-like* a graph is. The fundamental idea is that we can lift many results from trees to graphs that are tree-like. Related to treewidth is the notion of *pathwidth* which measures, in the same way, how *path-like* a graph is.

Many generally intractable problems become fixed-parameter tractable for the class of graphs that have bounded treewidth or bounded pathwidth, with the parameter being the treewidth or pathwidth of the input graph. Furthermore, treewidth and pathwidth generalize many other well-studied classes of graphs. For example, planar graphs with radius k have treewidth at most $3k$, series parallel multigraphs

have treewidth two, chordal graphs (graphs having no induced cycles of length four or more) with maximum clique size k have treewidth at most $k - 1$, interval graphs G' with maximum clique size k have pathwidth at most $k - 1$.

A graph G has treewidth at most k if we can associate a tree T with G in which each node represents a subgraph of G having at most $k + 1$ vertices, such that all vertices and edges of G are represented in at least one of the nodes of T , and for each vertex v in G , the nodes of T where v is represented form a subtree of T . Such a tree is called a *tree decomposition* of G , of width k .

DEFINITION 1.5 [Tree decomposition and Treewidth]

Let $G = (V, E)$ be a graph. A tree decomposition, TD , of G is a pair (T, \mathcal{X}) where

1. $T = (I, F)$ is a tree, and
2. $\mathcal{X} = \{X_i \mid i \in I\}$ is a family of subsets of V , one for each node of T , such that

- (i) $\bigcup_{i \in I} X_i = V$,
- (ii) for every edge $\{v, w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and
- (iii) for all $i, j, k \in I$, if j is on the path from i to k in T , then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition $((I, F), \{X_i \mid i \in I\})$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G , denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G .

DEFINITION 1.6 [Path decomposition and Pathwidth]

A path decomposition, PD , of a graph G is a tree decomposition (P, \mathcal{X}) of G where P is simply a path (i.e. the nodes of P have degree at most two). The pathwidth of G , denoted by $pw(G)$ is the minimum width over all possible path decompositions of G .

Any path decomposition of G is also a tree decomposition of G , so the pathwidth of G is at least equal to the treewidth of G . For many graphs, the pathwidth will be somewhat larger than the treewidth. For example, let B_k denote the complete binary tree of height k and order $2^k - 1$, then $tw(B_k) = 1$, but $pw(B_k) = k$.

Graphs of treewidth and pathwidth at most k are also called *partial k -trees* and *partial k -paths*, respectively, as they are exactly the subgraphs of k -trees and k -paths. There are a number of other variations equivalent to the notions of treewidth and pathwidth (see, for example, Bodlaender [9].) For algorithmic purposes, the characterizations provided by the definitions given above tend to be the most useful.

The typical method employed to produce FPT algorithms for problems restricted to graphs of bounded treewidth (pathwidth) proceeds in two stages.

1. Find a bounded-width tree (path) decomposition of the input graph that exhibits the underlying tree (path) structure.
2. Perform dynamic programming on this decomposition to solve the problem.

The following Lemma encapsulates the two properties of tree decompositions on which the dynamic programming approach relies.

LEMMA 1.2 Connected subtrees

Let $G = (V, E)$ be a graph and $TD = (T, \mathcal{X})$ a tree decomposition of G .

- (i) For all $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T .
- (ii) For each connected subgraph G' of G , the nodes in T which contain a vertex of G' induce a connected subtree of T .

In order for this approach to produce practically efficient FPT algorithms, as opposed to proving that problems are theoretically tractable, it is important to be able to produce the necessary decomposition reasonably efficiently.

Determining the treewidth or pathwidth of a graph is an *NP*-hard problem. However, polynomial time approximation algorithms have been found [11]. There is a polynomial time algorithm that, given a graph G with treewidth k , finds a tree decomposition of width at most $O(k \cdot \log n)$ for G . There is a polynomial time algorithm that, given a graph G with pathwidth k , finds a path decomposition of width at most $O(k \cdot \log^2 n)$ for G .

Bodlaender [8] gave the first *linear-time* FPT algorithms (i.e. linear in $|G|$) for the constructive versions of both κ -TREEWIDTH and κ -PATHWIDTH. Perkovic and Reed [40] have improved upon Bodlaender's work, although the $f(k)$'s involved mean that treewidth and pathwidth still remain parameters of theoretical interest only, at least in the general case.

For some graph classes, the optimal treewidth and pathwidth, or good approximations of these, can be found using practically efficient polynomial time algorithms. Examples are chordal bipartite graphs, interval graphs, permutation graphs, circle graphs, [13] and co-graphs [14].

For planar graphs, Alber et al. [3, 5] have introduced the notion of a *layerwise separation property* pertaining to the underlying parameterized problem that one might hope to solve via a small-width tree decomposition. The *layerwise separation property* holds for all problems on planar graphs for which a linear problem kernel can be constructed.

The idea here is that, for problems having this property, we can exploit the layer structure of planar graphs, along with knowledge about the structure of "YES" instances of the problem, in order to find small separators in the input graph such that each of the resulting components has small treewidth. Tree decompositions for each of the components are then merged with the separators to produce a small-width tree decomposition of the complete graph.

This approach leads to, for example, algorithms that solve K -VERTEX COVER and, more interestingly, K -DOMINATING SET, on planar graphs in time $2^{O(\sqrt{k})} \cdot n$. The algorithms work by constructing a tree decomposition of width $O(\sqrt{k})$ for the kernelized graph, and then performing dynamic programming on this decomposition.

An algorithm that uses dynamic programming on a *tree* works by computing some value, or table of values, for each node in the tree. The important point is that the value for a node can be computed using only information directly associated with the node itself, along with values already computed for the children of the node.

Extending the idea of dynamic programming on *trees* to dynamic programming on *bounded-width tree decompositions* is really just a matter of having to construct more complicated tables of values. Instead of considering a single vertex at each node and how it interacts with the vertices at its child nodes, we now need to consider a reasonably small *set* of vertices represented at each node, and how this small set of vertices can interact with each of the small sets of vertices represented at its child nodes.

The most important factor in dynamic programming on tree decompositions is the size of the tables produced. The table size is usually $O(c^k)$, where k is the width of the tree decomposition and c depends on the combinatorics of the underlying problem that we are trying to solve. We can trade off different factors in the design of such algorithms. For example, a fast approximation algorithm that produces a tree decomposition of width $3k$, or even k^2 , for a graph of treewidth k could be quite acceptable if c is small.

Cliquewidth is another graph width metric that has more recently gained prominence in algorithm design, first introduced in [24]. A graph that has cliquewidth k can be recursively constructed from single vertices with labels in $[k] = \{1, 2, \dots, k\}$ using only the *composition operations* of graph union $G = G_1 \cup G_2$, vertex relabeling $G = (G_1)_{i \rightarrow j}$, and cross-product edge insertion between label sets $G = (G_1)_{i \times j}$.

The series of composition operations (called a *k-expression*) that produces such a cliquewidth- k graph G gives rise to a decomposition of G into a tree of subgraphs of G . This decomposition then leads to a linear-time dynamic programming algorithm for many problems restricted to cliquewidth- k graphs. However, in contrast with treewidth and pathwidth, there is no known FPT algorithm that constructs such a decomposition for a given cliquewidth- k graph. A polynomial time approximation algorithm has recently been presented in [39].

For examples of dynamic programming algorithms on various bounded width graph decompositions see, for example, [15].

1.4.3 Algorithmic Meta-theorems

Descriptive complexity theory relates the logical complexity of a problem description to its computational complexity. In this context there are some useful results that relate to fixed-parameter tractability. We can view these results not as an end in themselves, but as being useful “signposts” in the search for practically efficient

fixed-parameter algorithms.

We will consider graph properties that can be defined in *first-order logic* and *monadic second-order logic*.

In first order logic we have an (unlimited) supply of *individual* variables, one for each vertex in the graph. Formulas of first-order logic (FO) are formed by the following rules:

1. **Atomic formulas:** $x = y$ and $R(x_1, \dots, x_k)$, where R is a k -ary relation symbol and x, y, x_1, \dots, x_k are individual variables, are FO-formulas.
2. **Conjunction, Disjunction:** If ϕ and ψ are FO-formulas, then $\phi \wedge \psi$ is an FO-formula and $\phi \vee \psi$ is an FO-formula.
3. **Negation:** If ϕ is an FO-formula, then $\neg\phi$ is an FO-formula.
4. **Quantification:** If ϕ is an FO-formula and x is an individual variable, then $\exists x \phi$ is an FO-formula and $\forall x \phi$ is an FO-formula.

We can state that a graph has a clique of size k using an FO-formula,

$$\exists x_1 \dots x_k \bigwedge_{1 \leq i < j \leq k} E(x_i, x_j)$$

We can state that a graph has a dominating set of size k using an FO-formula,

$$\exists x_1 \dots x_k \forall y \bigvee_{1 \leq i \leq k} (E(x_i, y) \vee (x_i = y))$$

In monadic second-order logic we have an (unlimited) supply of both individual variables, one for each vertex in the graph, and *set* variables, one for each subset of vertices in the graph. Formulas of monadic second-order logic (MSO) are formed by the rules for FO and the following additional rules:

1. **Additional atomic formulas:** For all set variables X and individual variables y , Xy is an MSO-formula.
2. **Set quantification:** If ϕ is an MSO-formula and X is a set variable, then $\exists X \phi$ is an MSO-formula, and $\forall X \phi$ is an MSO-formula.

We can state that a graph is k -colorable using an MSO-formula,

$$\exists X_1, \dots, \exists X_k \left(\forall x \bigvee_{i=1}^k X_i x \wedge \forall x \forall y (E(x, y) \rightarrow \bigwedge_{i=1}^k \neg (X_i x \wedge X_i y)) \right)$$

The problems that we are interested in are special cases of the *model-checking problem*.

Let Φ be a class of formulas (logic), and let \mathcal{D} be a class of finite relational structures. The model-checking problem for Φ on \mathcal{D} is the following problem.

Instance: A structure $\mathcal{A} \in \mathcal{D}$, and a sentence (no free variables) $\phi \in \Phi$.

Question: Does \mathcal{A} satisfy ϕ ?

The model-checking problems for FO and MSO are PSPACE-complete in general. However, as the following results show, if we restrict the class of input structures then in some cases these model-checking problems become tractable.

The most well-known result, paraphrased here, is due to Courcelle [23].

THEOREM 1.5 Courcelle 1990

The model-checking problem for MSO restricted to graphs of bounded treewidth is linear-time fixed-parameter tractable.

Detleef Seese [45] has proved a converse to Courcelle’s theorem.

THEOREM 1.6 Seese 1991

Suppose that \mathcal{F} is any family of graphs for which the model-checking problem for MSO is decidable, then there is a number n such that, for all $G \in \mathcal{F}$, the treewidth of G is less than n .

Courcelle’s theorem tells us that if we can define the problem that we are trying to solve as a model-checking problem, and we can define the particular graph property that we are interested in as an MSO-formula, then there is an FPT algorithm that solves the problem for input graphs of bounded treewidth. The theorem by itself doesn’t tell us how the algorithm works.

The automata-theoretic proof of Courcelle’s theorem given by Abrahamson and Fellows [1] provides a generic algorithm that relies on dynamic programming over labelled trees (see [25] for details of this approach.) However, this generic algorithm is really just further proof of *theoretical* tractability. The importance of the theorem is that it provides a powerful engine for demonstrating that a large class of problems is FPT. If we can couch a problem in the correct manner then we know that it is “worth looking” for an efficient FPT algorithm that works on graphs of bounded treewidth.

The next result concerns graphs of bounded *local treewidth*. The local tree width of a graph G is defined via the following function.

$$ltw(G)(r) = \max \{tw(N_r(v)) \mid v \in V(G)\}$$

where $N_r(v)$ is the neighbourhood of radius r about v .

A graph G has bounded local treewidth if there is a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that, for $r \geq 1$, $ltw(G)(r) \leq f(r)$. The concept is a relaxation of bounded treewidth. Instead of requiring that the treewidth of the graph overall is bounded by some constant, we require that, for each vertex in the graph, the treewidth of each neighbourhood of radius r about that vertex is bounded by some uniform function of r .

Examples of classes of graphs that have bounded local treewidth include graphs of bounded treewidth (naturally), graphs of *bounded degree*, *planar* graphs, and graphs of *bounded genus*.

Frick and Grohe [29] have proved the following theorem.

THEOREM 1.7 Frick and Grohe 1999

Parameterized problems that can be described as model-checking problems for FO are fixed-parameter tractable on classes of graphs of bounded local treewidth.

This theorem tells us, for example, that parameterized versions of problems such as DOMINATING SET, INDEPENDENT SET, or SUBGRAPH ISOMORPHISM are FPT on planar graphs, or on graphs of bounded degree. As with Courcelle's theorem, it provides us with a powerful engine for demonstrating that a large class of problems is FPT, but leaves us with the job of finding practically efficient FPT algorithms for these problems.

The last meta-theorem that we will present has a somewhat different flavour. We first need to introduce some ideas from topological graph theory.

A graph H is a *minor* of a graph G iff there exists some subgraph, G^H of G , such that H can be obtained from G^H by a series of *edge contractions*.

We define an edge contraction as follows. Let $e = (u, v)$ be an edge of the graph G^H . By G^H/e we denote the graph obtained from G^H by *contracting* the edge e into a new vertex v_e which becomes adjacent to all former neighbours of u and of v . H can be obtained from G^H by a series of edge contractions iff there are graphs G_0, \dots, G_n and edges $e_i \in G_i$ such that $G_0 = G^H$, $G_n \simeq H$, and $G_{i+1} = G_i/e_i$ for all $i < n$.

Note that every subgraph of a graph G is also a minor of G . In particular, every graph is its own minor.

A class of graphs \mathcal{F} is *minor-closed* if, for every graph $G \in \mathcal{F}$, every minor of G is also contained in \mathcal{F} . A very simple example is the class of graphs with no edges. Another example is the class of acyclic graphs. A more interesting example is the following:

Let us say that a graph $G = (V, E)$ is *within k vertices* of a class of graphs \mathcal{F} if there is a set $V' \subseteq V$, with $|V'| \leq k$, such that $G[V - V'] \in \mathcal{F}$. If \mathcal{F} is any minor-closed class of graphs, then, for every $k \geq 1$, the class of graphs within k vertices of \mathcal{F} , $\mathcal{W}_k(\mathcal{F})$, is also minor-closed.

Note that for each integer $k \geq 1$, the class of graphs of treewidth or pathwidth at most k is minor-closed.

Let \mathcal{F} be a class of graphs which is closed under taking of minors, and let H be a graph that is not in \mathcal{F} . Each graph G which has H as a minor is not in \mathcal{F} , otherwise H would be in \mathcal{F} . We call H a *forbidden minor* of \mathcal{F} . A *minimal forbidden minor* of \mathcal{F} is a forbidden minor of \mathcal{F} for which each proper minor is in \mathcal{F} . The set of all minimal forbidden minors of \mathcal{F} is called the *obstruction set* of \mathcal{F} .

In a long series of papers, collectively entitled “Graph Minors”, Robertson and Seymour [44] have essentially proved that any minor-closed class of graphs \mathcal{F} must have a *finite* obstruction set. Robertson and Seymour have also shown that, for a fixed graph H , it can be determined whether H is a minor of a graph G in time $O(f(|H|) \cdot |G|^3)$.

We can now derive the following theorem:

THEOREM 1.8 Minor-closed membership

If \mathcal{F} is a minor-closed class of graphs then membership of a graph G in \mathcal{F} can be determined in time $O(f(k) \cdot |G|^3)$, where k is the collective size of the graphs in the obstruction set for \mathcal{F} .

This meta-theorem tells us that if we can define a graph problem via membership in a minor-closed class of graphs \mathcal{F} , then the problem is FPT, with the parameter being the collective size of the graphs in the obstruction set for \mathcal{F} . However, it is important to note two major difficulties that we now face. Firstly, for a given minor-closed class we have a proof of the *existence* of a finite obstruction set, but no effective method for *obtaining* the obstruction set. Secondly, the minor testing algorithm has very large hidden constants (around 2^{500}), and the sizes of obstruction sets in many cases are known to be very large.

Thus, again we have a theorem that provides us with a powerful engine for demonstrating that a large class of problems is, in fact, FPT, but the problem of finding practically efficient FPT algorithms for such problems remains open.

1.5 Parameterized Approximation Algorithms

We close this chapter with a brief discussion of a relatively new topic, parameterized approximation, introduced independently by three papers presented at IWPEC 2006 (The 3rd International Workshop on Parameterized Complexity and Exact Algorithms) [26], [19], [17].

There are various ways in which parameterized complexity and parameterized computation can interact with approximation algorithms. The interplay between the two fields is covered comprehensively in [34].

We first consider approximation using instance parameters. In this approach we define a parameterized version of an optimization problem, and then, instead of looking for a polynomial time algorithm, our goal is to find an FPT algorithm that produces an approximate solution.

Examples.

We can also consider parameterization by cost. (Here we put the SPR algorithm.)

1.6 Conclusions

Finally, we refer the reader to the recently published collection of parameterized complexity survey papers in the Computer Journal [22], the monographs [36] and [28], as well as the original parameterized complexity text [25], for more extensive coverage of parameterized complexity theory and parameterized algorithms.

- [1] K. A. Abrahamson and M. R. Fellows: *Finite automata, bounded treewidth, and well-quasi-ordering*. Graph Structure Theory, editors N. Robertson and P. Seymour, Contemporary Mathematics Vol 147, American Mathematical Society, pp 539-564, 1993.
- [2] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters and C. T. Symons: *Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments*. Proceedings of the 6th ALENEX 2004, pp 62-69, SIAM 2004.
- [3] J. Alber: *Exact Algorithms for NP-hard Problems on Planar and Related Graphs: Design, Analysis, and Implementation*. PhD thesis in preparation, Universität Tübingen, Germany, 2002.
- [4] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier: *Fixed parameter algorithms for dominating set and related problems on planar graphs*. Algorithmica 33, pp 461-493, 2002.
- [5] J. Alber, H. Fernau, and R. Niedermeier: *Parameterized complexity: exponential speed-up for planar graph problems*. Proc. 28th ICALP, Springer-Verlag LNCS 2368, pp150-159, 2002.
- [6] N. Alon, R. Yuster, and U. Zwick: *Color-coding*. Journal of the ACM 42 (4), pp 844-856, 1995.
- [7] R. Balasubramanian, M. Fellows, V. Raman: *An improved fixed parameter algorithm for vertex cover*. Information Processing Letters 65 (3), pp163-168, 1998.
- [8] H. L. Bodlaender: *A linear time algorithm for finding tree decompositions of small treewidth*. SIAM J. Comput. 25, pp 1305-1317, 1996.
- [9] H. L. Bodlaender: *A partial k-arboretum of graphs with bounded treewidth*. Technical Report UU-CS-1996-02, Department of Computer Science, Utrecht University, Utrecht, 1996.
- [10] H. L. Bodlaender, R. G. Downey, M. R. Fellows, D. Hermelin: *Fixed-Parameter Tractability and Completeness VI: Foundations of Kernelization*. manuscript.

- [11] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks: *Approximating treewidth, pathwidth, and minimum elimination tree height*. J. Algorithms 18, pp 238-255, 1995.
- [12] H. L. Bodlaender and T. Kloks: *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*. J. Algorithms 21, pp 358-402, 1996.
- [13] H. L. Bodlaender, T. Kloks, and D. Kratsch: *Treewidth and pathwidth of permutation graphs*. Proceedings of the 20th International Colloquium on Automata, Languages and Programming, A. Lingas, R. Karlsson, and S. Carlsson (Eds.), Vol 700 LNCS, Springer-Verlag, pp 114-125, 1993.
- [14] H. L. Bodlaender and R. H. Möhring: *The pathwidth and treewidth of cographs*. SIAM J. Disc. Meth. 6, pp 181-188, 1993.
- [15] R. B. Borie, R. G. Parker and C. A Tovey: *Solving Problems on Recursively Constructed Graphs*. to appear.
- [16] S. Buss: private communication, 1989.
- [17] L. Cai and X. Huang: *Fixed-parameter approximation: conceptual framework and approximability results*. Proceedings of IWPEC 2006, Lecture Notes in Computer Science 4169, pp 96-108, 2006.
- [18] J. Chen, I.A. Kanj and W. Jia: *Vertex Cover: Further Observations and Further Improvements*. Journal of Algorithms 41, pp 280-301, 2001.
- [19] Y. Chen, M. Grohe and M. Gruber: *On parameterized approximability*. Proceedings of IWPEC 2006, Lecture Notes in Computer Science 4169, pp 109-120, 2006.
- [20] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege and P. Taillon: *Solving large FPT problems on coarse-grained parallel machines*. Journal of Computer and System Sciences 67, pp 691-706, 2003.
- [21] B. Chor, M. R. Fellows, and D. W. Juedes: *Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps*. Proceedings of the 30th WG, LNCS 3353, pp 257 - 269, 2004.
- [22] The Computer Journal, Special Issue in Parameterized Complexity, in press.
- [23] B. Courcelle: *The monadic second-order logic of graphs I: Recognizable sets of finite graphs*. Information and Computation 85, pp 12-75, 1990.
- [24] B. Courcelle, J. Engelfriet and G. Rozenberg: *Handle-Rewriting Hypergraph Grammars*. Journal of Computing and Systems Sciences 46(2), pp 218-270, 1993.
- [25] R. G. Downey and M. R. Fellows: *Parameterized Complexity* Springer-Verlag, 1999.

- [26] R. G. Downey, M. R. Fellows and C. McCartin: *Parameterized Approximation Algorithms*. Proceedings of IWPEC 2006, Lecture Notes in Computer Science 4169, pp 121-129, 2006.
- [27] F. Dehne, A. Rau-Chaplin, U. Stege and P. Taillon: *Solving Large FPT Problems on Coarse Grained Parallel Machines*. manuscript, 2001.
- [28] J. Flum and M. Grohe: *Parameterized Complexity Theory*. Springer, 2006.
- [29] M. Frick and M. Grohe: *Deciding First-Order Properties of Locally Tree-Decomposable Graphs*. Proceedings of the 26th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 1644, pp 331-340, Springer-Verlag, 1999.
- [30] M. Hallett and C. McCartin: *A Faster FPT Algorithm for the Maximum Agreement Forest Problem*. Theory of Computing Systems 41 (3), 2007.
- [31] F. Huffner, R. Niedermeier and S. Wernicke: *Techniques for practical fixed-parameter algorithms*. to appear in The Computer Journal, Oxford University Press, 2007.
- [32] J. Gramm, R. Niedermeier and P. Rossmanith: *Fixed-parameter algorithms for Closest String and related problems*. Algorithmica 37, pp 25-42, 2003.
- [33] M. Mahajan and V. Raman: *Parameterizing Above the Guarantee: MAXSAT and MAXCUT*. Journal of Algorithms 31, pp 335-354, 1999.
- [34] D. Marx: *Parameterized Complexity and Approximation Algorithms*. to appear in The Computer Journal, Oxford University Press, 2007.
- [35] G. L. Nemhauser and L. E. Trotter Jr: *Vertex packings: Structural properties and algorithms*. Mathematical Programming 8, pp 232-248, 1975.
- [36] R. Niedermeier: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- [37] R. Niedermeier and P. Rossmanith: *A general method to speed up fixed-parameter tractable algorithms*. Information Processing Letters 73, pp 125-129, 2000.
- [38] R. Niedermeier: *Invitation to fixed-parameter algorithms*. Oxford University press, 2006.
- [39] S.-I. Oum: *Approximating rank-width and clique-width quickly*. Proceedings of 31st International Workshop on Graph-Theoretic Concepts in Computer Science, pp 49-58, 2005.
- [40] L. Perkovic and B. Reed: *An Improved Algorithm for Finding Tree Decompositions of Small Width*. International Journal of Foundations of Computer Science 11 (3), pp 365-371, 2000.
- [41] B. Reed, K. Smith and A. Vetta: *Finding odd cycle transversals*. Operations Research Letters 32, pp 299-301, 2004.

- [42] N. Robertson and P. D. Seymour: *Graph minors - a survey*. Surveys in Combinatorics, I. Anderson (Ed.), Cambridge Univ. Press, pp 153-171, 1985.
- [43] N. Robertson and P. D. Seymour: *Graph minors II. Algorithmic aspects of tree-width*. Journal of Algorithms 7, pp 309-322, 1986.
- [44] N. Robertson and P. D. Seymour: *Graph minors I - XV*. appearing in J. Comb. Theory Series B, 1983 - 1996.
- [45] D. Seese: *The structure of models of decidable monadic theories of graphs*. Ann. Pure and Appl. Logic, Vol 53, pp 169-195, 1991.
- [46] C. Sloper and J. A. Telle: *An overview of techniques for designing parameterized algorithms*. to appear in The Computer Journal, Oxford University Press, 2007.
- [47] U. Stege: *Resolving Conflicts in Problems in Computational Biochemistry*. Ph.D. dissertation, ETH, 2000.
- [48] K. Weihe: *Covering trains by stations or the power of data reduction*. Proceedings of Algorithms and Experiments (ALEX98), R. Battiti and A. A. Bertossi (Eds.), pp 1-8, 1998.