Parameterized Algorithmics: A Graph-Theoretic Approach

Henning Fernau
Universität Tübingen, WSI für Informatik, Sand 13,
72076 Tübingen, Germany
`fernau@informatik.uni-tuebingen.de`

University of Newcastle, School of Electr. Eng. & Computer Sci.,
University Drive, Callaghan, NSW 2308, Australia

April 20, 2005

Contents

Chapter 1

Prologue

This Habilitationsschrift can be read in various ways by different people.

- People working in the area of parameterized complexity can hopefully find main ideas leading to *efficient* parameterized algorithms presented in a new way. This may spark new research, hopefully leading to more and more useful examples of parameterized algorithms.
- Practitioners having to cope with computationally hard problems will find an exposition how they can cast their everyday heuristic methodologies into the framework of parameterized algorithmics. They will be hopefully able to write up their own parameterized analysis of their heuristics, and working in this somewhat more abstract framework will then help them introduce improvements in their algorithms.

By addressing two audiences, we also express the hope that both "worlds" may come together to share their experiences. Especially, we are convinced that if more and more heuristic ideas are made public (often enough, we fear that such "straightforward ideas" are considered not to be publishable) in order to spike their mathematical in-depth analysis in the parameterized framework. This way, mathematicians and algorithm theorists can help explain why in many cases simple heuristics do work well on concrete instances of computationally hard problems.

1.1 Why practitioners may wish to continue reading

First, I should at least try to describe what "practitioner" may mean in this context. In fact, there are different possible interpretations:

- A programmer should benefit from the fact that this Habilitationsschrift pays special attention to presenting algorithms in a unifying way by means of pseudo-code. Those program fragments should be relatively easily translatable into real programs.

  The reason for being that explicit in the presentation of the algorithms (and this also contrasts with many other research papers and monographs on algorithmics) is that we are often under the impression that the typical implicit presentation of algorithms, cast within the scheme of a proof, often only indicating "other similar cases" along the mathematical argument, is not helpful for the propagation of the often clever algorithmic ideas. We hope that this style of the Habilitationsschrift helps proliferate the ideas of parameterized algorithmics into real-world programs, a step that is mostly still lacking (as is the fate also for many clever algorithmic ideas according to our experience and according to talks with people that write "everyday pieces" of software).

- A user might get some insights what is currently possible in terms of algorithm development.

  By giving appropriate feedback, software companies might use more and more of the techniques presented in this Habilitationsschrift (and also other books on algorithmics) to improve their products.

- A manager might have found himself / herself a couple of times in the situation sketched in [199, page 3], where some of his employees excuse themselves by pointing to the fact that

  > "I can't find an efficient algorithm, but neither can all these famous people."

  This Habilitationsschrift shows that sometimes there might be a way out (without relying on possibly better known techniques like approximation, randomization or pure heuristics) if one has to cope with a combinatorially hard problem: namely parameterized algorithmics.

If any practitioner finds that his / her area of interest in underrepresented in this Habilitationsschrift, please bear two things in mind:

- the graph theoretical problems we chose to tackle can be often find in disguise in applications (sometimes, this will be hinted at throughout the Habilitationsschrift);

- the concrete applications that we describe are due to personal experience based on personal contacts; different contacts and backgrounds would have probably led to different case studies.

However, I would warmly welcome any concrete pointers to problem classes that might be suitable to the presented approach.

1.2 Standard notations

Although we do present all necessary notions in this Habilitationsschrift, let us utter a cautious caveat: standard notions from elementary set theory and logic will be used without further explanation; however, their explanation should be obtainable from practically any textbook on elementary mathematics. Since some denotations sometimes vary in different books, we review them in what follows:

- $\mathbb{N}$ denotes the set of natural numbers, including zero.

finally check for consistent use

- $\mathbb{R}$ denotes the set of real numbers.
  $\mathbb{R}_{\geq 1} = \{x \in \mathbb{R} \mid x \geq 1\}$.
  $\mathbb{R}_{>0} = \{x \in \mathbb{R} \mid x > 0\}$.

Of crucial importance for the understanding of the running time estimates given in this Habilitationsschrift are the following (again rather standard) notations.

We write "an algorithm A has running time $\mathcal{O}(f(n))$" if

- a and b are constants independent of n,
- n is a (specified) way of measuring the size of an input instance,
- $f : \mathbb{N} \to \mathbb{R}_{>0}$,
- I is an arbitrary input of size n, and
- A uses $t \leq af(n) + b$ time steps on a standard *random access machine (RAM)* when given input I.

We will not actually specify what a RAM is; a definition can be found in any standard text in algorithmic complexity. For most purposes, it is enough to envisage a RAM as being your favorite desktop or laptop computer. Technology will only influence the constants a and b.

This is the core of the well-known concept of *$\mathcal{O}$-notation* in algorithmics. Observe that the constants a and b suppressed in this notation may play two different roles:

- They may hide specifics of the hardware technology; this hiding is wanted in order to be able to compare different algorithms (rather than different hardware technologies).
- They may also hide peculiar points of the algorithmics. In the algorithms that are described in this Habilitationsschrift, this second point will not be important, either. However, notice that there are some elements in the development of exact algorithms for hard problems where these constants do play a big role. This is true in particular for algorithms directly taken from the graph minor approach.[1] From a practical point of view, large constants due to the algorithms themselves are kind of malicious, so they should not be hidden by the $\mathcal{O}$-notation.

From time to time we will also use the related $\Omega(\cdot)-$ and $o(\cdot)-$notations. Since they are not at the core of this Habilitationsschrift, we refrain from giving detailed definitions here.

For exact algorithms of hard problems (that seemingly inherently have superpolynomial running times), a different notation has been established.

We write "an algorithm A has running time $\mathcal{O}^*(f(n))$" if

- p is a polynomial dependent on n,
- $f : \mathbb{N} \to \mathbb{R}_{>0}$ is a superpolynomial function (usually, an exponential function), and
- A has running time $\mathcal{O}(f(n)p(n))$.

Observe that the *$\mathcal{O}^*$-notation* not only hides additive and multiplicative constants, but also polynomials. The reason is that within exponential-time algorithmics, finally the exponential functions that estimate the running times of the algorithms will dominate any polynomial. However, the caveat formulated above is even stronger in this context: Large-degree polynomials might dominate exponential functions for all practically relevant input values. The astute reader will notice that the polynomials that are suppressed in the formulation of the run time estimates of our algorithms (when given in $\mathcal{O}^*$-notation) are all of low degree, which justifies the use of this shorthand notation.

1.3 What mathematicians can find

Well, it depends what kind of mathematician you are. Let me offer you at least some possible answers.

[1]We won't give details of this approach here but rather refer to [134].

- Throughout the Habilitationsschrift, you will find many examples of notions that were originally developed for good theoretical reasons, but later also revealed their usefulness for the development of efficient algorithms. This should give a good stimulus for theoreticians to continue developing mathematically nice and elegant notions, although even the purest theoretical ideas often run the risk of getting applied sooner or later. In the area of graph theory (where most examples of this Habilitationsschrift are drawn from), particular such examples are the notions of treewidth and pathwidth, which originated in the deep mathematical theory of graph minors, as primarily developed by Robertson and Seymour.

- Besides developing good definitions, proving good theorems is of course what mathematics is after. Again, there are numerous examples of deep theorems that are used as kind of subroutines in this Habilitationsschrift. Some of them are well known as the Four Color Theorem for planar graphs established by Appel and Haken, other less well known (in fact, we use a couple of coloring-type theorems at various places). Funny enough, the co-author of one of these theorems complained to us that they did not get properly published their result, since referees thought it would not be of too much interest...

  Again, this should be a stimulus for mathematicians to continue proving nice theorems.

- In fact, although for some of the pure mathematicians this might be a sort of detour, it would be also nice if some emphasis could be put on computability aspects of mathematical proofs. Often, proofs can be actually read as algorithms, but I suspect that programmers would have hard times deciphering these algorithms. Putting some more effort in this direction would surely also increase the impact of certain papers, especially if possible applications are sketched or at least mentioned.

- Finally, and not in the least, mathematicians are and should be problem solvers by profession. You will find lots of examples to pursue your research upon reading this Habilitationsschrift. Please report your solutions to concrete problems or results inspired by this text to me; I would be really interested in such results.

1.4 Messages to algorithm developers

In fact, and possibly a bit contradictory to what has been said before, there is a third possible audience: algorithm developers.

In fact, this type of audience has a kind of bridging function between the two audiences described before, so that some of the comments we have given before apply here, as well.

However, there are some peculiar things to observe for algorithm developers when reading this Habilitationsschrift:

- Good mathematical knowledge in the specific area for which algorithms are to be developed is often beneficial to the development and (maybe even more) to the analysis of (parameterized) algorithms.

- Real-world problems should be cast into a suitable model expressed in mathematical terms. Often, graph theory is the key of developing such a model.

- The status of a combinatorially hard problem in terms of its practical solvability with the help of parameterized algorithms often depends on the choice of the parameter. Here, a "good" parameter tends to be small in practice. So, the choice of a good parameter can only be determined in close partnership with the actual programmers and program testers, as well as with the (potential) users. As P. Moscato once told us (personal communication), referring to a specific problem of interest in the context of computational biology:

  > "Whenever I talk with a biologist about this problem, I walk away with a new parameter."

  It is of utmost importance to keep the possible parameter choices in mind; because if for one parameterization a certain problem turns out to be "intractable" in a technical sense, it might be tractable with another parameterization. This is one of the reasons why the "downsides" of parameterized algorithmics are not the core of this Habilitationsschrift: hard problems may turn out to be easy under different parameterizations.

- In order to actually build a bridge between theory and practice, algorithm developers should not be content with having classified a problem as "tractable" (here, this would mean "parameterized tractable"); rather, they should strive to get really good bounds on the running times of their algorithms. The better the running times of the algorithms, the more useful are the proposed algorithms. This is true in particular for the development of exact algorithms for computationally hard problems: the difference between an $\mathcal{O}^*(2^k)$ algorithm and an $\mathcal{O}^*(1.3^k)$ algorithm is tremendous.

- Conversely, it does not make much sense to strive for smaller and smaller bases of the exponential terms if this comes at the cost of very intricate algorithms; rather, one should see if there are actually simple algorithms which are easy to implement and clearly structured and that still have good running times. This would surely help proliferate ideas in the area of exact algorithmics in general, and in the fixed-parameter area in particular.

1.5 Acknowledgments

This Habilitationsschrift would not have come into "life" without the steady support from my family, friends and colleagues.

In particular, I like to mention (and thank in this way) for many scientific discussions the following people (in alphabetical order):

Faisal Abu-Khzam, Jochen Alber, Regina Barretta, Hans Bodlaender, Oleg Borodin, Ljiljana Branković, David Bryant, Frank Dehne, Frederic Dorn, Vida Dujmović, Mike Fellows, Jens Gramm, Jiong Guo, Torben Hagerup, Falk Hüffner, David Juedes, Iyad Kanj, Michael Kaufmann, Ton Kloks, Klaus-Jörn Lange, Mike Langston, Margaret Mitchell, Pablo Moscato, Rolf Niedermeier, Naomi Nishimura, Ljubomir Perkovic, Elena Prieto-Rodriguez, Prabhakar Ragde, Klaus Reinhardt, Frances Rosamond, Peter Shaw, Christian Sloper, Ulrike Stege, Matthew Suderman, Bjarne Toft, Magnus Wahlström, Sue Whitesides, and Gerhard Woeginger.

Apologies to everybody whom I forgot to mention in that list.

1.6 Keeping up to date

In a fast-growing field as the one of parameterized algorithmics, or more generally speaking, parameterized complexity and algorithms to denote the whole area, it is close to impossible to keep up to date with recent developments. So, our apologies to to everybody whose work is underrepresented if not completely missed out in this Habilitationsschrift.

The interested reader is advised to follow up overview articles that seem to appear every second month on this area. We list the survey papers we came across that only appeared in the second half of 2004:

- R. Niedermeier gave an invited talk at MFCS in August [307]. In this context, his Habilitationsschrift [306] is also worth mentioning.
- J. Flum and M. Grohe gave a nice overview on recent developments of parameterized complexity in the complexity theory column of the Bulletin of the EATCS (October issue), see [185].
- R. Downey and C. McCartin had an invited talk at DLT in December [138].

We therefore recommend trying to keep up to date by following up recent conferences and to simply search the internet with appropriate catchwords from time to time. The special chapter 11 collecting web addresses can be seen as a starting point for such a research.

The internet is also a good place to look for sources on algorithmics, for looking up definitions etc., see Site [Chapter 11, Site 1].

However, one should not completely disregard "older" overview articles. For example, the papers [137, 136] still contain lots of programmatic material that hasn't been properly dealt with in the last couple of years.

Chapter 2

Introduction

This introductory chapter is meant to introduce the basic notions used in this Habilitationsschrift:

- Sec. 2.1 will acquaint the reader with the basic notions of parameterized complexity and algorithmics, since this is the basic framework for our work.
- Sec. 2.3 provides a primer in graph theory, since most problems we approach in this Habilitationsschrift are drawn from that area (or can be conveniently expressed in those terms).
- Then, in Sec. 2.4 we make this more concrete by exhibiting some of our favorite combinatorial graph problems.
- Sec. 2.5 is meant to be an appetizer to show one of the main techniques of this area: that of data reduction.

2.1 The parameterized landscape

Parameterized complexity has nowadays become a standard way of dealing with computationally hard problems.

2.1.1 The world of parameterized algorithmics: definitions

In this section, we are going to provide the basic definitions of parameterized algorithmics.

Definition 2.1 (parameterized problem) A *parameterized problem* $\mathcal{P}$ is a usual decision problem together with a special entity called *parameter*. Formally, this means that the language of YES-*instance*s of $\mathcal{P}$, written $L(\mathcal{P})$, is a subset of $\Sigma^* \times \mathbb{N}$. An *instance* of a parameterized problem $\mathcal{P}$ is therefore a pair $(I, k) \in \Sigma^* \times \mathbb{N}$.

If $\mathcal{P}$ is a parameterized problem with $L(\mathcal{P}) \subseteq \Sigma^* \times \mathbb{N}$ and $\{a, b\} \subseteq \Sigma$, then $L_c(\mathcal{P}) = \{Iab^k \mid (I, k) \in L(\mathcal{P})\}$ is the *classical language* associated to $\mathcal{P}$. So, we can also speak about $\mathcal{NP}$-hardness of a parameterized problem.

As classical complexity theory, it helps classify problems as "nice," or more technically speaking, as *tractable*, or as "bad," or *intractable*, although these notions may be misleading to a practical approach, since intractable does not imply that such problems cannot be solved in practice. It rather means that there are really bad instances that can be constructed on which any conceivable algorithm would badly perform. It might also indicate that the current choice of parameter is not the "right" one.

Since this Habilitationsschrift is focusing on the algorithmic aspects, let us in the first place define the class of problems that we consider to be tractable from a parameterized perspective.

Another thing that is worth mentioning is the way how the complexity of an algorithm is mentioned. In general, we assume a suitable underlying RAM model. Since the technical details are not really crucial in our setting, we deliberately refrain from giving more details here, as they are contained in any textbook on classical complexity theory or algorithmics. Rather, we assume an intuitive understanding of what an algorithm is and how the complexity of an algorithm is measured. Hence, we will give all algorithms in a high-level pseudo-code notation.

When measuring the complexity of an algorithm, it is crucial against what we measure, i.e., how we measure the *size* of the input. We already discussed this issue for "number parameters." In most cases, a rather intuitive understanding of size is sufficient, assuming a suitable encoding of the instance as a binary encoded string and taking the length of this string as the size of the instance. Given I, $|I|$ should denote this size measure. However, in some cases, other size functions make sense, as well. For examples, graphs are usually measured in terms of the number of edges or in terms of the number of vertices. In classical complexity, this does not really matter as long as we only distinguish between say polynomial time and exponential time. However, these things may become crucial for algorithmics that deals with algorithms that run in exponential time. Therefore, at some times we will be more picky and explicitly refer to a size function $\text{size}(\cdot)$ to measure the size $\text{size}(I)$ of the instance I. More details are discussed in Chapter 3. Some of the following definitions already make use of this function. At this point, it is sufficient to always think about size(·) as referring to $|\cdot|$, i.e., the length of a binary string encoding the instance, where a "reasonable encoding" function is assumed.

Definition 2.2 (parameterized tractability) A parameterized problem $\mathcal{P}$ is called *fixed-parameter tractable* if there exists a solving algorithm for $\mathcal{P}$ running in time $\mathcal{O}(f(k)p(|I|))$ on instance (I, k) for some function f and some polynomial p, i.e., the question if $(I, k) \in L(\mathcal{P})$ or not can be decided in time $\mathcal{O}(f(k)p(|I|))$.

The class of problems collecting all fixed-parameter tractable parameterized problems is called $\mathcal{FPT}$.

A parameter can, in principle, be nearly everything, as detailed in Chapter 3. However, in most examples we consider, the parameter will be a number. From a theoretical point of view, it does not matter whether such a number is given in unary or in binary to classify a problem in $\mathcal{FPT}$. However, this would of course matter for the algorithmics, since the size of a number considerably differs depending on the encoding. Therefore, let us fix here that numbers that appear as parameters will be considered as unary-encoded if not stated otherwise. Hence, the size of the number would correspond to its numerical value. This can be justified, since in most cases (say for graph problems) the parameter (say the size of a selection of vertices) is upper-bounded by the size of a list of items that is explicitly part of the input. As can be seen above, we already tailored our definitions to the unary encoding of the "number parameter."

Assuming some basic knowledge of graph theory on side of the reader (who might otherwise first wish to browse through Sec. 2.3), we illustrate the notions with shortly discussing one example in this section:

Problem name: VERTEX COVER (VC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $|C| \leq k$?

Definition 2.3 (Kernelization) Let $\mathcal{P}$ be a parameterized problem. A *kernelization* is a function K that is computable in polynomial time and maps an instance (I, k) of $\mathcal{P}$ onto an instance (I', k') of $\mathcal{P}$ such that

- (I, k) is a YES-instance of $\mathcal{P}$ if and only if (I', k') is a YES-instance of $\mathcal{P}$
- $\text{size}(I') \leq f(k)$, and

- $k' \leq g(k)$ for some arbitrary functions f and g.

To underpin the algorithmic nature of a kernelization, K may be referred to as a *kernelization reduction.* (I', k') is also called the *kernel* (of I), and size(I') the *kernel size.* Of special interest are *polynomial-size kernels* and *linear-size kernels*, where f is a polynomial or a linear function, respectively.

A kernelization is a *proper kernelization* if $g(k) \leq k$.

A parameterized problem that admits a kernelization is also called *kernelizable.*

Of course, of uttermost importance are proper kernelizations yielding linear-size kernels. In fact, this is the best we can hope for when dealing with hard problems, since sub-linear kernels for $\mathcal{NP}$-hard problems would mean that $\mathcal{P}$ equals $\mathcal{NP}$, see Lemma 9.11.

With one exception, all kernelization reduction that are presented in this Habilitationsschrift are proper.

Once a kernelization is found, membership of the corresponding problem in $\mathcal{FPT}$ is easy to see:

Algorithm 1 A brute force $\mathcal{FPT}$ algorithm from kernelization

Input(s): kernelization function K, a brute-force solving algorithm A for $\mathcal{P}$, instance (I, k) of $\mathcal{P}$

Output(s): solve (I, k) in $\mathcal{FPT}$-time

```
  Compute kernel $(I', k') = K(I, k)$
  Solve $(I', k')$ by brute force, i.e., return $A(I', k')$.
```

Interestingly, the converse is also true: each problem in $\mathcal{FPT}$ is kernelizable. The corresponding construction is contained in [134] and is not reproduced, since it is not of any algorithmic interest.

Theorem 2.4 *A parameterized problem is in $\mathcal{FPT}$ iff it is kernelizable.*

In the rest of this Habilitationsschrift, we are mostly sloppy when it comes to the issue of defining kernelizations. The issue of *reduction rule*s and how they lead to kernelizations is discussed in Sec. 4.1.

Secondly, we will also accept algorithms as valid kernelizations if they actually completely solve or even reject the given instance. In our algorithms, we denote this behavior by, e.g.,

if . . . **then** YES

Formally, an algorithm that contains such statements can be interpreted as a kernelization function by first choosing some small YES- (or NO-) instances in the beginning and outputting them whenever the algorithm detects that the given (or transformed) instance can be trivially solved or rejected. Particular examples for such forms of reductions can be found by (ab)using (non-trivial) theorems combinatorics, as listed in Sec. 4.4.

Finally, and possibly even more sloppy, we write down reduction rules in the form:

> Let (I, k) be an instance of problem P. If conditions ... are met, then reduce the instance to (I', k') with $k' < k$.

What we sweep under the carpet is that it might be that k' might have a value that is not permitted; e.g., often the parameter is a positive integer, and therefore $k' < 0$ would not be permitted. So, whenever a reduction rule application would yield an instance that has a parameter which is not permitted, we would implicitly return NO, in the sense described in the previous paragraph.

Let us state two simple reduction rules for VERTEX COVER (as the running example not only of this section) that are usually attributed to S. Buss: [1]

Reduction rule 1 *Delete isolated vertices (and leave the parameter unchanged).*

Reduction rule 2 (*Buss' rule*) *If v is a vertex of degree greater than k in the given graph instance (G, k), then delete v from the instance and reduce the parameter by one, i.e., produce the instance $(G - v, k - 1)$.*[2]

The soundness of these rules can be "easily" seen by the following observations.

- If v is a vertex with no neighbors, v can be removed from the graph, since v will not be part of any <u>minimum</u> vertex cover.

[1]Let us mention the following historical aside: Although the two reduction rules (in particular, the second one) listed in the following are generally attributed to a personal communication of Sam Buss, in particular [64], there is a reference of Evans [158] that considerably predates the Buss reference. Admittedly, Evans considers a special variant of vertex cover (namely, the CONSTRAINT BIPARTITE VERTEX COVERproblem discussed in detail in Chap. 5) which arises in connection with VLSI reconfiguration, but the reduction rules are basically the same.

[2]$G - v$ denotes the graph that is obtained from $G = (V, E)$ by removing v from V and by removing all edges that contain v. If V_ℓ is a vertex set $V_\ell = \{v_1, \dots, v_\ell\}$, $G - V_\ell$ is inductively defined: $G - V_1 = G - v_1$, $G - V_i = (G - V_{i-1}) - v_i$ for $i > 1$.

- If v is a vertex of degree greater than k, v must be in any vertex cover, since otherwise all neighbors would be in the cover, which is not feasible, because we are looking for vertex covers with at most k vertices. Hence, we can remove v from the graph.

More precisely, to actually prove the soundness of the reduction rules, the following has to be shown (since the polynomial-time computability of the rules is trivial to see):

Lemma 2.5 *Let (G,k) be an instance of* VERTEX COVER *that contains isolated vertices I. Then, (G,k) is a* YES*-instance of* VC *iff $(G-I,k)$ is a* YES*-instance of* VC*.*

Lemma 2.6 *Let (G,k) be an instance of* VERTEX COVER *that contains a vertex v of degree larger than k. Then, (G,k) is a* YES*-instance of* VC *iff $(G-v,k-1)$ is a* YES*-instance of* VC*.*

To get acquainted with the proof strategy involved in this kind of reasoning, let us formally prove the soundness of Buss' rule:

Proof. If (G,k) is a YES-instance of VERTEX COVER that contains a vertex v of degree larger than k, then v must be part of any solution; hence, $(G-v,k-1)$ is a YES-instance of VC.

Conversely, if C' is a cover verifying that $(G-v,k-1)$ is a YES-instance of VC, then $C=C'\cup\{v\}$ shows that (G,k) is also a YES-instance of VC; this is true irrespectively of the degree of v. ∎

The very idea of kernelization (as detailed in Chap. 4) now means to apply the available reduction rules to a given problem instance *as long as possible.* This leaves us finally with a *reduced instance*, as discussed in the following theorem.

Theorem 2.7 *A* YES*-instance (G,k), with $G=(V,E)$, of* VERTEX COVER *to which neither Rule 1 nor Rule 2 is applicable, satisfies $|E|\leq k^2$ and $|V|<k^2$.*

Proof. Due to Rule 2, every vertex has degree bounded by k. Since (G,k) is a YES-instance, there is a selection of k vertices that cover all edges E. But each vertex can cover at most k edges, so that $|E|\leq k^2$ follows. Since Rule 1 is not applicable, the number of vertices of G is basically bounded by the number of edges of the graph. ∎

Theorem 2.7 allows to state the following kernelization rule in the form of a little program 2; observe that the ELSIF-branch codifies a third reduction rule whose correctness follows from the preceding theorem. Theorem 2.7 then also shows the kernel size that is claimed in Alg. 2.

Algorithm 2 A kernelization algorithm for VERTEX COVER, called Buss-kernel

Input(s): A VERTEX COVER instance (G, k)
Output(s): an instance (G', k') with $k' \leq k$, $|E(G')|, |V(G')| \leq (k')^2$, such that (G, k) is a YES-instance of VC iff (G', k') is a YES-instance

```
if possible then
  Apply Rule 1 or Rule 2; producing instance $(G', k')$.
  return Buss-kernel$(G', k')$.
else if $|E(G)| > k^2$ then
  return $((\{x, y\}, \{\{x, y\}\}), 0)$ {encoding NO}
else
  return $(G, k)$
end if
```

To explicitly show that VERTEX COVER belongs to $\mathcal{FPT}$, we can consider Alg. 3. This allows us to state:

Algorithm 3 A kernelization algorithm for VERTEX COVER, called VC-kernelization-based

Input(s): A VERTEX COVER instance (G, k)
Output(s): YES iff (G, k) is a YES-instance of VC.

```
Let $(G', k')$:=Buss-kernel$(G, k)$. Let $G' = (V, E)$.
if $k' \leq 0$ then
  return $(k' = 0$ AND $E = \emptyset)$
else
  for all $C \subseteq V$, $|C| = k$ do
    if $C$ is a vertex cover of $G'$ then
      return YES
    end if
  end for
  return NO
end if
```

Theorem 2.8 *$VC \in \mathcal{FPT}$. More specifically, given an instance (G, k) of* VERTEX COVER*, the question "$(G, k) \in L(VC)$?" can be answered in time*

$\mathcal{O}(f(k) + |V(G)|)$, where

$$f(k) = \binom{k^2}{k} \in \mathcal{O}(2^{k^2}).$$

Proof. The correctness is immediate by Theorem 2.7. That theorem also shows the claimed bound on the running time. Observe that the reduction rules themselves can be efficiently implemented. ∎

After having introduced the essential concepts of this Habilitationsschrift, let us formulate a warning when it comes to interpreting results that are stated without explicit reference to the "world" of parameterized complexity. For example, the following can be found in a recent paper dedicated to specific aspects of graph theory [360, page 338]:

> Garey and Johnson [199] proved that testing $CR(G) \leq k$ is $\mathcal{NP}$-complete, ... Testing planarity, and therefore testing $CR(G) \leq k$ for any fixed k can be done in polynomial time—introduce at most k new vertices for crossing points in all possible ways and test planarity.

More specifically, $CR(G)$ is the crossing number of a graph G, i.e., the number of edge crossing necessarily incurred when embedding G into the Euclidean plane;[3] it is however not necessary to understand the exact definition of this notion for the warning that we like to express. Namely, when superficially read, the quoted sentence could be interpreted as showing fixed-parameter tractability of the following problem:

Problem name: CROSSING NUMBER (CRN)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is $CR(G) \leq k$?

However, this is not a correct interpretation. Namely, how would you interpret or even implement the algorithm sketch in more concrete terms? Our interpretation of the quoted lines would be as detailed in Alg. 4. However, the complexity of this algorithm is roughly $\mathcal{O}(|G|^k)$, and hence this algorithm does *not* show that CROSSING NUMBER is fixed-parameter tractable.

[3]There seem to be some intricate problems with the exact definition of what is meant by a crossing number; papers on this topic should be read with careful scrutiny to see what definitions the authors adhere to. This is detailed in the paper of Szèkély [360], as well as in [317]. In fact, the interpretation we detail on the quoted algorithm sketch for CROSSING NUMBER is referring to the so-called *pairwise crossing number*.

Algorithm 4 Determining pairwise crossing numbers

```
Input(s): Graph $G = (V, E)$, parameter $k$
Output(s): Is $CR(G) \leq k$?
  for all sets of $k$ pairs of edges $P = \{\{e_1, e'_1\}, \dots, \{e_k, e'_k\}\}$ do
    Initialize $G' = (V', E')$ as $G$.
    for $i = 1, \dots, k$ do
      Into $G'$, we introduce one new vertex $x_i$ and new edges $f_i, g_i, f'_i, g'_i$
      that contain $x_i$ as one endpoint and that satisfy $|f_i \cap e_i| = |g_i \cap e_i| =$
      $|f'_i \cap e'_i| = |g'_i \cap e'_i| = 1$.
      Mark $e_i$ and $e'_i$.
    end for
    Remove all marked edges from $G'$.
    if $G'$ can be drawn without crossings into the plane (planarity test)
    then
      return YES
    end if
  end for
  return NO.
```

In fact, to our knowledge, it is open if CROSSING NUMBER is parameterized tractable. Only if the input is restricted to graphs of maximum degree three, membership in $\mathcal{FPT}$ has been established, see [134, page 444].

Conversely, this caveat does not mean that—albeit it has become sort of standard to refer to parameterized complexity when stating exact algorithms—all results that do not reference the parameterized paradigm are necessarily not interpretable as results in parameterized algorithmics. A good (positive) and recent example is the tree editing problems investigated in [198], which show that these problems are in $\mathcal{FPT}$ when parameterized by the number of admissible edit operations (which is also the standard parameter following the terminology introduced in Chap. 3).

2.1.2 The world of parameterized algorithmics: methodologies

A whole toolbox for developing these algorithms has been developed. Marx (in his recent talk at CCC 2004) compared these tools with a Swiss army knife, see [Chapter 11, Site 12].

Those tools are, in particular (according to the mentioned slides):

1. well-quasi-orderings;

2. graph minor theorems;
3. color-coding;
4. tree-width, branch-width etc.;
5. kernelization rules;
6. bounded search tree.

This list is probably not exhaustive but gives some good impression of the available techniques. This list is ordered by "increasing applicability," as will be explained in the following.

The first two methods are the most abstract tools. In a certain sense, they are useful to classify problems as being tractable by the parameterized methodology, but (as far as we know) they never led to algorithms that can be really termed useful from a practical point of view. Once a problem has been classified as tractable, a good research direction is to try to improve the corresponding algorithms by using the more "practical" methods.

Color coding is a methodology that is helpful for getting *randomized* parameterized algorithms. Since this is not the primary interest of this Habilitationsschrift, we will neglect this methodology, as well.

The last three methods are of definite practical interest. They comprise the methods which yielded the best parameterized algorithms in nearly all cases. They will be in the focus of the exposition of this Habilitationsschrift. To each of these three methodologies, a chapter of the Habilitationsschrift will be devoted.

More specifically, treewidth and branchwidth are typical examples of structural parameters for graphs that are particularly helpful in the sense that once it is known that a graph has a small treewidth, then problems that are $\mathcal{NP}$-hard for general graphs can be solved in polynomial, often even in linear time. We will give details in Chapter 7.

Kernelization rules can be viewed as a method of analyzing data reduction rules. These have been always successfully used in practice for solving huge data instances, but parameterized algorithmics offers now a way to tell why these rules actually work that well. Further introductory remarks can already be found in this chapter. Note that the very first theorem of this Habilitationsschrift, which is Thm. 2.4, says that the basic algorithmic class we are dealing with can be characterized by algorithms that use kernelization rules (although the rules leading to that theorem are admittedly of artificial nature). A worked-out example of this technique is contained in Sec. 2.5. Further details on this technique are collected in Chapter 4.

Finally, exact algorithms for computationally hard problems are often if not mostly based on a clever search through the space of all possibilities. This naturally leads to the notion of a *search tree*, and the art of developing search tree algorithms can be seen as devising clever rules how to work one's way through the search space. Chapter 5 collects many ideas how to find and how to improve on search tree algorithms. More examples can be found throughout the whole Habilitationsschrift.

We deliberately ignored some techniques that others would find essential to parameterized algorithmics, notably including techniques related to logic. However, at several places, the reader will find hints to further reading.

2.2 Is parameterized algorithmics the solution to "everything"?

Such a bold question cannot be possibly answered affirmatively. So, what can we find on the downside, the *parameterized complexity* theory? Let us here only briefly mention that there does exist a whole area dealing with this. We will explore more of this in Chapter 9. The reader who is primarily interested in complexity theory should be warned, however, that this is not the focus of the present Habilitationsschrift. Rather, (s)he could take Chapter 9 as a collection of pointers to further reading.

From an algorithmic point of view, the most important notion of parameterized complexity is that of a *parameterized reduction*, since it makes it possible to link seemingly different problems in a way that algorithms for one problem can be used to solve another one. Even if the idea to construct such a simple reduction fails, it often gives us good hints at how to use ideas from a "solved" problem to tackle a new problem.

Definition 2.9 (Parameterized reduction) Let $\mathcal{P}$ and $\mathcal{P}'$ be parameterized problems. Let $g : \mathbb{N} \to \mathbb{N}$ be some arbitrary function.

A *parameterized reduction* is a function r that is computable in time $\mathcal{O}(g(k)p(\text{size}(I)))$ for some polynomial p and maps an instance (I, k) of $\mathcal{P}$ onto an instance $r(I, k) = (I', k')$ of $\mathcal{P}'$ such that

- (I, k) is a YES-instance of $\mathcal{P}$ if and only if (I', k') is a YES-instance of $\mathcal{P}'$ and
- $k' \leq g(k)$.

We also say that $\mathcal{P}$ *reduces to* $\mathcal{P}'$.

Note that we defined a sort of many-one reduction. More generally speaking, it is also possible to define Turing reductions, see [134, 185]. The corresponding details won't matter in the remainder of this Habilitationsschrift.

Observe that the kernelization reduction introduced above is a sort of "self-reduction" and fits into the concept of a parameterized reduction.

Unfortunately, most classical reductions between $\mathcal{NP}$-hard problems turn out not to be reductions in the parameterized sense, although there are also some exceptions, as contained in [98, 282].

Let us discuss some simple examples (drawn from graph theory; the reader who is not so familiar with the corresponding concepts might first wish to consult the primer contained in Sec. 2.3):

Problem name: CLIQUE (CQ)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *clique* $C \subseteq V$ with $|C| \geq k$?

Alternatively, we may focus on the edges:

Problem name: CLIQUE (CQE)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *edge-induced clique* $C \subseteq E$ with $|C| \geq k$?

Lemma 2.10 CLIQUE *and* CLIQUE (EDGE-INDUCED) *are parameterized interreducible.*

This simple result is also mentioned in [232] (without reference to parameterized complexity); however, since this gives a nice example for an easy parameterized reduction, we provide a proof in what follows.

Proof. Let $G = (V, E)$ be a graph. For some k, let (G, k) be an instance of CLIQUE. If (G, k) is a YES-instance, then $(G, k(k-1)/2)$ is a YES-instance for CLIQUE (EDGE-INDUCED) and vice versa.

Conversely, consider an instance (G, k) of CLIQUE (EDGE-INDUCED). If (G, k) is a YES-instance, then consider integers k', k'' such that

$$(k'-1)(k'-2)/2 < k \leq k'(k'-1)/2 =: k''.$$

Obviously, (G, k'') is a YES-instance of CLIQUE (EDGE-INDUCED), as well. Moreover, if (G, k) is a NO-instance, then (G, k'') is a NO-instance of CLIQUE (EDGE-INDUCED), as well.

Hence, (G, k) is a YES-instance of CQE iff (G, k'') is a YES-instance of CQE iff (G, k') is a YES-instance of CQ. ∎

In Chapter 9, we will be more explicit about the complexity classes (i.e., the classes of problems) that are *not* amenable to the parameterized approach. At this stage, it is sufficient to know that the lowest complexity class thought to surpass $\mathcal{FPT}$ is called W[1] (according to the exposition in the monograph [134]). All those classes are closed under parameterized reductions.

Since it is well-known that CLIQUE is W[1]-complete, we may deduce:

Corollary 2.11 CLIQUE (EDGE-INDUCED) *is W[1]-complete.*

Of course, taking the $\mathcal{NP}$-completeness of CLIQUE for granted, the proof of Lemma 2.10 also shows that CLIQUE (EDGE-INDUCED) is $\mathcal{NP}$-complete.

Now, consider the following two related problems:

Problem name: VERTEX CLIQUE COMPLEMENT COVER (VCCC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex clique complement cover* $C \subseteq V$ with $|C| \leq k$?

Here, $C \subseteq V$ is a *vertex clique complement cover* in $G = (V, E)$ iff $V - C$ induces a complete graph (i.e., a clique). Similarly, $C \subseteq E$ is a *clique complement cover* in $G = (V, E)$ iff $E - C$ induces a complete graph.

Problem name: CLIQUE COMPLEMENT COVER (CCC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *clique complement cover* $C \subseteq E$ with $|C| \leq k$?

The latter problem has been studied in [34] under the viewpoint of approximation; in actual fact, a weighted version was examined in that paper.

The following is an easy consequence from the definition (also see [232]:

Lemma 2.12
- *C is a vertex clique complement cover of $G = (V, E)$ iff $V \setminus C$ induces a clique in G iff C is a vertex cover of G^c.*
- *C is a clique complement cover of $G = (V, E)$ iff $E \setminus C$ induces a clique in G.*

However, the induced translation from say CLIQUE to VERTEX CLIQUE COMPLEMENT COVER or from CLIQUE (EDGE-INDUCED) to CLIQUE COMPLEMENT COVER (or vice versa) is *not* a parameterized reduction; in fact, the status of the problems is quite different: while VERTEX CLIQUE COMPLEMENT COVER (and CLIQUE COMPLEMENT COVER, as we will later see[4]) is in $\mathcal{FPT}$(combining Lemma 2.12 with Theorem 2.8), CQ and CQE are W[1]-complete and hence not believed to lie in $\mathcal{FPT}$.

2.3 A primer in graph theory

Most examples in this Habilitationsschrift will be basically graph theoretic questions, although not all problems (at first glance) are of graph theoretic nature. The reason is that graph theory offers a very nice framework of stating combinatorial problems of any kind.[5] Therefore, in this section, we briefly introduce some basic notions related to graphs (and hypergraphs). Reader familiar with graph theory may skip this section and return to it whenever some possibly non-familiar notions are used throughout the text; the index will help find the appropriate place of definition.

A *graph* G can be described by a pair (V, E), where V is the *vertex* set and E is the *edge* set of G. Abstractly speaking, E is a relation on V. If this relation is symmetric, then G is an *undirected graph*, otherwise (and more general) we also speak of a *directed graph*. A *loop* is an edge of the form (x, x). If an undirected graph contains no loops, its edge set can be also specified by a set of 2-element subsets of V. We therefore often use set notation to denote edges. Specifically, this is true for hypergraphs as defined below, so that undirected graphs (even those containing loops) become special cases of hypergraphs.

If not stated otherwise, graphs in this Habilitationsschrift will be undirected and without loops. Edges are then alternatively denoted as (x, y), $\{x, y\}$ or simply xy, whatever is more convenient in the concrete example.

As mentioned, undirected graphs can be generalized to hypergraphs. A *hypergraph* G is specified by a pair (V, E), where V is the vertex set and E is the *hyperedge* set, i.e., a set of subsets of V. If e is a hyperedge, $|e|$ is its *hyperedge size*. For convenience, we often refer to hyperedges simply as edges. To underline the duality of vertices and hyperedges in hypergraphs,

[4]Bar-Yehuda and Hochbaum [34, 232] indicate that CLIQUE COMPLEMENT COVER has a *direct* connection to VERTEX COVER; however, we believe that this is not true, as we will detail in Sec. 6.3.

[5]A nice selection of such "applied problems" modeled by graphs can be found in [283], although that book rather focuses on Graph Drawing than on Graph Theory as such.

we sometime also refer to the hyperedge size as *hyperedge degree.*

The following notions mostly apply both to (undirected) graphs and to hypergraphs:

- $V(G)$ denotes the *set of vertices* of G.
- $E(G)$ denotes the *set of edges* of G.
- $P(G)$ denotes the *set of paths* of G, where a *path* is a sequence of edges $p = e_1e_2 \dots e_k$ with $e_i \cap e_{i+1} \neq \emptyset$ for $i = 1, 2, \dots, k-1$; k is also referred to as the *length of the path.* A path $p = e_1e_2 \dots e_k$ is called a *simple path* if $e_i \cap e_{i+1} \cap e_j \neq \emptyset$ iff $j = i$ or $j = i+1$ for $i = 1, 2, \dots, k-1$, and $j = 1, 2, \dots, k$. Two vertices x, y are *connected* iff there is a path between them. A *connected component*, or simple said a *component*, is a set of vertices C in a graph such that each pair of vertices $x, y \in C$ is connected. A graph is called connected if it contains only one connected component.

  A simple path $p = e_1e_2 \dots e_k$ is a *cycle* iff $e_1 \cap e_k \neq \emptyset$; k is also referred to as the *length of the cycle.*

  A cycle of length k is usually abbreviated as C_k. A simple path of length k that is not a cycle is usually abbreviated as P_k.

- If G is a (hyper-)graph, then (V', E') is called a *subgraph* of G iff $V' \subseteq V(G)$ and $E' \subseteq E(G)$.

  A graph is called *cycle-free* or *acyclic* if it does not contain a cycle as a subgraph.

  A subgraph (V', E') of $G = (V, E)$ is *vertex-induced* (by V') iff

  $$\forall e \in E : e \subseteq V' \iff e \in E'.$$

  In other words, E' contains all edges between vertices in V' that are mentioned in G. We also write $G[V']$ to denote G'. A subgraph (V', E') of $G = (V, E)$ is *edge-induced* (by E') iff $V' = \bigcup_{e \in E'} e$. We also write $G[E']$ to denote G'.

- $N(v)$ collects all vertices of v that are *neighbor*s of v, i.e., $N(v) = \{u \mid \exists e \in E(G) : \{u, v\} \subseteq e\}$. $N(v)$ is also called the *open neighborhood* of v, while $N[v] := N(v) \cup \{v\}$ is the *closed neighborhood* of v. $\deg(v)$ denotes the *degree* of vertex v, i.e., $\deg(v) = |N(v)|$. If $X \subseteq V$, we can also use $N(X) = \bigcup_{x \in X} N(x)$ and $N[X] = N(X) \cup X$.
- If G is a (hyper-)graph, then a vertex set I is an *independent set* iff $I \cap N(I) \neq \emptyset$. A set $D \subseteq V$ is a *dominating set* iff $N[D] = V$.

- If G is a (hyper-)graph, then a mapping $c : V \to C$ is a *coloring* iff, for all $a \in C$, $c^1(a)$ is an independent set. A graph is *k-colorable* iff there is a coloring $c : V \to C$ with $|C| = k$. A 2-colorable graph is also called *bipartite*. Note that a graph is bipartite iff it does not contain a cycle of even length as a subgraph.

It is sometimes convenient to link graphs and (binary) matrices.

- If $G = (V, E)$ is a (directed) graph, then $A(G)$ is a binary two-dimensional matrix (called *adjacency matrix* of G) whose rows and columns are indexed (for simplicity) by V. Then, $A[x, y] = 1$ iff $(x, y) \in E$. Put it in another way, $A(G)$ is the relation matrix associated to the adjacency matrix. Observe that $A(G)$ is symmetric iff G is undirected. $A(G)$ contains ones on the main diagonal iff G contains loops.

- If $G = (V, E)$ is a hypergraph, then $I(G)$ is a binary two-dimensional matrix (called *incidence matrix* of G) whose rows are indexed by V and whose columns are indexed by E. Then, $I[x, e] = 1$ iff $x \in e$. Conversely, every binary matrix M can be interpreted as the incidence matrix of some hypergraph $G(M)$. Observe that undirected loop-free graphs have incidence matrices with the special property that, in each column, there are exactly two entries that equal one.

- If $G = (V, E)$ is a bipartite undirected graph with bipartization $V = V_1 \cup V_2$, $A(G)$ has the special form that the submatrices indexed by $V_1 \times V_1$ and by $V_2 \times V_2$ only contain zeros. Moreover, since the graph is supposed to be symmetric, the submatrix indexed by $V_1 \times V_2$ is the transposal of the submatrix indexed by $V_2 \times V_1$. Hence, the information contained in $A(G)$ can be compressed and stored in a binary matrix $A_B(G)$ which is the submatrix of $A(G)$ indexed by $V_1 \times V_2$. $A_B(G)$ is also called *bipartite adjacency matrix* of G. Observe that any binary matrix M can be interpreted as the bipartite adjacency matrix of some bipartite undirected graph $B(M)$.

The definitions we gave up to now also provide some good hints how to *represent graphs on computers*; either as *edge lists* (where to each vertex, a list of incident edges is attached) or as matrices of different types. Although we are dealing with algorithms on graphs, we won't we in general too specific about the actual data structures we use for graphs. All the data structures we mentioned (and many other conceivable "reasonable" ones) are polynomially related, so that for the exponential-time algorithms we mostly look at, it does not so much matter which representation we actually choose.

Of particular importance are certain (undirected) graph classes in what follows.[6] The possibly simplest non-trivial graph is a *tree*, i.e., a connected graph without cycles. A graph that consists of a collection of components each of which is a tree is known as a *forest*. A vertex of degree one in a tree (and more generally, in a graph) are also called a *leaf*, and any other vertex is called an *inner node*. Here, we see another convention followed in this text: a vertex in a tree is often referred to as a *node*.

For example, a graph $G = (V, E)$ is cycle-free iff V induces a forest in G or, in other words, if G is a forest. Then, a tree can be characterized as a connected forest.

More generally speaking, if $\mathcal{G}$ is a class of graphs, then a graph G is a forest of $\mathcal{G}$ iff $G = (V, E)$ can be decomposed into $G_1 = (V_1, E_1)$, $\ldots$, $G_k = (V_k, E_k)$, with $V_i \cap V_j = \emptyset$ iff $i \neq j$ and $\bigcup_{i=1}^{k} V_i = V$, such that all $G_i \in \mathcal{G}$.

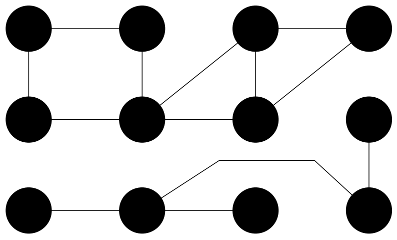


Figure 2.1: An example graph with 12 vertices

Example 2.13 Let us view at a particular example to explain most of the notions introduces so far. Here and in the following, we consider the graph with 12 vertices as depicted in Fig. 2.1. This graph has edges that are (as usually) graphically represented as line segments or (in one case) more generally as a polygon. The vertices of the graph are represented as circular blobs.

The following table gives an adjacency matrix of this graph:

[6]For a nearly complete overview on various graph classes, we refer to [Chapter 11, Site 8].

$$\begin{array}{cccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}$$

This table should be read as follows: the vertices (used as indices of that binary matrix) are read row by row from the picture (observe that in the picture there are three rows of vertices, each one containing four vertices), and in each row, from left to right. Therefore, the fifth column of the matrix contains the adjacency of the vertex depicted in the leftmost place of the second row.

We leave it as a small exercise to the reader to write up the incidence matrix of this graph.

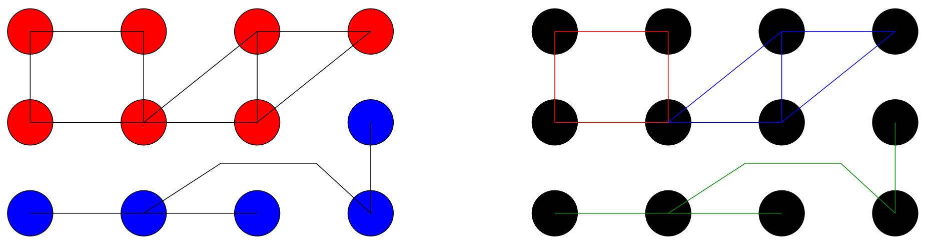
Let us look more closely at the graph to explain some more of the notions introduced so far. The graph has two connected components, depicted red and blue in Fig. 2.2(a). The blue component is a tree. Therefore, the graph induced by the green edges in Fig. 2.2(b) is also induced by the vertices of that component. Moreover, that component is bipartite.

Since the red component contains a triangle, it is not bipartite.

Similarly, also the graph induced by the red edges in Fig. 2.2(b) is vertex-induced. More specifically, that subgraph is a cycle of length four. However, the cycle of length for as given by the outermost four blue edges in Fig. 2.2(b) is not vertex-induced, since in the corresponding vertex-induced subgraph, a fifth edge is contained (also colored blue).

One can also define simple operations on graphs.

- If $G = (V, E)$ is a graph, its *graph complement* is the graph $G^c = (V, E^c)$, where $xy \in E^c$ iff $xy \notin E$.
- If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, their *graph union* is the graph $(V_1 \cup V_2, E_1 \cup E_2)$, assuming (in general) $V_1 \cap V_2 = \emptyset$.


(a) Two components: red and blue.

(b) Induced graphs.

Figure 2.2: Basic graph notions

- If $G = (V, E)$ is a graph and $x, y \in V$, then
$$G[x = y] = ((V \cup \{[x, y]\}) \setminus \{x, y\}, E')$$
denotes the *vertex merge* between x and y, where
$$E' = \{\{u, v\} \mid u, v \in V \setminus \{x, y\}\} \cup \{\{u, [x, y]\} \mid \{\{u, x\}, \{u, y\}\} \cap E \neq \emptyset\}.$$
(If $e = \{x, y\} \in E$, this operation is also known as the *contraction* of e.)

For example, a graph is a forest iff it can be expressed as the union of trees (hence its name).

We have already encountered some important graph classes like paths, cycles and trees. Of particular importance are also graphs that can be nicely drawn; planar graphs (as defined below) furthermore enjoy the fact that they are often encountered in practical circumstances, when graphs are to model phenomena "down on earth."

A graph $G = (V, E)$ is called a *planar graph* if it can be drawn in the plane without crossings between edges. To be more specific, this means that there exists an *embedding* of G into the Euclidean plane $\mathbb{R}^2$, which is a mapping that associates to every vertex v a unique point $\phi(v) \in \mathbb{R}^2$ and to every edge $\{u, v\}$ it associates a simple path (Jordan curve) $\phi((u, v))$ that connects $\phi(u)$ and $\phi(v)$, such that two paths have more than one common points only if their endpoints are identical. Recall that a simple path refers to a continuous curve in the plane that has no loops, i.e., every point in the plane is visited at most once, in accordance with the purely graph-theoretic notion of a simple path.

If a planar graph is given together with its embedding, we also speak of a *plane graph*.

Observe that an embedded graph partitions the surface on which it is embedded into open regions that are called *face*s.

2.4 Some favorite graph problems

Many problems considered in this Habilitationsschrift can be seen as covering or domination problems in (hyper-)graphs. To give the reader an idea, we list some of them here.

Problem name: d-HITTING SET (d-HS)
Given: A hypergraph $G = (V, E)$ with *hyperedge size* bounded by d
Parameter: a positive integer k
Output: Is there a *hitting set* $C \subseteq V$ with $|C| \leq k$?

Observe that if $d = 2$, HITTING SET actually becomes a problem on graphs, likewise known as VERTEX COVER, already considered above. Besides of being one of the first combinatorial problems whose $\mathcal{NP}$-hardness has been shown, VERTEX COVER is one of the problems that show up in many different places. For example, in [251, Chapter V] it is shown that the *minimum gene conflict resolution problem* from computational biology can be solved with the aid of VERTEX COVER.

Hence, this type of problems is also known as a *covering problem.* A good overview on algorithmic aspects of these types of problems (although a bit outdated by now) is [320].

Closely related to VERTEX COVER from a classical perspective are the following two problems: INDEPENDENT SET and CLIQUE, where the former is defined as follows:

Problem name: INDEPENDENT SET (IS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Namely, the complement of a minimal vertex cover is a maximal independent set; a maximal independent set in a graph is a maximal clique in the complement graph.

These problems are also known under different names. For example, an independent set is sometimes called: stable set, vertex packing, or anticlique.

If d is not set apart, but taken as part of the input, we arrive at:

Problem name: HITTING SET (HS)
Given: A hypergraph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *hitting set* $C \subseteq V$ with $|C| \leq k$?

Interestingly, HITTING SET can be also equivalently represented as a dominating set problem on graphs:

Problem name: RED-BLUE DOMINATING SET (RBDS)
Given: A graph $G = (V, E)$ with V partitioned as $V_{\text{red}} \cup V_{\text{blue}}$
Parameter: a positive integer k
Output: Is there a *red-blue dominating set* $D \subseteq V_{\text{red}}$ with $|D| \leq k$, i.e., $V_{\text{blue}} \subseteq N(D)$?

An instance $G = (V, E)$ of HITTING SET would be translated into an instance $G' = (V', E')$ of RED-BLUE DOMINATING SET as follows.

- The vertices of G would be interpreted as the red vertices of G'.
- The edges of G would be interpreted as the blue vertices of G'.
- An edge in G' between $x \in V$ and $e \in E$ means that $x \in e$ (in G).

More specifically, the bound d on the size of the hyperedges would have been reflected by a bound d on the degree of vertices in V_{blue}; let us call this restricted version of RED-BLUE DOMINATING SET d-RBDS.

The translation also works backwards, if we note that edges in a graph being an instance of RED-BLUE DOMINATING SET that connect vertices of the same color are superfluous and can be deleted. Namely, an edge connecting two red vertices is useless, since red vertices only dominate blue vertices and not red vertices; likewise, an edge between two blue vertices is useless, since blue vertices can be only dominated by red vertices. This observation also gives an example of reduction rules for this problem:

Reduction rule 3 *If e is an edge in a* RBDS *instance that connects vertices of the same color, then delete e.*

The exhaustive application of this reduction rule transfers each instance of RED-BLUE DOMINATING SET into a bipartite graph, where the bipartization of the vertex set is just given by the red and blue vertices.

Let us explicitly state this connection in the following lemma:

Lemma 2.14 RED-BLUE DOMINATING SET *and* RED-BLUE DOMINATING SET, RESTRICTED TO BIPARTITE GRAPHS *are parameterized interreducible.*

Then, it is easy to read the translation we gave above (working from HS into RBDS) in a backward manner.

Observe that this also gives an example of two (easy) parameterized reductions between these two problems. More specifically, we can state:

Lemma 2.15 HITTING SET *and* RED-BLUE DOMINATING SET *are parameterized interreducible. Moreover, assuming a bound d on the degree of the blue vertices and likewise a bound d on the size of the hyperedges, d-*HS *and d-*RBDS *are again parameterized interreducible.*

Essentially, the transformation between RBDS and HS can be also expressed as follows: if G is a hypergraph (as an instance for HS), then $B(A(G))$ is the corresponding RBDS instance, and if G is a bipartite graph (as a reduced instance for RBDS with respect to the rule 3), then $G(I(G))$ is the corresponding HS instance.

In fact, there is a third formulation of HITTING SET that can be found in the literature:

Problem name: SET COVER (SC)
Given: A groundset X, a collection T of subsets of X
Parameter: a positive integer k
Output: Is there a *set cover* $C \subseteq T$ with $|C| \leq k$, i.e., every element in X belongs to at least one member of C?

The translation works here as follows:

- The *groundset* X of the SC instance corresponds to the set of hyperedges in a HS instance.

- The collection of subsets T corresponds to the set of vertices in a HS instance, where for a subset $t \in T$, $x \in t$ means that the vertex t belongs to the hyperedge x.

From a practical point of view, HS and SC only formalize two kind of dual representations of hypergraphs:

- HS corresponds to represent hypergraphs by hyperedge sets, each of them being a list of incident vertices;

- SC means to represent hypergraphs by vertex sets, each of them being a list of incident hyperedges.

For the (in the sense explained above) equivalent problems HITTING SET and SET COVER, there exists a vast amount of literature. We only mention the (already a bit out-dated) annotated bibliography [73] for further reference.

The basic *domination problem* is not RBDS but the following one:

Problem name: DOMINATING SET (DS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *dominating set* $D \subseteq V$ with $|D| \leq k$?

Important variants are INDEPENDENT DOMINATING SET and CONNECTED DOMINATING SET:

Problem name: INDEPENDENT DOMINATING SET (IDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an independent *dominating set* $D \subseteq V$ with $|D| \leq k$?

This problem can be equivalently expressed as a *bicriteria problem*—and the reader may wish to check this equivalence— (a proof can be found in [314, Sec. 13]), without explicit reference to the domination property:

Problem name: MINIMUM MAXIMAL INDEPENDENT SET (MMIS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a maximal independent set of cardinality $\leq k$?

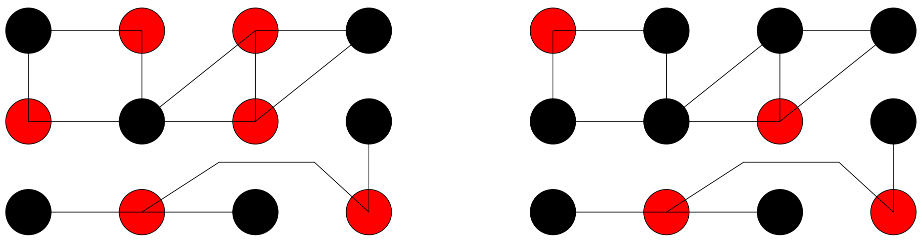
Also the other mentioned problem can be equivalently expressed without explicit reference to the domination property:

Problem name: CONNECTED DOMINATING SET (CDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *connected dominating set* $D \subseteq V$ with $|D| \leq k$, i.e., D is both a connected set and a dominating set?

Problem name: MINIMUM INNER NODE SPANNING TREE (MinINST)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *spanning tree* of G with at most k inner nodes?

The importance of domination problems is underlined by the fact that already around 1988 there have been about 300 papers dedicated towards this topic, see [230].[7]

Let us explain our favorite graph problems by continuing our example from Ex. 2.13.


(a) A minimum vertex cover.

(b) A minimum dominating set.

Figure 2.3: Basic graph problems

Example 2.16 What is a minimum vertex cover or a minimum dominating set of the graph depicted in Fig. 2.1?

Possible solutions are listed in Fig. 2.3. Why are these solutions minimum solutions?

Firstly, observe that we can deal with the two graph components independently. Then, it is not hard to see that the solutions proposed for the "lower" tree component are optimal. In fact, in the case of VERTEX COVER, the reduction rules as given in the next section allow to optimally solve that component, as the reader may verify.

Let us turn towards the "upper" component. It is now important to focus on the cycles in that component, since (as can be easily verified) $n/2$ vertices must go into any vertex cover of a cycle of length n if n is even, and $(n+1)/2$ vertices are contained in any vertex cover of a cycle of length n if n is odd. This implies that any cover for the graph induced by the blue edges in Fig. 2.2(b) has at least two vertices besides possibly the leftmost of those vertices. Hence, the chosen cover is optimum.

A similar reasoning is also possible in the case of DOMINATING SET. More specifically, for cycles basically every third vertex is needed to dominate it. Details are left to the reader.

[7]or, as one referee once commented to one of our papers: "Garey and Johnson [199] is not a democracy:" not all problems listed there are of equal importance

Moreover, observe that the dominating set in Fig. 2.3(b) is neither connected nor independent. The reader may wish to prove that there is indeed an independent dominating set of cardinality four of that graph, but no connected dominating set of that cardinality.

Let us mention here that covering and domination problems often appear in disguise, for example, in various puzzle questions.

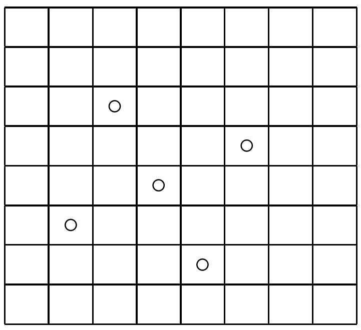
Example 2.17 One instance is the so-called *Five Queens Problem* on the chessboard:[8] it is required to place five queens on the board in such positions that they dominate each square. This task corresponds to DOMINATING SET as follows: the squares are the vertices of a graph; there is an edge between two such vertices x, y iff a queen placed on one square that corresponds to x can directly move to y (assuming the board were empty otherwise). In other words, the edge relation models the way a queen may move on a chess board.

According to [97], the general version played on a (general) $n \times n$ board is also known as the *Queen Domination Problem.* The task is to find the minimum number of queens that can be placed on a general chessboard so that each square contains a queen or is attacked by one. Recent bounds on the corresponding domination number can be found in [62, 63, 100, 315]. It appears to be that the queen domination number is "approximately" $n/2$, but only few exact values have been found up to today. By way of contrast, observe that on the $n \times n$ square *beehive*, the queen domination number has been established to be $\lfloor (2n+1)/3 \rfloor$, see [366].

As special classes of graphs (like planar graphs) will become important in later chapters of this Habilitationsschrift, let us mention that, although the structure we started with, namely the chessboard, is of course a planar structure, the corresponding graph model we used is not a planar graph: in particular, all vertices that correspond to one line on the chessboard form a clique, i.e., the family of chessboard queen graphs contains arbitrary cliques, quite impossible for planar graphs by Kuratowski's theorem.

Following Ore [314, Sec. 13], one solution to the Five Queens Problem is as follows:

[8]The terminology in this area is a bit awkward: as can be seen, the Five Queens Problem is not a special case of the n-Queens problem introduced below. We will rather distinguish these problems by referring to them as the Queen domination problem and the Queen independence problem later on.



It is clear that similar domination-type problems may be created from other chess pieces as rooks, bishops, or knights (or even artificial chess pieces), see [202] as an example.

Also the variant that only certain (predetermined) squares need to be dominated has been considered. For example, the $n \times n$ *Rook Domination Problem* is trivially solvable by choosing one main diagonal and putting n rooks on this diagonal. By pidgeon-hole, any trial to dominate all squares by fewer than n rooks must fail. However, if only a certain number of predetermined squares need to be dominated by placing a minimum number of rooks on one of those predetermined squares, we arrive at a problem also known as MATRIX DOMINATION SET; this is treated in more detail in Sec. 8.2.

Further variants of the Queen domination problem are discussed in [100].

Example 2.18 A related problem on the chessboard is the following one, originally introduced in 1850 by Carl Friedrich Gauß:[9]

The *n-Queens Problem* may be stated as follows: find a placement of n queens on an $n \times n$ chessboard, such that no queen can be taken by any other, see [333]. Again, this problem can be also considered for other chess pieces and modified boards, see (e.g.) [52] as a recent reference.

We can use the same translation as sketched in Ex. 2.17 to translate the chessboard into a (non-planar) graph that reflects reachability by movements of a queen.

If (for the moment) we "forget" about the well-known property that in fact a solution exists to the n-Queens Problem for each n, then the n-Queens Problem corresponds to the task of solving MAXIMUM INDEPENDENT SET on the chess-board graph (with n^2 vertices).

In fact, this approach was taken in [270], where an integer linear programming formulation was presented. The example for $n = 10$ of Fig. 2.4 was taken from that paper.

[9]This solution is contained in a letter to H. C. Schuhmacher dating from Sept. 12th, reprinted on pp. 19–21 in [204]; this was a sort of reply to a problem that was posed two

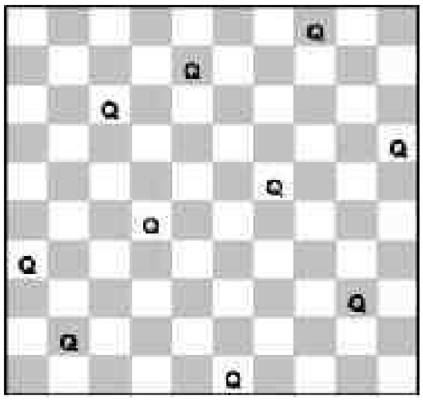


Figure 2.4: A sample solution of the n-Queens Problem for $n = 10$.

More information on Mathematical Games and their solutions can be found in [Chapter 11, Site 9]. Regarding mathematical chess, we recommend [Chapter 11, Site 4].

2.5 The power of data reduction

As we will see in the following chapters, data reduction is (in practice) the basic trick to efficiently solve computationally hard problems. In the following, we show this potential of data reduction rules with the example of HITTING SET.

Data reduction rules for HITTING SET

Reduction rule 4 (hyper)edge domination: *A hyperedge e is* dominated *by another hyperedge f if $f \subset e$. In such a situation, delete e.*

Reduction rule 5 tiny edges: *Delete all hyperedges of degree one, place the corresponding vertices into the hitting set.*

Reduction rule 6 vertex domination: *A vertex x is* dominated *by a vertex y if, whenever x belongs to some hyperedge e, then y belongs to e, as well. In such a situation, delete x.*

Lemma 2.19 *The three reduction rules are sound.*

years earlier in the magazine of the "Berliner Schachgesellschaft."

Although the reduction rules were elsewhere mentioned, we are not aware of any formal correctness proof. Since the soundness of the reductions is essential to the correctness of the overall algorithm, we provide the corresponding soundness proofs right here. This also gives examples of how to actually prove the correctness (usually called *soundness*) of reduction rules.

Proof. We have to show that, whenever S is a solution to an instance (G, k), then there is a solution S' to the instance (G', k'), where (G', k') is obtained from (G, k) by applying any of the reduction rules. We must also show the converse direction.

1. (hyper)edge domination: Let e and f be hyperedges in G such that $f \subset e$. If S is a solution to (G, k), then trivially S is also a solution to the instance (G', k) obtained from (G, k) by applying the edge domination rule to the situation $f \subset e$. Conversely, if S' is a solution to (G', k), then in particular $S' \cap f \neq \emptyset$, which means that $S' \cap e \neq \emptyset$, so that S' is a solution to (G, k), as well.

2. tiny edges: Hyperedges of degree one can only be covered by the vertices they contain.

3. vertex domination: Let x and y be vertices in G such that x is dominated by y. If S is a solution to (G, k) which does not contain x, then S is also a solution to the instance (G', k) obtained from (G, k) by applying the vertex domination rule triggered by the domination of x by y. If S is a solution to (G, k) which contains x, then because x is dominated by y, $(S \setminus \{x\}) \cup \{y\}$ is also a solution to (G, k) and, by the preceding sentence, this is a solution to the reduced instance (G', k), as well. Conversely, if S' is a solution to (G', k), S' is a solution to (G, k), since no edges are deleted when forming G'. ∎

The rules themselves are not new: the last two are also used by Niedermeier and Rossmanith in [308] and all of them are used by Wahlstöm in [373]. Actually, the rules seem to be "around" since many years. The oldest reference (which was found by Regina Barretta, Newcastle) is to our knowledge [201, Chapter 8]. They are also known for related problems as, e.g., the PATH COVER PROBLEM, see [375].

Reduction rules are often valuable to produce problem instances that have specific properties. More precisely, if R is a set of reduction rules and I is a problem instance to which none of the rule in R applies, then I is also called an *R-reduced instance* or an *R-irreducible instance.* If R is clear from the context, then R will be also suppressed. Hence, we can formulate:

Lemma 2.20 *In a reduced* HITTING SET *instance with at least two vertices, no vertex has degree less than two.*

Proof. In the reduced instance, there are no vertices of degree zero, since this would trigger the vertex domination rule, because there are at least two vertices. The vertex domination rule would also get rid of vertices of degree one that are connected to other vertices. Vertices of degree one that are not connected to other vertices are coped with by the tiny edge rule. ∎

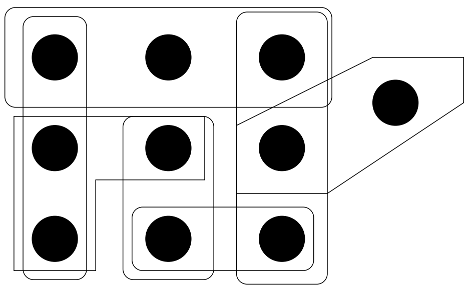


Figure 2.5: A sample HITTING SET instance

Example 2.21 To see how these reduction rules work, consider the example depicted in Fig. 2.5. The different black circles indicate different vertices, and the regions that include certain vertices denote the corresponding hyperedges. Observe that the size of the hyperedges is bounded by three in this example, but this does not matter regarding the applicability of the reduction rules.

In Fig. 2.6(a), two locations are indicated (by coloring them) to which the vertex domination rule is applicable in the given HS instance. More precisely, in each of the two hyperedges whose boundaries are colored blue, the vertex colored red is dominated by the vertex colored green. The rule tells us to remove those dominated vertices, which leads to Fig. 2.6(b). What happens next is indicated by coloring one vertex and one hyperedge (boundary) red and one hyperedge blue; actually, there are two ways of explaining the same outcome:

1. The red-colored hyperedge dominates the "surrounding hyperedge" colored blue. Hence, the edge domination rule is triggered. Then, the tiny edge rule puts the red vertex into the hitting set, and the red hyperedge is covered and hence removed.

2. The tiny edge rule puts the red vertex into the hitting set, and both the red hyperedge and the blue hyperedge are covered and hence removed.

Anyways, the picture from Fig. 2.6(c) is produced. Again, the vertices colored red in that picture are dominated by green vertices.

The next Figure 2.6(d) is obtained by applying the vertex domination rule twice (as described) to Fig. 2.6(c). In Fig. 2.6(d), the tiny edge rules (and edge domination rules) are applicable to the two hyperedges colored red, incidentally also destroying the blue hyperedges (similar to the situation in Fig. 2.6(b)).

Applications of the mentioned rules lead to the situation depicted in Fig. 2.6(e). Finally, the vertex domination rule applies twice in that situation (as indicated by the by now familiar color coding).

Hence, we can conclude that the reduction rules 4, 5 and 6 allow to completely resolve the original HS instance. A solution is indicated by coloring the hitting set vertices blue in Fig. 2.6(f).

Let us add some more remarks in this place:

Remark 2.22 *The three rules we listed for* HITTING SET *have the specific property that they are formulated without explicit reference to the (change of the) parameter by applying the reduction rules. This indicates that the rules are also true for the minimization variant* MINIMUM HITTING SET, *as well as for different parameterizations.*

However, the proof of the soundness of the rules given in Lemma 2.19 is based on the standard parameter.[10] *Hence, the soundness proof is also valid for* MINIMUM HITTING SET, *but not necessarily for other parameterizations. Nonetheless, we leave it as an—easy—exercise to the reader to prove the soundness of the rules with respect to the following (natural) parameters, that will be also considered in this Habilitationsschrift:*

- *parameterization by the number of vertices, and*
- *parameterization by the number of hyperedges.*

Remark 2.23 *Reduction rules have always been used by practitioners to solve hard problems. Parameterized algorithmics makes use of this idea in at least three different places:*

- *for devising kernelization algorithms,*
- *for developing search tree algorithms, and*
- *for the final tuning of search tree algorithms, trading off space for time.*

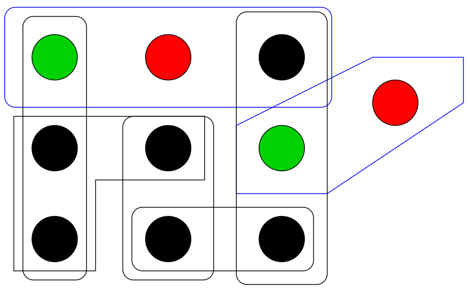
[10]These notions are formally explained in the next chapter.

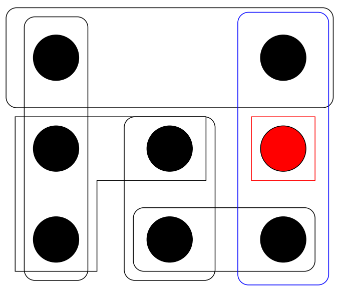
The reduction rules we discussed for HITTING SET *has not (yet) turned out to be fruitful to develop kernelization algorithms for* HITTING SET*. However, as we will explain in Chapter 5, they are the key for the development of the fastest known parameterized algorithms for* 3-HS *(and variants thereof).*

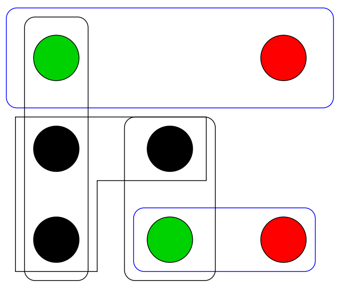
As to the third point, note that Lemma 2.20 is the key observation to a final tuning of parameterized algorithms for VERTEX COVER *that use both exponential time and exponential space, see [77].*[11]

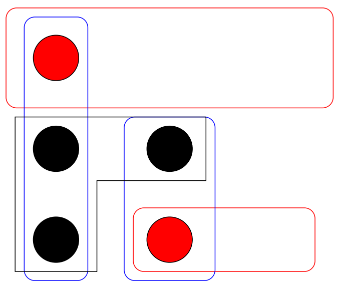
Finally, a reader might ask for an example on which the given reduction rules don't work so smoothly, i.e., an example of an irreducible instance. In fact, this can be obtained by a slight modification of Fig. 2.5, as depicted in Fig. 2.7; the only edge that is added is colored blue. By observing that the vertex domination rule is the only rule applicable to Fig. 2.5, and the only places to which it could be applied are the ones indicated in Fig. 2.6(a), it can be seen that Fig. 2.7 is indeed a reduced instance of HITTING SET.

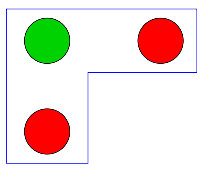
[11]This trick is not new and is also exhibited in [77, 306, 342, 373] to mention only a few references. However, we think that still exponential time is "more affordable" than exponential space, not only from the presumed difference between time and space classes with the same resource bounds, but also from purely practical considerations, since abundant use of storage will first lead to paging and finally kill any application. Therefore, we did not expand on this "trick" in this Habilitationsschrift.


(a) First applying the vertex domination rule to Fig. 2.5, ...


(b) ...then applying the tiny edges & edge domination rules, ...


(c) ...then again applying the vertex domination rule, ...


(d) ...and the tiny edges & edge domination rules.


(e) The vertex domination rule resolves the problem.

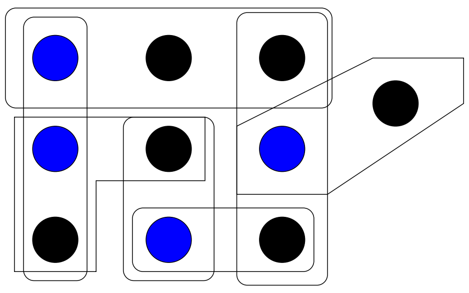

(f) A mimimum hitting set produced by reduction rules.

Figure 2.6: How the reduction rules completely resolve the HITTING SET instance given in Fig. 2.5

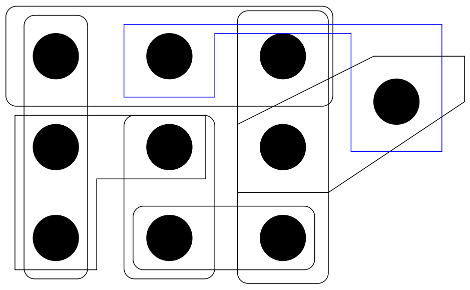


Figure 2.7: A sample HITTING SET instance that is reduced

Chapter 3

Parameterizations

In this chapter, we are going to develop ideas what can serve as a parameter for the parameterized approach. This question looks funny, at first glance, but is an issue, indeed. Problems may be hard from a parameterized viewpoint with a certain parameterization, but tractable with another parameterization. In fact, this statement is true in a very general sense: since in principal anything could be the parameter, especially the overall size of the problem instance could be taken as the parameter of that instance. Under such a parameterization, any computable problem becomes parameterized tractable. Of course, this is rather a gambler's trick, but it shows the dramatic effect that a change of the parameterization can have. However, the basic techniques presented in this Habilitationsschrift also apply to this rather non-parameterized setting; more about this in Chapter 10.

This chapter is structured as follows: In Sec. 3.1, we are introducing and discussing a simple conceptual scheme to classify different choices of parameters. This should also make clear that a problem might have different reasonable parameterizations, as exemplified by exploring the well-known game Rush Hour. However, it has become common in the parameterized complexity community to view the entity to be optimized as a standard parameter for problems that are usually formulated as optimization problems. This issue is critically discussed in the following section. Moreover, linear arrangement and facility location problems are introduced and investigated to illustrate the ideas. In Sec. 3.3, alternative parameterization strategies are discussed and illustrated with various examples. Up to this point, a problem used to have only one distinguished parameter. This need not be the case, in particular in several application areas, where several entities might be identified that turn out to be small and hence useful as a parameter. This issue is detailed in Sec. 3.4. Finally, Sec. 3.5 discusses the issue of parameter duality, which can be also viewed as discussing two possible parameterizations of a classical optimization problem from two seemingly equivalent angles: maximizing the profit is kind of equivalent to minimizing the costs, but from the point of view of parameterized algorithmics, it might make a huge difference if we choose a bound either on the profit or on the cost as parameter. In this context, it turns out to be important to formalize the size of an instance formally, an issue that will be also important in other chapters.

3.1 Internal and external parameters

Let us begin our considerations on parameterizations with a simple classification of parameters.

3.1.1 A classification scheme

Parameters classify as *explicit internal parameter*s if they explicitly show up in the specification list of an instance of the problem under consideration. For example, in VERTEX COVER, entities that qualify as explicit internal parameters are:

- The bound k on the cover size. This is also the standard parameter, see Sec. 3.2.
- The number n of vertices of the graph.
- The number m of edges of the graph.

Since n and m are naturally polynomially related to the overall size of the input, they are usually not regarded as *proper parameterization*s, but rather considered to be the corresponding *non-parameterized problem*, see Chapter 10. By way of contrast, observe that a valid vertex cover might be arbitrarily small in comparison to the overall number of vertices: consider a *star graph* as an example.

Another form of *internal parameter*s are *implicit internal parameter*s. Here, we only require that the parameter is bounded by a polynomial in the overall size of the instance (counted without the parameter). A typical example of an internal parameter that is implicit is the treewidth of a graph, see Chapter 7 for the definitions; of course, every graph $G = (V, E)$ can be represented by a tree decomposition with $|V|$ vertices in just one bag.

Finally, there are parameters that are not bounded by a polynomial in the overall size of the instance (counted without the parameter). For that case, we are only aware of natural examples that qualify themselves as counting or enumeration problems, if we take the number of items to be counted or enumerated as a parameter, as a sort of parameterization that measures the output-sensitivity. This specific type of problems are usually differently parameterized. Therefore, we omit a further classification here and refer to Chapter 8.

Finally, there are *external parameter*s that are not bounded at all by the input specification, measured without taking the parameter into account. Typically, external parameters are stemming from dynamical aspects of a problem. For example, when considering Turing machines, the number of possible moves till the machine halts or the number of visited squares are typical external parameters.

Unfortunately, usually external parameterizations lead to problems that are not parameterized tractable. This is particularly the case of (most) Turing machine computation variants (along the lines previously sketched; more can be found in Chapter 9). However, already in the next subsection, we will encounter an $\mathcal{FPT}$ result with an external parameterization.

3.1.2 Different classifications in games

Let us elucidate the choice of different parameters with the discussion of an (abstract) game, namely RUSH HOUR, see [178] for more details.

We consider the parameterized complexity of a generalized version of the game *Rush Hour*[1], which is a puzzle requiring the player to find a sequence of moves by vehicles to enable a special target vehicle to escape from a grid-shaped game board that may contain obstacles. Although the problem is $\mathcal{PSPACE}$-complete [184], we demonstrate algorithms that work in polynomial time when either the total number of vehicles or the total number of moves is bounded by a constant.

The goal of the Rush Hour puzzle is to remove a target vehicle (that is colored red in the original version of Rush Hour) from a grid by moving it and other vehicles until eventually it reaches an exit on the perimeter of the grid. So, in Fig. 3.1, the task is to move out the red car to leave the parking at the exit that is located at the right side of the parking site. Each vehicle can move either horizontally or vertically, but cannot change its initial orientation (not even in successive moves; hence, it cannot make turns), and a vehicle can be moved from one position to another only when all intervening grid positions are empty. The grid may also contain obstacles that do not move; we use the generic term "car" to denote a vehicle or an obstacle. Fig. 3.1 contains no obstacles. The original puzzle consists of a 6×6 grid with a single fixed

[1]The name "Rush Hour" is a trademark of Binary Arts, Inc.; more info on the current status of Rush Hour can be found on the Web, see [Chapter 11, Site 19].

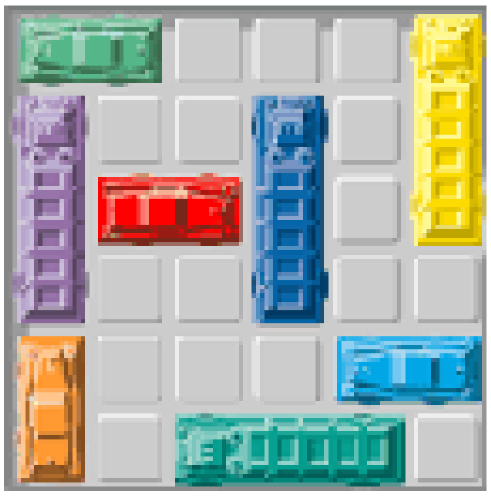


Figure 3.1: A sample instance of the classical rush hour puzzle.

exit and vehicles of width one and length two or three; a sample instance is depicted in Fig. 3.1.

The Rush Hour puzzle is one of many games and puzzles determined to be computationally difficult to solve [111]; such problems include the (n^2-1)-puzzle [327] and Atomix [235, 244], related to a variant of Rush Hour in which all cars are squares of shape 1×1. The original Rush Hour problem was generalized by Flake and Baum to allow arbitrary grid sizes and placements of the exit and shown to be $\mathcal{PSPACE}$-complete [184].

We extend the generalization formulated by Flake and Baum [184]. In our generalization, the game board is an infinite plane, cars are arbitrary axes-parallel rectangles, and each vehicle must reach one of a set of goal positions in the plane.

The $\mathcal{PSPACE}$-completeness result of Flake and Baum indicates that there is not much hope of finding a solution to a given Rush Hour instance in polynomial time. However, we show that when the number of cars or the number of moves allowed is small, the problem can be solved in polynomial time. Observe that the number of cars is an (explicit) internal parameter, while the number of moves is an external parameter.

In still greater generality, an instance of RUSH HOUR can be formalized as a tuple (C, S, p^0, d, Z), where C is a finite set (of *cars*) and the remaining components are functions mapping C to $2^{\mathbb{R}^2}$, $\mathbb{R}^2$, $\mathbb{R}^2$ and $2^{\mathbb{R}}$, respectively. For each $c \in C$, $S(c)$ is the *car shape* of c, $p^0(c)$ is its *initial position*, $d(c)$

is its *directional vector*, and $Z(c)$ is the set of its *RH goals*.[2] A car c is an obstacle iff $d(c) = (0, 0)$. A car c initially covers the points $p^0(c) + S(c)$ in the plane, and it can be moved to positions of the form $p^0(c) + ud(c)$, for $u \in \mathbb{R}$, where it will cover the points $(p^0(c)+ud(c))+S(c)$. Correspondingly, the *RH position* of car c (at a certain moment in time) can be described by a number $u(c) \in \mathbb{R}$, and the set of goals available to a car can be described by a set of *goal intervals*.

The object of the game is to move each car to one of its goal positions. By defining trivially achievable goal intervals, the original version of the game can be seen to be a special case of our definition.

A *RH configuration* is a function $u : C \to \mathbb{R}$, and it is a *legal configuration* if

$$(p^0(c) + u(c)d(c) + S(c)) \cap (p^0(c') + u(c')d(c') + S(c')) = \emptyset$$

for all $c, c' \in C$ with $c \neq c'$. A *RH move* is an operation that adds to (or subtracts from) the position of a car a multiple of its directional vector; it is a *legal move* if the initial and final configurations as well as all intermediate configurations are legal, that is, no other car blocks the passage. A *RH solution* to an instance is a sequence of legal configurations such that the first consists of the cars in their initial positions, the last consists of the cars in goal positions, and each pair of successive configurations is connected via a legal move.

Most of our work focuses on a special case of the general problem in which all cars are **a**xes-**p**arallel **r**ectangles and the only directional vectors allowed are $(1, 0)$, $(0, 1)$ and $(0, 0)$. An *APR instance* satisfies the additional constraints that for all $c \in C$, $d(c) \in \{(1, 0), (0, 1), (0, 0)\}$ and $S(c)$ is an open rectangle contained in the first quadrant of the plane and with the origin as a corner. In this case, a car c is called a *horizontal car* if $d(c) \in \{(1, 0), (0, 0)\}$ and a *vertical car* if $d(c) \in \{(0, 1), (0, 0)\}$. Flake and Baum's version is a special case of APR RUSH HOUR.

Problem name: RUSH HOUR, PARAMETERIZED BY CARS (RH (CARS))
Given: A RH tuple (C, S, p^0, d, Z) of an APR instance
Parameter: a positive integer k, upperbounding $|C|$
Output: Is there a sequence of legal moves that solves the given RH instance?

By observing that, for each car, there is only a certain number of so-called *discretized positions* in which it may be, i.e., other positions are in

[2]Here and in the following, we add "RH" to some notions that have a special meaning for RUSH HOUR.

a way equivalent to one of the discretized positions, and by upperbounding the number of discretized positions by a function in k, in [178] it is shown that RUSH HOUR, PARAMETERIZED BY CARS is parameterized tractable, since then there are at most $(2k^3)^k$ *discretized configurations.* To limit the construction of the space of discretized configurations, we assume that all cars are initially in an $n \times n$ square. Thus, we can state:

Theorem 3.1 RUSH HOUR, PARAMETERIZED BY CARS *can be solved in time $\mathcal{O}((2k^3)^{2k}p(n))$, where p is a polynomial.*

We omit to list the corresponding algorithm here, but only emphasize once more its two basic steps:

- Generate (at least implicitly) the search space of discretized configurations.
- Exhaustively search through that space by first testing if there is a one-step transition between each pair of states and then looking for a path from the initial configuration to some goal configuration.

Observe that the parameterization of RUSH HOUR in terms of the number of cars is an obvious explicit internal parameterization. However, the following alternative parameterization is obviously external, and due to the classical complexity of the problem [235], it was even a bit surprising to see here a parameterized tractability result according to our comments in the previous subsection.

Problem name: RUSH HOUR, PARAMETERIZED BY MOVES (RH (MOVES))
Given: A RH tuple (C, S, p^0, d, Z) of an APR instance
Parameter: a positive integer m
Output: Is there a sequence of at most m legal moves that solves the given RH instance?

To handle the case where the number of moves m is bounded, we compute a set of cars with the property that if all other cars are removed, we are certain that for every $t \leq m$ this will not change an instance without a t-step solution into one that has a t-step solution. In other words, the computed set should contain all cars that might be relevant. It may contain additional spurious cars that, in actual fact, do not make any difference, but we bound the number of relevant cars by 3^m. Then we apply an algorithm similar to the one previoustly sketched which leads to a running time of the form $2^{\mathcal{O}(m^2 \cdot 3^m)}p(n)$.

The main point that might be applicable to other situations, as well, is that we can make use of an $\mathcal{FPT}$ algorithm for one parameterization to devise an $\mathcal{FPT}$ for a different, seemingly more complicated parameterization.

3.2 Standard parameters

3.2.1 Minimization problems

For *minimization problems*, it is kind of standard approach to consider the entity to be minimized as the parameter. This is in accordance with the observation that, as long as minimization problems aim at minimizing some sort of cost function, it is desirable to have these costs as small as possible. Even "worse:" if a minimal solution incurs tremendous costs, then it would not be worthwhile considering them at all in practical terms. Let us point the reader here especially to the examples drawn from VLSI fabrication (discussed in the next section) and from graph drawing, see Sec. 6.4.

Let us be more specific with a certain number of examples:

- A good selection of problems where parameterized algorihms have been developed for are *cover problems*. The most notable examples are VERTEX COVER and, more generally, HITTING SET.

- A whole lot of minimization problems stem from (technical) *failure models*. Since typically failures can be repaired in multiple ways, the corresponding minimization task is to determine a minimum set of repair operations that can cope with all failures.

  Conversely, by assuming the principle of *Occam's razor*, a *theory of diagnosis* has been developed [331]. Here, the task is to analyze a given set of failure symptoms to explore the reasons of failure. This has been set up rather in the context of diagnosis in medicine than in the context of analysis of technical failures, but the ideas are of course transferrable. Occam's razor implies that the best diagnosis is showing the least complicated explanation for all observed symptoms. More mathematically speaking, but adhering to the medical context, this is about finding the smallest set of deseases that can explain all symptoms. Having found this set, the "repair" (e.g., the choice of adequate medication) is relatively easy.

  Both related setting often quite easily translate into cover problems. In fact, Reiter [331] based his theory of diagnosis on a clever search along so-called *hitting set trees*, using standard methods from Artificial Intelligence.

In fact, this motivates why cover problems play a central role in this Habilitationsschrift. More explanations on Reiter's approach can be found in Chapter 8.2.

- Many examples of optimization problems have a *cost function* that determines the value of a valid solution. Naturally, a solution incurring minimum cost is used. In the graph-theoretic terminology, such problems often correspond to weighted versions of graph minimization problems. Then, weights (costs) are associated to vertices, edges, etc.

As a more "natural" problem (providing an example for the last mentioned category of minimization problems based on cost functions), let us consider FACILITY LOCATION.

Problem name: FACILITY LOCATION (FL)
Given: A bipartite graph $B = (F \uplus C, E)$, consisting of a set F of potential *facility location*s, a set C of *customer*s, and an edge relation E, where $\{f, c\} \in E$ indicates that c can be served from the facility (at) f; and a weight functions $w_F : F \to \mathbb{N}$ and $w_E : E \to \mathbb{N}$ (both called w if no confusion may arise)
Parameter: $k \in \mathbb{N}$
Output: Is there a set $F' \subseteq F$ of facility locations and a set $E' \subseteq E$ of ways to serve customers such that (1) $E' \cap F = F'$, (2) $E' \cap C = C$, and (3) $\sum_{f \in F'} w_F(f) + \sum_{e \in E'} w_E(e) \leq k$?

Alternatively, and sometimes more convenient, this problem can be formulated in terms of a "matrix problem:"

Problem name: FACILITY LOCATION (MATRIX FORMULATION)
Given: A matrix $M \in \mathbb{N}^{(n+1)\times m}$, indexed as $M[0 \dots n][1 \dots m]$.
Parameter: $k \in \mathbb{N}$
Output: Is there a set $C \subseteq \{1, \dots, m\}$ of columns and a function $s : \{1, \dots, n\} \to C$ such that $\sum_{f \in C}(M[0, f] + \sum_{c:s(c)=f} M[c, f]) \leq k$?

In the matrix formulation, the columns play the role of the potential facility locations and the rows represent the customers to be served. Since "missing edges" in the bipartite graph formulation can be expressed as edges which have "infinite weight" (corresponding in turn to a weight larger than k in the decision problem formulation), we can assume that the graph is indeed a complete graph. Then, the matrix $M[1 \dots n][1 \dots m]$ contains the weights of the edges, while $M[0][1 \dots m]$ stores the weights associated to potential facility locations.

Whenever we return to this example in the following, we will use terminology from both formulations interchangeably, whatever seems to be more convenient. Notice that the matrix representation is related to the adjacency matrix of a bipartite graph.

Summararizing what we said above, when facing a minimization problem, the *standard parameterization (for minimization problems)* is that the entity to be minimized is the parameter.

3.2.2 Maximization problems

Also with *maximization problems*, it has turned out to be fruitful to consider the entity to be maximized as the parameter. We will also refer to this setting as the *standard parameterization (for maximization problems)*.

Both forms of standard parameterizations are called *parameterized versions* of an optimization problem in [84]. However, in the case of maximization problems, there are some philosophical arguments against using the standard parameterization as a "standard."

- A natural counterpart to covering problems are packing problems; here the aim is to fit in as many items as possible into a given boundary or framework. The probably best-known of such "packing problems" is MAXIMUM KNAPSACK, although this particular problem can be also viewed as a profit problem as discussed below. In graph-theoretical terms, MAXIMUM INDEPENDENT SET can be viewed as a packing problem, as well: the task is to place as many pebbles as possible onto vertices of a graph that serves as the specified framework. The restriction is that no two pebbles may be placed on neighboring vertices.

  As will be explicitly discussed below in the next section, the "duality" of covering and packing problems can be formally expressed in the framework of parameterized algorithms. Let us only state here that if we assume, as argued above, that the entity to be minimized (in the standard parameterization of a covering problem) is "small" (when compared to the overall size of the instance, measured without the parameter), then its dual packing problem cannot enjoy the same property (namely, that the standard parameter is small), unless the overall instance is small, anyhow.

  So, expecting a small standard parameter for packing problems seem to be hard to justify on these philosophical grounds.

  However, there might be situations where certain graph parameters that are to be maximized (to keep within this rather general model) are known to be small compared to the overall instance size. Then, it would make perfect sense to consider the entity to be maximized as a parameter.

- The natural maximization correspondence to minimize the cost is to maximize the profit of an enterprise. This leads to *profit problem*s. As well as the expectation can be justified that costs should be small, one wants profits to be large. Again, this counteracts the basic underlying assumption of parameterized algorithms that the parameter (values) are small. MAXIMUM KNAPSACK is probably the best known among such profit problems:

  Problem name: MAXIMUM KNAPSACK (KS)
  Given: n items $\{x_1, \dots, x_n\}$ with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k. All numbers are natural numbers encoded in binary.
  Parameter: k
  Output: Is there a subset of items which yield a profit larger than k and has an overall size of less than b?

  Profit variants of graph problems have been also discussed, see [354] for a variant of VERTEX COVER that is defined as follows:

  Problem name: PROFIT VERTEX COVER (PrVC)
  Given: A graph $G = (V, E)$
  Parameter: a positive integer p
  Output: Is there a *profit vertex cover* $C \subseteq V$ with $|E| - |E(G[V \setminus C])| - |C| \geq p$?

  Intuitively speaking, the *profit of a cover* is the gain of the cover, i.e., the size of the edge set the is covered that can be expressed as $|E| - |E(G[V \setminus C])|$, minus the cost of the cover, i.e., its size $|C|$. In [354], a $\mathcal{O}(p|V(G)| + 1.151^p)$-algorithm is reported.

As ways out of the dilemma with maximization problems (especially if it is expected that the values of the standard parameterization of the maximization problem are known to be large), we might think of the following:

- Consider the *parameterized dual* of the problem, which will always be a minimization problem in case we started with a maximization problem, and when we expect that the values of the standard parameterization of the maximization problem are "big," then the values of the standard parameterization of the dual minimization problem are "small." This technique is discussed in detail in Sec. 3.5.

- If the optimization problem can be expressed as an *integer linear program (ILP)*, then we can translate its so-called *relaxation* into an "equivalent" dual problem (where dual is meant in terms of linear programming; we won't give details here). Since the dual program of a maximization problem is a minimization problem, we might use this as an idea for the definition of parameters for the problem in question that are meant to be small.

  For example, an instance MAXIMUM KNAPSACK can be expressed as follows as ILP:

  $$\max \sum_{i=1}^{n} p_i x_i$$

  subject to

  $$\sum_{i=1}^{n} s_i x_i \leq b,$$

  where $x_i \in \{0,1\}$ is the integrality condition. When we ignore the integrality condition, we arrive at a linear program.

  The corresponding dual minimization problem could give us the idea of the following reparameterization of KS:

  Problem name: MAXIMUM KNAPSACK, MINIMUM WEIGHT (KSMW)
  Given: n items $\{x_1, \dots, x_n\}$ with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k. All numbers are natural numbers encoded in binary.
  Parameter: b
  Output: Is there a subset of items which yield a profit larger than k and has an overall size of less than b?

- Finally, if it is *known* that the value of any solution to a maximization problem must be above a certain threshold, then it is useful to consider *re-parameterization* according to the strategy of *parameterizing above guaranteed values*, as detailed in the next section.

Except from Sec. 4.3, we will not further the study of problems related to MAXIMUM KNAPSACK in the following. However, let us notify that all these problems are typically in $\mathcal{FPT}$. More precisely, Chen *et al.* [84, Theorem 1] have shown that so-called *scalable* optimization problems have a *fully polynomial-time approximation scheme (FPTAS)* iff they are in a certain subclass of $\mathcal{FPT}$ they called *efficient-$\mathcal{FPT}$*. Since MAXIMUM KNAPSACK and many similar problems are scalable and known to possess an FPTAS (see [30]), it is clear that all these optimization problems (with the standard parameterization) are in $\mathcal{FPT}$. In fact, this already follows from [68, Theorem 3.2]. More precisely, if the FPTAS allows to solve the optimization problem in time $p(|x|, 1/\epsilon)$ (where x is the input and ϵ is the approximation factor to be attained), then the standard parameter k yields a problem instance (x, k) that can be solved in time $\mathcal{O}(p(|x|, 2k))$.

3.2.3 Philosophical remarks, inspired by LINEAR ARRANGEMENT

It is finally worth mentioning that the standard parameterization is not always very meaningful even for minimization problems. For example, consider the following problem that is number GT 42 in [199]; also refer to [200]:

Problem name: LINEAR ARRANGEMENT (LA)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that

$$\sum_{\{u,v\}\in E} |\sigma(u) - \sigma(v)| \leq k \ ?$$

In actual fact, this problem is given in the reference as defined above, i.e., including the bound k, and not as a minimization problem (to which it is naturally related). LINEAR ARRANGEMENT is only one amongst a huge number of so-called arrangement or (more generally) layout problems. Some of them will be discussed in this Habilitationsschrift. A recent survey by J. Díaz, J. Petit and M. Serna has been published in [126]. Let us briefly review the literature on this problem first. The most comprehensive study on this problem appears to be the PhD thesis of S. B. Horton [239]. Surprisingly few graph classes are known on which LINEAR ARRANGEMENT can be efficiently solved:

- trees [5, 99, 352],

- a subclass of Halin graphs [150, 239],
- outerplanar graphs [194], ...

Of interest are of course also variations of this scheme; for example, the fitting of a graph into a two-dimenional grid was discussed in [239]. Also, the rather geometric problem of fitting a k-point polyline seems to be related, see [28], although in general these more geometric problems tend to be easier, see [268]. Some literature is also reviewed at [Chapter 11, Site 10].

To see the flavor of this problem, look at a simple example:

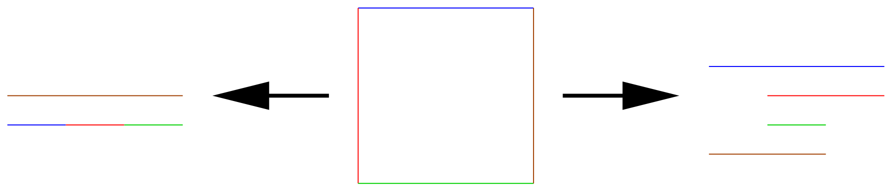


Figure 3.2: Two ways of mapping C_4

Example 3.2 As can be easily seen, the circle C_4 cannot be mapped onto a line in a way that preserves edge lengths, i.e., not every edge corresponds to an interval of length one. In Fig. 3.2, we used color coding to visualize two mappings. The one to the left is optimal: all but one edge correspond to an interval of unit length; this arrangement then "costs" 6 length units. The one to the right shows a different mapping that costs 8 length units (and is therefore worse than the other one).

Since the distance even between the σ-images of two vertices that are mapped on neighboring points is at least one, for connected graphs one can immediately derive for the *linear arrangement number*

$$\lambda(G) = \sum_{\{u,v\}\in E} |\sigma^*(u) - \sigma^*(v)|$$

for a minimum arrangement σ^*:[3]

Proposition 3.3 *If $G = (V, E)$ is a connected graph, then $\lambda(G) \geq |V| - 1$.*

Hence, for connected graphs, the following rule is sound:

[3]The result of Prop. 3.3 is also mentioned in [239, page 34]. Further bounds based on cuts and on eigenvalues of the Laplacian of a given graph do not seem to help for the kernelization we have in mind; they can be found in [239], as well; also compare [249, 241].

Reduction rule 7 *If (G, k) is an instance of* LINEAR ARRANGEMENT*and if $G = (V, E)$ is a connected graph, then return* NO *if $k < |V| - 1$.*

However, the following assertion is pretty clear:

Lemma 3.4 *Let $G = (V, E)$ be a graph. If C_1 and C_2 is a partition of V into two vertex sets such that, for all $e \in E$, either $e \subseteq C_1$ or $e \subseteq C_2$, then $\lambda(G) = \lambda(G[C_1]) + \lambda(G[C_2])$.*

Proof. Given linear arrangements σ_1 of $G[C_1]$ and σ_2 of $G[C_2]$, we can produce a linear arrangement $\sigma = \sigma[\sigma_1, \sigma_2]$ for G by defining:

$$\sigma[\sigma_1, \sigma_2](v) = \begin{cases} \sigma_1(v) & \text{if } v \in C_1 \\ \sigma_2(v) + |C_1| & \text{if } v \in C_2 \end{cases}$$

Hence,

$$\begin{aligned} \sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| &= \sum_{\{u,v\} \in E \cap C_1} |\sigma(u) - \sigma(v)| + \sum_{\{u,v\} \in E \cap C_2} |\sigma(u) - \sigma(v)| \\ &= \sum_{\{u,v\} \in E(G[C_1])} |\sigma_1(u) - \sigma_1(v)| + \sum_{\{u,v\} \in E(G[C_2])} |\sigma_2(u) - \sigma_2(v)| \end{aligned}$$

This shows that $\lambda(G) \leq \lambda(G[C_1]) + \lambda(G[C_2])$.

Let σ be an arrangement of G. For $v \in C_i$, let $c(v)$ denote the number of vertices u from C_{3-i} such that $\sigma(u) < \sigma(v)$. Then, set $\sigma_i(v) = \sigma(v) - c(v)$ to define σ_i as an arrangement of the vertices from C_i. Consider the arrangement $\sigma' = \sigma[\sigma_1, \sigma_2]$: it is not hard to see that

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| \geq \sum_{\{u,v\} \in E} |\sigma'(u) - \sigma'(v)|$$

which proves $\lambda(G) \geq \lambda(G[C_1]) + \lambda(G[C_2])$. ∎

This shows that:

Lemma 3.5 LINEAR ARRANGEMENT *remains $\mathcal{NP}$-hard when restricted to connected graphs.*

Proof. (Sketch) Due to (repeated applications of) Lemma 3.4, any (efficient) algorithm for determining $\lambda(G)$ for connected graphs G can be used to efficiently solve the general problem, which was shown to be $\mathcal{NP}$-hard in [200]. ∎

Our preceding reasoning allows us to state:

Theorem 3.6 LINEAR ARRANGEMENT*, restricted to connected graphs, is $\mathcal{NP}$-complete but belongs to $\mathcal{FPT}$.*

Proof. We already argued for the $\mathcal{NP}$-hardness. Membership in $\mathcal{NP}$ is of course trivially inherited from LA. Membership in $\mathcal{FPT}$ can be seen as follows: after applying rule 7, we will have an instance $((V, E), k)$ with $|V| \leq k + 1$ (unless already solved). Hence, that rule is a kernelization rule, so that Theorem 2.4 applies. ∎

We mention this restricted problem here, since it is a problem where one can actually argue that the standard parameterization is not very natural: the reduction rule we employed is simply answering that there cannot be any solution if the parameter size is small (related to the number of vertices in the graph). So, small parameter values do not exist. This is a bit different in the general setting of LINEAR ARRANGEMENT which is treated in more details in Chap. 4. However, the basic criticism regarding the standard parameterization is valid there, as well, since it is based on the following proposition and an according reduction rule that generalize Prop. 3.3. Put in another way: due to the lower bounds expressed in the propositions, it might make more sense to consider a parameterization via the number of vertices or by the number of edges instead of taking the entity to be minimized as the parameter. However, this kind of parameterization would generally be not regarded as belonging to the realm of parameterized algorithmics, but rather to exact algorithmics (for hard problems) in a broader sense, see Chap. 10.

Proposition 3.7 *If $G = (V, E)$ is a graph, then $\lambda(G) \geq |E|$.*

Proof. According to the formula how to compute the linear arrangement cost, each edge contributes at least a cost of one, since the mapping σ that associates vertices and natural numbers is one-to-one and the graphs we consider have no loops. ∎

Reduction rule 8 *If (G, k) is an instance of* LINEAR ARRANGEMENT*, then return* NO *if $k < |E|$.*

Other lower bounds on the size of the linear arrangement number of a graph are discussed in [321] from a very practical perspective. Other (theoretical) bounds are obtained in [5, 277]. In [321], it is also mentioned that LINEAR ARRANGEMENT is only known to be approximable within a logarithmic factor (and even that algorithm is pretty impractical for sufficiently large graph instances, since it involves solving a linear program with an exponential number of variables), although the related problem of crossing minimization in layered graphs (as treated in Chap. 6) can be approximated within a small constant factor; take [301] as a recent reference.

In fact, many other similar problems are listed in the given references [199, 200, 277]; mostly, they can be shown to be parameterized tractable by using similar techniques; we mention in particular GT 43 and GT 45 from [199]. However, there is one notably exception, namely GT 40:

Problem name: BANDWIDTH (BW)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \ldots, |V|\}$ such that $\forall\{u, v\} \in E : |\sigma(u) - \sigma(v)| \leq k$?

Recall that a bandwidth-minimal ordering (say upperbounded by k) of the vertices is an ordering where the corresponding adjacency matrix of the graph has bandwidth upperbounded by k; hence the name.

Let us have a look at our example first:

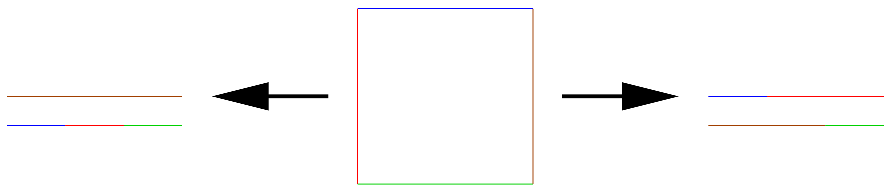


Figure 3.3: One more way of mapping C_4

Example 3.8 In Fig. 3.2, we used color coding to visualize two possible mappings of C_4. As can be seen, the maximum length of the color interval is the “bandwidth of the drawing.” But what is the bandwidth of the graph? Both drawings have one worst color of length three. Fig. 3.3 shows that there are better drawings. The one to the right in that figure is in fact optimal, since not all edges can be mapped on intervals of length one for a C_4.

Notice that our argument from Lemma 3.4 is still true in this case. Hence, BW, restricted to connected graphs, is also $\mathcal{NP}$-complete. However, we cannot rule out any small value of the parameter k that easily. Even worse: this problem is known to be W[t]-hard for all natural numbers t, see Chap. 9 for the hardness notions in parameterized complexity. This is quite unfortunate in the sense that there exist nice polynomial-time algorithms for solving (in general) $\mathcal{NP}$-hard graph problems on graphs of bounded bandwidth, see [297].

The similarity between LINEAR ARRANGEMENT and BANDWIDTH can be best seen if one considers the following generalization, based on a given *vector norm* $\| \cdot \|$ on vectors of numbers; to this end, we extend a mapping $\sigma : V \to \{1, \dots, |V|\}$ to a mapping $\sigma_E : E \to \{0, \dots, |V| - 1\}$ by setting $\sigma_E(\{u, v\}) = |\sigma(u) - \sigma(v)|$. If we now assume an arbitrary but fixed ordering of the edges, i.e., $E = \{e_1, \dots, e_m\}$, then we can associate to σ a vector, also written σ_E for brevity, defined as:

$$\sigma_E = (\sigma_E(e_1), \dots, \sigma_E(e_m)).$$

Problem name: LINEAR ARRANGEMENT (GENERALIZED TO A VECTOR NORM $\| \cdot \|$)
Given: A graph $G = (V, E)$, $E = \{e_1, \dots, e_m\}$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that $\| \sigma_E \| \leq k$?

Observe that both LINEAR ARRANGEMENT and BANDWIDTH are special cases by choosing $\| \cdot \|$ to be the sum norm $\| \cdot \|_1$ and the maximum norm $\| \cdot \|_\infty$, respectively. What about the usual "intermediate norms" $\| \cdot \|_p$ for natural numbers $1 < p < \infty$? Recall that

$$\| (x_1, \dots, x_n) \|_p = \sqrt[p]{\sum_{i=1}^{p} |x_i|^p}.$$

These problems are also known as *minimum p-sum problems.*

To continue with our example, let us focus on $p = 2$:

Example 3.9 The arrangement on the left-hand side of Fig. 3.2 yields a 2-sum of

$$\sqrt{1^2 + 1^2 + 1^2 + 3^2} = \sqrt{12}.$$

On the right-hand side of that figure, the arrangement has a 2-sum of

$$\sqrt{2^2 + 1^2 + 2^2 + 3^2} = \sqrt{18}.$$

The arrangement on the right-hand side of Fig. 3.3 however has a 2-sum of

$$\sqrt{2^2 + 1^2 + 2^2 + 1^2} = \sqrt{10}.$$

The reader might wish to see if there are arrangements with smaller 2-sum numbers.

For the corresponding linear arrangement numbers λ_p, the following generalizations of earlier results are easy to obtain:

Proposition 3.10 *If $G = (V, E)$ is a graph, then $\lambda_p(G) \geq \sqrt[p]{|E|}$.*

Reduction rule 9 *If (G, k) is an instance of* LINEAR ARRANGEMENT *(generalized to a vector norm $\|\cdot\|_p$), then return* NO *if $k < \sqrt[p]{|E|}$.*

Hence, we can state the following result:

Theorem 3.11 *For each fixed natural number p, the generalized* LINEAR ARRANGEMENT *problem (based on norm $\|\cdot\|_p$), lies in $\mathcal{FPT}$.*

Proof. Reduction rule 9 provides a kernel of size $|k|^p$ for the generalized problem, which is showing $\mathcal{FPT}$-membership when p is assumed to be a fixed constant; here, the problem size is measured in terms of the number of edges of the input graph. ∎

Remark 3.12 *According to [239], the (classical) complexity status of the generalized* LINEAR ARRANGEMENT *problem (in actual fact, this is a family of problems indexed by p) is open if $p \notin \{1, \infty\}$.*

However, for a different set of problems (namely scheduling problems), the question of finding approximation algorithms that are good with respect to different norms has been recently addressed in [31], so that a closer scrutiny to this type of problem is not too artificial, keeping in mind that for practical purposes different metrics might indeed exhibit different behaviors.

Some of the problems (for $p \notin \{1, \infty\}$) were addressed in [51, 249, 277]. From a parameterized perspective, it would make also sense to consider the natural number p as part of the input; the proof of Theorem 3.11 only works when p is not part of the input. Hence, the parameterized status of this problem is open. Of course, if $p = \infty$ is also allowed as input, then it is clear that such a problem would be at least as hard as BANDWIDTH, so that membership in $\mathcal{FPT}$ would be very unlikely.

Two further generalizations (motivated from practical applications) are mentioned in [5, 277]:

- Edges could be given integer weights ≥ 1 to model the possibility of multiple connections between points or of different flow capacities.
- Instead of edges, hyperedges could be considered. Given an arrangement σ, the cost of drawing a hyperedge h would then be measured as

$$\sigma_E(h) = \max_{x \in h} \sigma(x) - \min_{y \in h} \sigma(y).$$

The cost of a drawing (generalizing the LA problem) would then be

$$\sum_{h\in E}\sigma_E(h)$$

for a given hypergraph $G=(V,E)$ where, for all $h\in E$, $|h|\geq 2$, since smaller hyperedges can be removed from a given instance without changing the parameter (in other words, we have found a reduction rule). As can be easily seen, all kernelization results stated above transfer to this case, as well.

We can therefore conclude:

Corollary 3.13 LINEAR ARRANGEMENT, *generalized to the (integer-)weighted case or to the case allowing hyperedges (also in combination), lends to problems in $\mathcal{FPT}$, even if we generalize the allowed vector norms as in Thm. 3.11.*

Let us describe one of the practical motivations to consider generalizations of LINEAR ARRANGEMENT a bit more in detail, since this also gives a good example how to derive graph-theoretical formalizations from problems that were originally not stemming from graph theory itself.

Example 3.14 Consider the following placement problem occurring in VLSI, inspired from [277]: we are given a set of *modules* and a *wiring prescription* telling us which modules are to be connected; this information can even come as a kind of wiring proposal, see Fig. 3.4. In that picture, the modules are drawn as boxes labeled A through E. The wires are drawn as straight-line segments. For better readability of the further solutions, they have different colors. The connection points are drawn as filled circles. The question is if the proposed arrangement of modules is best. In other words, is it possible to reorder the modules to improve the arrangent given in the proposal?

What is our criterion of optimality? It could be at least three things:

1. The total wire length should be minimal.

   Such an optimality might help save fabrication costs. Moreover, keep in mind that long wires mean long run times of signals on these wires.

2. The wire length between any two points connected by a wire should be minimal.

   The overall run-time behavior of a manufactured chip is determined by its weakest spot, i.e., the length of the longest point-to-point interconnection.

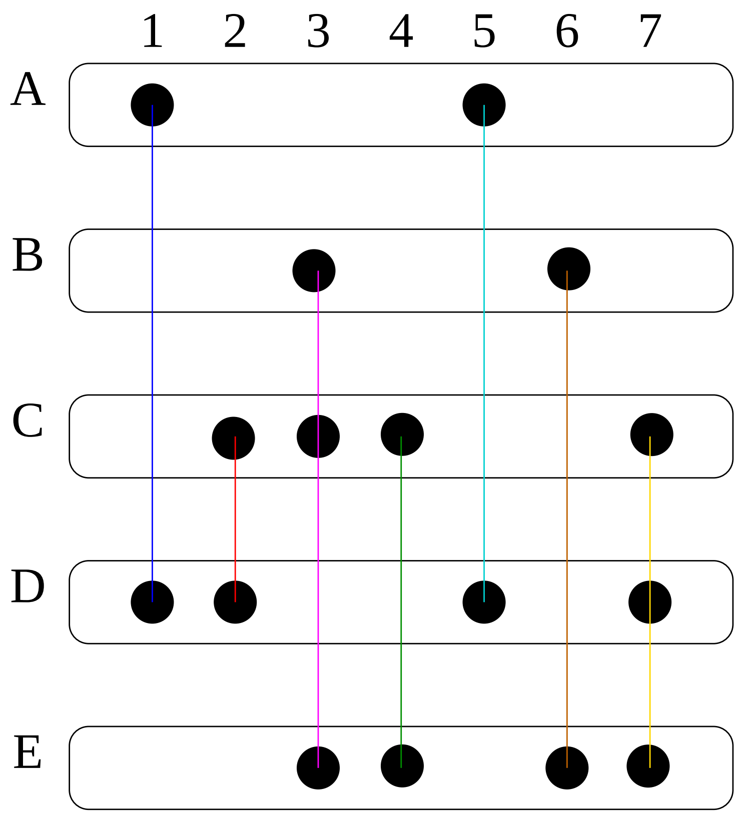


Figure 3.4: A wiring proposal for some modules

3. Observe that we used one column for each wire in Fig. 3.4. From the viewpoint of fabrication, it would be however desirable to use as few of these columns (usually called *tracks* in these context) as possible.

How do these VLSI problems relate to the graph problems we considered so far? Let us see this with the first of the mentioned problems, that can be formalized as follows:

Problem name: MODULE PLACEMENT PROBLEM (MPP)
Given: a set of modules M, a set of wires W connecting modules, i.e., each wire $w \in W$ is a subset of M
Parameter: a positive integer k
Output: Is it possible to find a mapping $\sigma : M \to \{1, \dots, |M|\}$ such that the overall wire length is less than or equal to k?

If we measure the overall length of the wires by

$$\sum_{w \in W} (\max_{x \in w} \sigma(x) - \min_{y \in w} \sigma(y)),$$

then the corresponding to the LINEAR ARRANGEMENT problem on hypergraphs introduced above should be clear: modules correspond to vertices, and wires are the hyperedges. Observe that wires may connect several modules, so that we actually do have hyperedges.

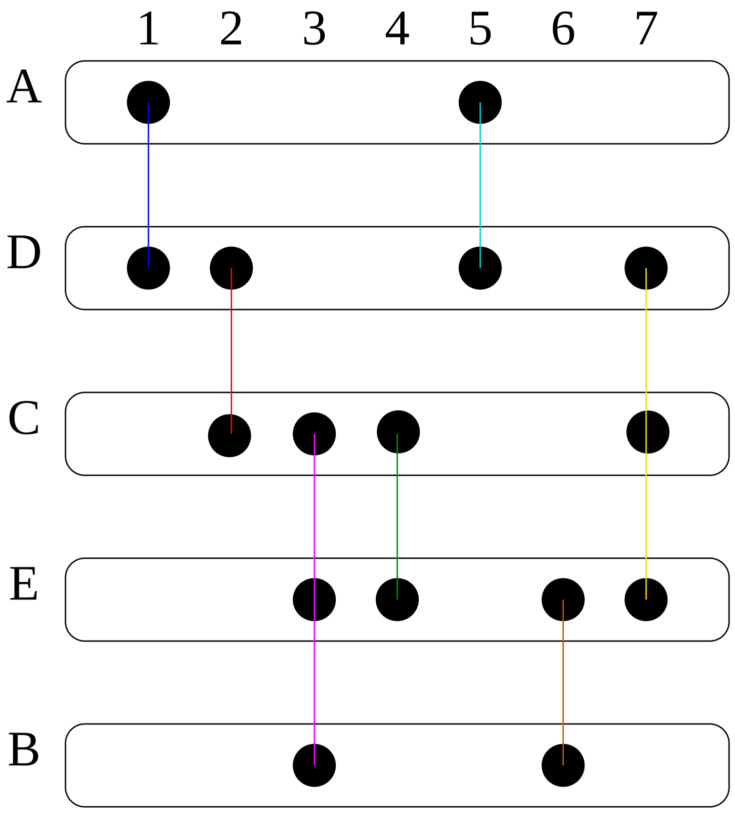


Figure 3.5: An optimal wiring...

Is the arrangement in Fig. 3.4 optimal, when seen as the input of MPP? A better arrangement is proposed in Fig. 3.5; optimality of this arrangement can be easily seen by observing that the distance between two points that are to be connected by a wire is always one, except for "real hyperedges." Therefore, this arrangement is also incidentally optimal for the second criterion, which is (as the reader might have observed) corresponding to determining the *bandwidth of hypergraphs*; we omit details here. Of course, since already BANDWIDTH is parameterized intractable, we cannot hope for an $\mathcal{FPT}$ result in this more general case.

We finally observe that the wiring from Fig. 3.5 also minimizes the third entity we mentioned, namely the number of tracks; some of the wires can be now put on the same track, as indicated in Fig. 3.6.

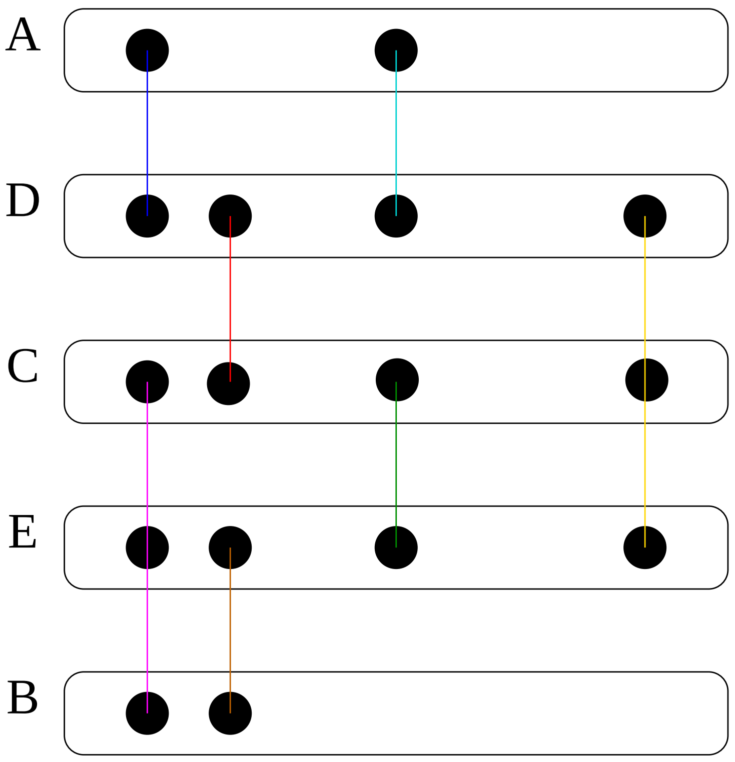


Figure 3.6: …using less tracks

Let us finally mention that *cutwidth* is also a related parameter:

Problem name: CUTWIDTH (CW)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that $\forall 1 \leq i < |V| : |\{\{u, v\} \in E \mid \sigma(u) \leq i < \sigma(v)\}| \leq k$?

In actual fact, CUTWIDTH is also known as the "cut linear arrangement problem," see [169, 367]. Notice that also cutwidth tries to measure the "path-similarity" of a graph; in fact, it is known that the pathwidth of a graph is upperbounded by its cutwidth (see Chap. 7 and [129, 258, 367]).

More precisely, Kinnersley [258] considers the problem of determining the *vertex separation number* of a graph. Given a graph $G = (V, E)$ and a *linear layout* σ of G, i.e., a bijective mapping $\sigma : V \to \{1, \dots, |V|\}$, let

$$V_\sigma(i) = |\{u \in V \mid \exists\{u, v\} \in E : \sigma(u) \leq i < \sigma(v)\}|.$$

The vertex separation number of the linear layout σ of G is then

$$\mathrm{vs}_\sigma(G) = \max_{1\leq i<|V|} V_\sigma(i),$$

and the vertex separation number of G is

$$\mathrm{vs}(G) = \min\{\mathrm{vs}_\sigma(G) \mid \sigma \text{ is a linear layout of } G\}.$$

Kinnersley has shown that the vertex separation number of a graph equals its pathwidth (as defined in Chap. 7). Conversely, the *cutwidth* of a graph $G = (V, E)$ could be described as follows: Let

$$E_\sigma(i) = |\{\{u, v\} \in E \mid \sigma(u) \leq i < \sigma(v)\}|.$$

Then define

$$\mathrm{cw}_\sigma(G) = \max_{1\leq i<|V|} E_\sigma(i),$$

and the cutwidth of G is

$$\mathrm{cw}(G) = \min\{\mathrm{cw}_\sigma(G) \mid \sigma \text{ is a linear layout of } G\}.$$

Since for all layouts and all positions

$$V_\sigma(i) \leq E_\sigma(i),$$

$$\mathrm{vs}(G) \leq \mathrm{cw}(G)$$

follows.

With the standard parameterization, the CUTWIDTH problem is known to be in $\mathcal{FPT}$, see [134, 169].

Let us have a look at some examples to better understand these concepts:

Example 3.15 In Fig. 3.3, we used color coding to visualize two possible mappings of C_4. For better readability, the intervals that correspond to edges are drawn in different "bands." As can be seen, the number of necessary bands for such drawings is just the cutwidth of the layout, which happens to be the same in both cases. Both are in fact optimal arrangements with respect to cutwidth.

Incidentally, also the vertex separation number of both layouts is two.

Example 3.16 Consider the family of *star graph*s S_n with vertices $\{1, \ldots, n\}$ and edges $\{1, i\}$ for $i = 2, \ldots, n$. Imagine 1 to be drawn in the center and i for $i \neq 1$ being the rays of the star.

The identity then provides a layout with vertex separation number one. However, the cutwidth of that layout is $n-1$. A better layout (with respect to cutwidth) can be produced by mapping 1 in the center of the image, placing half of the ray vertices to the left and the other one (arbitrarily) to the right of the center vertex; the cutwidth of this layout (which is now equal to the vertex separation number of the layout) is then $\lfloor n/2 \rfloor$. This also gives the optimal arrangement for star graphs when considering them as inputs for LINEAR ARRANGEMENT, see [277, Cor. 2.4].

Let us briefly re-examine Ex. 3.14 from the point of view of cutwidth; at first glance, it might appear that the third mentioned goal in the module placement scenario would correspond to something like CUTWIDTH in hypergraphs. However, if this were the case, then the second and third tracks (where the wires actually only have two connection points each, so that they actually correspond to edges) in Fig. 3.6 could be merged; this is not the case (for easily seen physical reasons). Hence, we would have as optimization goal to minimize (in the formulation of MODULE PLACEMENT PROBLEM):

$$\max_{1 \leq i \leq |M|} |\{w \in W \mid \min_{m \in w} \sigma(m) \leq i \leq \max_{m \in w} \sigma(m)\}|.$$

3.3 Alternative parameterizations

As we have seen above, the standard parameterization is not the only way to parameterize problem, even though they might be given as optimization problems; it is clear that decicision problems that don't correspond to or stem from optimization problems have no standard parameterization, so that the ideas we present here will apply in that scenario, as well.

Let us first discuss one of our favourite problems, again: VERTEX COVER. Mohar [294] had shown that k-VERTEX COVER RESTRICTED TO (PLANAR) GRAPHS OF MAXIMUM DEGREE THREE is $\mathcal{NP}$-hard. This result is as close as $\mathcal{NP}$-hardness could go with respect to the degree bound: for graphs of maximum degree two, the problem becomes easily solvable.[4]

In [220], another idea of parameterization is advocated which they called *distance from triviality*. In this context, the number ℓ of vertices of degree three or more could be such a distance measure. If this parameter is small, testing all 2^ℓ possible assignments of 0 and 1 to the vertices of degree three or more (interpreted as "not going into the cover" or "going into the cover") would allow solving the MINIMUM VERTEX COVER problem in time $\mathcal{O}(2^\ell |G|)$ for each graph G with at most ℓ vertices of degree three or more. More specifically, this shows $\mathcal{FPT}$-membership of the following problem:

[4]Even the more general problem EDGE COVER is solvable in polynomial time, see [311].

Problem name: VERTEX COVER, PARAMETERIZED BY NUMBER OF VERTICES OF DEGREE THREE OR LARGER (VCDEG)
Given: A graph $G = (V, E)$
Parameter: a positive integer ℓ that equals the number of vertices in G of degree three or larger
Output: What is the size of a minimum *vertex cover* $C \subseteq V$?

As another example from the literature (that can be seen as a reparameterization of SET COVER and hence of HITTING SET) we quote [344], where triviality is measured in terms of the so-called (almost) consecutive ones property. Observe that the referenced paper does not explicitly mention parameterized complexity in any way, yet the results can be interpreted this way.

As seen in the previous section (and also in Chap. 4), LINEAR ARRANGEMENT and DOMINATING SET OF QUEENS can be viewed as problems that have trivial kernels if viewed from the perspective of the standard parameterization. This motivates the idea of *parameterizing above guaranteed values*: for the mentioned two problems, this might mean the following:

Problem name: LINEAR ARRANGEMENT (LA), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| \leq k + |E|?$$

Problem name: DOMINATING SET OF QUEENS (QDS), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: An $n \times n$ chessboard C
Parameter: a positive integer k
Output: Is it possible to place $n/2 + k$ queens on C such that all squares are dominated ?

To our knowledge, the status of the two mentioned re-parameterizations is currently unknown.

The idea of *parameterizing above guaranteed values* was actually "born" in the context of maximization problems. More specifically, Mahajan and Raman [285] showed that the following two problems are parameterized tractable by (again) a kind of mathematical "cheating" argument:

Problem name: MAXIMUM SATISFIABILITY (MAXSAT)
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X
Parameter: a positive integer k
Output: Is there an assignment $\alpha : X \to \{0, 1\}$ such that at least k clauses in F evaluate to 1 (true) under α?

and

Problem name: MAXIMUM CUT (MAXCUT)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *cut set* $C \subseteq E$ with $|C| \geq k$, i.e., $(V, V \setminus C)$ is a bipartite graph?

Namely, the following reduction rule is easily seen to be sound for MAXSAT:

Reduction rule 10 *Let F be a CNF formula containing $m = |F|$ clauses, over the set of variables X. If $k < m/2$, then* YES.

Why? Simply pick an arbitrary assignment α. If α evaluates more than half of the formulae to be true, then the YES of the reduction rule is justified by α. Otherwise, consider the assignment $\bar{\alpha}$ that assigns $\bar{\alpha}(x) = 1 - \alpha(x)$, i.e., always the opposite of what α assigns. Then, exactly those clauses that used to be unsatisfied via α are now satisfied with respect to $\bar{\alpha}$. Hence, the rule is justified in this case, as well.

So, we can trivially conclude:

Corollary 3.17 MAXIMUM SATISFIABILITY *has a problem kernel of size $2k$ (where the problem size is measured in the number of clauses).*

Due to the trivial nature of this "reduction," Mahajan and Raman suggested *parameterizing above guaranteed values*, that is, in this case, they studied the following problem variant:

Problem name: MAXIMUM SATISFIABILITY (MAXSAT), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X
Parameter: a positive integer k
Output: Is there an assignment $\alpha : X \to \{0, 1\}$ such that at least $m/2 + k$ clauses in F evaluate to 1 (true) under α?

Mahajan and Raman showed that also this modified problem is fixed-parameter tractable; in fact, their algorithms (in final form based on search-trees and not merely on kernelization) were later improved in [212]. It is worthwhile mentioning that J. Gramm *et al.* also considered different parameterizations (for restricted problems like MAXIMUM SATISFIABILITY on formulae whose clauses all contain at most two literals), namely a parameterization by the number of clauses (or also the number of clauses that contain exactly two literals); however, this could be also considered as being a treatment within the nonparameterized setting, see Chap. 10.

For MAXIMUM CUT, Mahajan and Raman gave a similar argument for proving the soundness of the following reduction rule:

Reduction rule 11 *Let G be a graph containing $m = |E|$ edges. If $k < m/2$, then* YES*.*

Corollary 3.18 MAXIMUM CUT *has a problem kernel of size $2k$ (measured in terms of number of edges).*

Again, the question of parameterizing above guaranteed values was discussed, and the following problem was placed into $\mathcal{FPT}$:

Problem name: MAXIMUM CUT (MAXCUT), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *cut set* $C \subseteq E$ with $|C| \geq |E|/2+k$, i.e., $(V, V \setminus C)$ is a bipartite graph?

These questions are more recently discussed in [323, 322].

A further example of parameterizing above guaranteed values (NONBLOCKER SET) will be discussed in Chap. 4.

We already discussed different parameterizations of optimization problems in the preceding section and will return to the topic in the context of dual parameterizations below.

One might finally ask how one can actually obtain a "good" parameterization for a given problem? From a practical perspective, when facing a concrete problem, it is surely worthwhile analysing which part of the input can be usually considered as small. Observe that this can be very application-dependent, even for the seemingly identical problem. From a theoretical standpoint, bear in mind that anything may serve as a parameter. Of particular interest might be parameters that are hidden in the structure of the problem instance. What this could mean, we will discuss for graph problems in Chap. 7. However, all good intentions for choosing a parameter might also fail if the problem (under the chosen parameterizations) turns out to be parameterized intractable. This possibility is not stressed in this Habilitationsschrift, but it should be kept in mind if trials to find fixed-parameter algorithms fail.

From a philosophical point of view, we might quote Gomes [208], who wrote:

> The ability to capture and exploit problem structure is of central importance, a way of taming computational complexity.

Our tool of taming, i.e., our whip, is the choice of a good parameter. The quality of this choice is highly dependent on our understanding of the problem. Conversely, if a particular choice of a parameter turned out to work very well or very bad, this could also add to our understanding of the structure of the problem at hand. As a guideline for choosing a good parameter, let us once more stress the idea of distance from triviality; in fact, this has been used in various areas without explicit notice of the connection with parameterized algorithmics, partly because the methodologies we mention now do much predate the advent of parameterized algorithmics.

In her recommendable survey, Gomes [208] compares the methodologies employed by the Operations Research (OR) and Artificial Intelligence (AI) communities for solving computationally hard problems. She mentions three main veins to find tractable (sub-)instance of a problem, where tractability refers to polynomial-time computability:

- Linear Programming,
- Network Flow, and
- Horn clause formulations.

What happens, for example, if an OR problem cannot be "completely solved" with linear programming techniques due to additional integrality constraints? Usually, one first ignores these integrality constraints. If the so-called relaxation gives an optimal solution that already complies with the integrality constraints, the problem is solved. If not, then if the number of variables that do not comply with the integrality constraints is small, usually a search tree is started, where in each node of the search tree new Linear Programs are solved by fixing one of the variables with non-integer solutions by rounding the non-integer solutions either up or down (two cases, i.e., two children per internal node of the search tree). If the integrality constraints are not just leaving the choices 0 and 1 (as it is often the case), this idea more generally leads to the *cutting planes* approach, where, e.g., a solution with $x_i = 3.5$ is excluded by introducing the constraints $x_i \leq 3$ and $x_i \geq 4$ in the two branches. From the point of view of parameterized complexity, this approach can be seen as a practical implementation of the idea of parameterizing by the distance from triviality, especially, if the choices of a variable are just 0 and 1, since then the described branching actually decrements the parameter value in each branch. We are not aware of similar approaches in the other two cases, although it seems to be that here is a largely unexplored but promising area of research for people working in parameterized algorithmics. An illuminating worked-out example comparing different approaches to solve a particular problem both by OR and by AI methods can be found in [236].

3.4 Multiple parameters

Sometimes, not only one parameter appears to be natural, but it seems to be reasonable to consider *multiple parameters*, i.e., several parameters that are thought of being "the" parameter at the same time. Abstractly, this poses no problems at all, since a parameter list $\langle k_1, \ldots, k_\ell \rangle$ can be also read as encoding a single (number) parameter.

We shall see, in what follows, a couple of examples showing that multiple parameters do show up in many applications and are hence nothing one should consider to be artificial.

Let us start with examples stemming from VLSI manufacturing. Kuo and Fuchs [267] provide a fundamental study of the *spare allocation problem*. Put concisely, this "most widely used approach to reconfigurable VLSI" uses spare rows and columns to tolerate failures in rectangular arrays of identical computational elements, which may be as simple as memory cells or as complex as processor units (see [267] for details). If a faulty cell is detected, the entire row or column is replaced by a spare one.

This problem can be formally formulated as follows:

Problem name: SPARE ALLOCATION (SAP)
Given: A $n \times m$ binary matrix A representing an erroneous chip with $A[r, c] = 1$ iff the chip is faulty at position $[r, c]$
Parameter: positive integers k_1, k_2
Output: Is there a *reconfiguration strategy* that repairs all faults and uses at most k_1 spare rows and at most k_2 spare columns?

With reconfiguration strategy, we mean a prescription which rows and columns from A have to be replaced by spares.

	1	2	3	4	5	6	7	8	9
1	?			?					?
2									
3	?								
4			?	?			?		
5									
6									
7				?					?

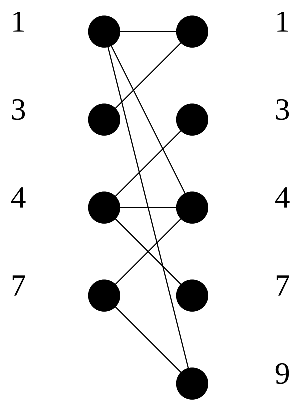


Figure 3.7: A 7×9 array with faults "?" and the bipartite graph model

By considering A as the *bipartite adjacency matrix* of a graph, SPARE ALLOCATION can be easily seen to be parameterized interreducible with the following problem:

Problem name: CONSTRAINT BIPARTITE VERTEX COVER (CBVC)
Given: A bipartite graph $G = (V_1, V_2, E)$
Parameter: positive integers k_1, k_2
Output: Is there a *vertex cover* $C \subseteq V_1 \cup V_2$ with $|C \cap V_i| \leq k_i$ for $i = 1, 2$?

Consider the following example of a reconfigurable array of size 7×9 with 2 spare rows and 3 spare columns in Fig. 3.7, where faulty cells are marked with a "?", together with its corresponding bipartite graph, where, e.g., an edge between 3 and 1 indicates the faulty cell in row 3 and column 1. Herein, we omit isolated vertices.[5] Obviously, the array is repairable using one spare row (replacing row 4) and 3 spare columns (replacing columns 1, 4, 9). This corresponds to a solution of the CBVC problem with $k_1 = 1$ and $k_2 = 3$, as one can easily see. This also reflects the reduction mentioned above.

Typical examples in [267] are arrays with up to 1024 rows and 1024 columns (nowadays possibly even more) and with up to 20 additional spare rows and columns each time (cf. their Table 1), making this problem a natural candidate for a fixed parameter approach, Kuo and Fuchs obtained the following results. First, they showed $\mathcal{NP}$-completeness for CONSTRAINT BIPARTITE VERTEX COVER by reducing CLIQUE to it. Then they present two

[5]Alternatively, this can be seen as a reduction rule.

heuristic algorithms, first a branch-and-bound approach which always delivers optimal results, but is only “efficient for arrays with a moderate number of faulty cells” [267, page 29]. The second one is a polynomial time approximation algorithm. By way of contrast, our $\mathcal{FPT}$-approach always delivers optimal results. Furthermore, its complexity basically depends on the number of available spare rows and columns. More details will be given in later chapters.

It should be noted here that people developing algorithms for VLSI design actually discovered the $\mathcal{FPT}$ concept in the analysis of their algorithms, coming up with $\mathcal{O}(2^{k_1+k_2}k_1k_2 + (k_1 + k_2)|G|)$ algorithms in [222, 278]. They observed that “if k_1 and k_2 are small, for instance $\mathcal{O}(\log(|G|))$, then this may be adequate.” [222, p. 157]. It was in the context of this problem that Evans [158] basically discovered *Buss' rule* to prove a problem kernel for CONSTRAINT BIPARTITE VERTEX COVER. Kuo and Fuchs called this step quite illustratively the “must-repair-analysis.” The reader is encouraged to formulate the necessary reduction rules to prove the following assertion:

Lemma 3.19 CONSTRAINT BIPARTITE VERTEX COVER *has a problem kernel of size* $2k_1k_2$*.*

CONSTRAINT BIPARTITE VERTEX COVER (in the disguise of SPARE ALLOCATION) has important applications especially in the fabrication of high-end VLSI chips: With increasing integration in chip technology, a fault-free chip fabrication can no longer be assumed. So, fault-tolerance has to be taken into consideration within the very fabrication process to obtain reasonable production yields [1, 78]. These ideas are already quite old, see [348] for an early treatment of the topic, but have continued to persist into the most recent developments in VLSI technology. Interestingly, the most challenging (parts of) high-end processors are the seemingly simplest of all possible VLSI chips, namely memories due to their rapidly expanding needs; moreover, e.g., they account for approximately 44% of transistors of the UltraSparc processor, so that they are used as technology and yield drivers [271, 384]. One common solution to increase yield in memory fabrication is to supply a few spare rows and columns to each memory array. These can be used in an internal (on-chip) or external reconfiguration phase (using lasers to “program” fuses) where faulty array rows or columns are replaced by spare ones, cf. some reports on DEC Alpha and UltraSparc processor manufacturing [46, 271, 384].

As pointed out in, e.g., [222], there are several points due to which the problem formulated above is not a completely adequate model:

1. In the manufacturing process, the cost of repairing a chip by using vertical movements of the repair laser may be different from that of

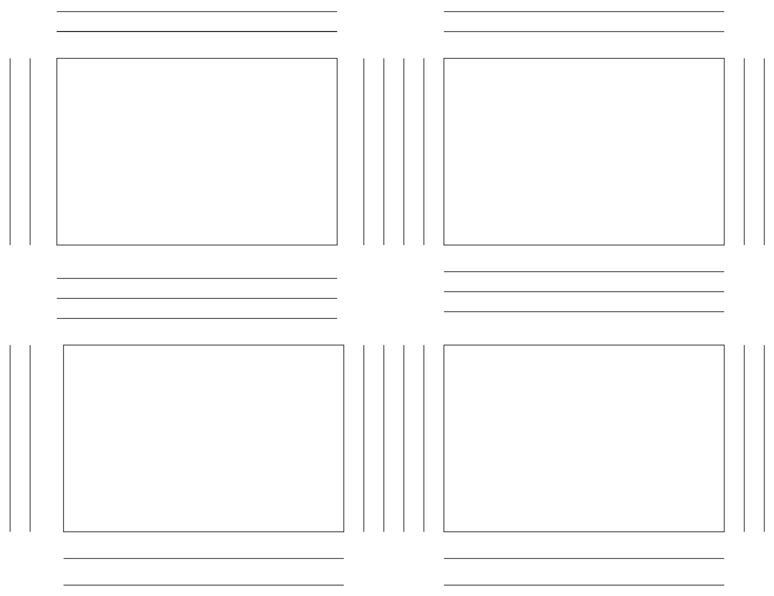


Figure 3.8: Sharing repair resources

horizontal movements. This leads to a sort of weighted variant of CONSTRAINT BIPARTITE VERTEX COVER. The according (pair of standard) parameters would then be budgets for the costs of vertical and horizontal laser movements, again denoted k_1 and k_2.

2. As indicated in Fig. 3.8, a huge memory chip may be split into smaller *blocks*, each of them possibly having its own spare rows and columns. For reasons of economy, other designs are preferred in this case, e.g., each spare row depicted inbetween two memory blocks can be individually used to reconfigure either the block above or the block below it. In other words, in such complex designs, spares may be *shared spares*. These can be seen in Fig. 3.8 schematically drawn inbetween different blocks. Moreover, there may be spare rows or columns which are *linked spares*, which means that such a spare can only be used to reconfigure *one* certain row or column in several blocks. (Linked spares are not shown in the Figure.) Usually, linked spares are not shared, although this could be the case, as well. Obviously, the idea is here to reduce the costs of chip repair.

   This more general scenario introduces a further natural third parameter: the number k_3 of blocks of the array. The parameters k_1 and k_2 should then be understood as the available resources per block. This allows us to formulate CONSTRAINT BIPARTITE VERTEX COVER WITH

SHARED SPARES.

Finally, we could create the problem CONSTRAINT BIPARTITE VERTEX COVER WITH SHARED SPARES AND LINKS with *five* natural parameters, incorporating furthermore the numbers k_4 and k_5 that upperbound the linked spare rows and the linked spare columns, resp., on the whole board.

This example should have made clear that it is quite imaginable that a whole lot of parameters show up in a “real problem” in a fairly natural way. Observe that the basic assumptions concerning parameter values being small in practice is pretty clear in this example, as well, for all five parameters.

Further examples for multi-parameter problems can be found in many application areas. Let us now turn to a more abstract view on this topic.

Typical examples are also delivered by *bicriteria problems*.[6] For example, the following problem that generalizes the usual VERTEX COVER problem is tackled in [234] from the viewpoint of approximation:

Problem name: t-VERTEX COVER (TVC)
Given: A graph $G = (V, E)$
Parameter: positive integers k and t
Output: Is there a *t-vertex cover* $C \subseteq V$ with $|C| \leq k$, i.e., $|\{e \in E \mid C \cap e \neq \emptyset\}| = |E| - |E(G[V \setminus C])| \geq t$?

Of course, choosing $t = |E|$ yields the usual VC problem, where t is the not considered as a parameter. However, although the approximability of TVC is as good as the one of VC, its parameterized complexity status is open (to our knowledge). Observe that this problem is related to PROFIT VERTEX COVER discussed above, since the *profit* is the difference of the gain (lowerbounded by t) and the cost (upperbounded by k) of the cover. Furthermore, notice that t-VERTEX COVER is also naturally related to VERTEX COVER from the point of view of the integer linear program (ILP) related to VERTEX COVERwhich is

$$\min \sum_{i=1}^{n} x_i$$

subject to

$$x_i + x_j \leq 1, \quad \text{for all } v_i v_j \in E$$

[6]Such problems are also sometimes called MAX-MIN problems, see [42].

where $x_i \in \{0,1\}$ is the integrality condition. When we ignore the integrality condition, we arrive at a linear program. The corresponding dual maximization problem would be

$$\max \sum_{j=1}^{m} y_j$$

subject to

$$\sum_{e \text{ incident with } v} y(e) \geq 1 \quad \text{for all } v \in V,$$

where $y(e)$ is the variable y that corresponds to edge e. So, k and t bound the primal and the dual entity to be optimized in a rather natural fashion. This also shows a general technique how to obtain two-parameter versions out of problems that can be formulated as ILP. A similar technique (how to obtain alternative parameterizations by ILP dualization) has been discussed above.

Further rather natural examples from graph theory also concern HITTING SET: namely, as we will see, for each d, d-HITTING SET is fixed-parameter tractable (parameterized by the standard parameter). But, if we consider d to be a parameter, as well, this statement can be extended to this two-parameter version of HITTING SET, giving a more uniform view on this problem. This is discussed more deeply with a related problem motivated from networks in Sec. 6.2.2.

3.5 Dual parameters

To formally specify what we mean by the dual of a parameterized problem, we explicitly need a proper notion of a size function.

Definition 3.20 (Size function) A mapping size : $\Sigma^* \times \mathbb{N} \to \mathbb{N}$ is called a *size function*

- if $0 \leq k \leq \text{size}(I,k)$,[7]
- if $\text{size}(I,k) \leq |I|$ (where $|I|$ denotes the length of the string I) and
- if $\text{size}(I,k) = \text{size}(I,k')$ for all appropriate k, k' *(independence).*

[7]We deliberately ignore instances with $k > \text{size}(I,k)$ in this way, assuming that their solution is trivial. Moreover, it is hard to come up with a reasonable notion of "duality" if larger parameters are to be considered.

"Natural size functions" (in graphs, for example, these are entities as the "number of vertices" or the "number of edges") are independent. We can then also write size(I).[8]

For clarity, we denote a problem $\mathcal{P}$ together with its *size function* size as $(\mathcal{P}, \text{size})$. To the *dual problem* $\mathcal{P}_d$ then corresponds the language of YES-instances $L(\mathcal{P}_d) = \{(I, \text{size}(I) - k) \mid (I, k) \in L(\mathcal{P})\}$. The dual of the dual of a problem with size function is again the original problem. Sometimes, we will call $\mathcal{P}$ the *primal problem* (distinguishing it from $\mathcal{P}_d$). Then, k is the *primal parameter* and k_d is the *dual parameter*.

Let us discuss a couple of examples to explain these notions.

Example 3.21 We first return to d-HITTING SET. The special case $d = 2$ is known as VC in undirected graphs. Let $L(d - \text{HS})$ denote the language of YES-instances of d-HITTING SET. Taking as size function size$(G) = |V(G)|$, it is clear that the dual problem obeys $(G, k_d) \in L(d - \text{HS}_d)$ iff G has an *independent set* of cardinality k_d.

More precisely, if C is a hitting set of size at most $|V| - k_d$ in $G = (V, E)$, then $V \setminus C$ has at least k_d many vertices and is independent, i.e., for all edges $e \in E$, $|(V \setminus C) \cap e| < |e|$.

Example 3.22 In Sec. 2.2, we discussed CLIQUE together with some variants.

- When taking the size function size$(G) = |V(G)|$, CLIQUE and VERTEX CLIQUE COMPLEMENT COVER become parameterized duals of each other.

- When taking the size function size$(G) = |E(G)|$, CLIQUE (EDGE-INDUCED) and CLIQUE COMPLEMENT COVER become parameterized duals of each other.

Example 3.23 Let us turn to DOMINATING SET. Taking as size function size$(G) = |V|$, it is clear that the dual problem obeys $(G, k_d) \in L(\text{DS}_d)$ iff G has a *nonblocker set* N of cardinality k_d, where a set N is a nonblocker set if, for each $v \in N$, we can find a $u \notin N$ such that $\{u, v\} \in E(G)$.

So, more precisely, we deal with the following problem:

Problem name: NONBLOCKER SET (NB)
Given: A graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq k_d$?

[8]Observe that Chen and Flum [91] are using a less restrictive notion of size function when discussing the parameterized complexity of so-called miniaturized problems.

Example 3.24 A cover problem related to VERTEX COVER is the following one:

Problem name: FEEDBACK VERTEX SET (FVS)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *feedback vertex set* of size at most k, i.e.,

$$\exists F \subseteq V, |F| \leq k, \forall c \in C(G)(F \cap c \neq \emptyset)?$$

Here, $C(G)$ denotes the set of cycles of G, where a *cycle* is a sequence of vertices (also interpreted as a set of vertices) $v_0, v_1, \dots, v_\ell$ such that $\{v_i, v_{(i+1) \bmod \ell}\} \in E$ for $i = 0, \dots, \ell - 1$.

This problem is (again) a vertex selection problem. Hence, we naturally take $\text{size}(G) = |V|$ as the size function. Then, the dual problem can be described as follows.

Problem name: VERTEX INDUCED FOREST (ViF)
Given: a (simple) graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *vertex-induced forest* of size at least k_d, i.e.,

$$\exists F \subseteq V, |F| \geq k_d, C(G[F]) = \emptyset?$$

In a similar fashion, one can define the problem FEEDBACK EDGE SET, where we ask if it is possible to turn a graph into a forest by deleting at most k edges. Being an edge selection problem, the natural size function is now $\text{size}(G) = |E|$. The dual problem can be hence called EDGE INDUCED FOREST.

Moreover, notice that both FVS and FES can be seen as *graph modification problems*: namely, the task is to find a set of vertices (or edges) whose removal makes the graph cycle-free. The complexity status of these problems different, however: FVS is known to be in $\mathcal{FPT}$; only recently, independently two groups of researchers have found $\mathcal{O}^*(c^k)$ algorithms for FVS for some constant c.[9]

[9]J. Gramm and F. Rosamond, independent personal communications; the work about which we were informed by F. Rosamond will appear in the Proceedings of COCOON 2005.

FEEDBACK EDGE SET can be solved in polynomial time: finding an edge-induced forest will be done by any spanning forest algorithm. Hence, EDGE INDUCED FOREST is a trivial problem, as well.

By way of contrast, VERTEX INDUCED FOREST is parameterized intractable see [257], also cf. Chap. 9.

A somewhat related problem is the following one:

Example 3.25 Above, we already discussed the problem MAXIMUM CUT. There, the task (seen as a maximization problem that is parameterized in the standard way) is to partition the vertices of a graph $G = (V, E)$ into two sets V_1 and V_2 so that the number of edges in

$$\{\{v_1, v_2\} \in E \mid v_i \in V_i \text{ for } i = 1, 2\}$$

is maximized. The natural dual problem (as also discussed in [347]) would therefore be:

Problem name: BIPARTIZATION, EDGE VARIANT (BPEDGE)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *bipartization set* $C \subseteq E$ with $|C| \leq k$ whose removal produces a bipartite graph?

Example 3.26 It is known that the following problem is in $\mathcal{FPT}$, see [57] for the currently best algorithm; the "history" of this problem is best described in [164].

Problem name: MAXIMUM LEAF SPANNING TREE (MAXLST)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *spanning tree* of G with at least k_d leaves?

Being a vertex selection problem, the natural size function is the number of vertices of G, i.e., size$(G) = |V|$. Hence, the dual problem can be described as follows.

Problem name: MINIMUM INNER NODE SPANNING TREE (MININST)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *spanning tree* of G with at most k inner nodes?

In terms of decision problems, the latter problem can be equivalently reformulated as follows, when observing that any set of inner nodes of a spanning tree in a graph G forms a connected dominating set of G, and—conversely—any spanning tree of the graph induced by a connected dominating set D of G can be transformed into a spanning tree T of G with D as the set of inner nodes of T:

Problem name: CONNECTED DOMINATING SET (CDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *connected dominating set* $D \subseteq V$ with $|D| \leq k$, i.e., D is both a connected set and a dominating set?

Without the observations of the preceding paragraph, it is not immediately apparent that MAXIMUM LEAF SPANNING TREE and CONNECTED DOMINATING SET are parameterized duals. M. Fellows mentions in [164] that it is unlikely that CONNECTED DOMINATING SET is parameterized tractable; more specifically, he mentions a W[2]-hardness result, see Chap. 9.

Generally speaking, it is easy to "correctly" define the dual of a problem for selection problems as formalized in [14].

Particularly easy cases (with respect to "correctly" defining the duals) are also bicriteria problems; the following example is of course also a vertex selection problem:

Example 3.27 As seen in Chapter 2, INDEPENDENT DOMINATING SET can be equivalently expressed as the problem MINIMUM MAXIMAL INDEPENDENT SET. Hence, the dual problem can be described as follows:

Problem name: MAXIMUM MINIMAL VERTEX COVER (MMVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a minimal vertex cover set of cardinality $\geq k$?

As detailed in Chap. 9, primal and dual problems often show different behavior when examined from a parameterized perspective, also see [163, 257, 324], although this is not alway the case, as we will see in this Habilitationsschrift. The first observation in this direction was probable made in [285]: they observed that the parameterized dual of MAXIMUM SATISFIABILITY is W[2]-hard (as problem size measure of a given CNF input formula, the number of clauses is taken). This paper also contains an example where (at that time) the parameterized status of the primal problem was settled (containment within $\mathcal{FPT}$), while the dual seemed to be open, namely MAXIMUM CUT; here the problem size is measured in terms of the number of edges of the graph. Observe that the dual can be seen as an "edge-version" of BIPARTIZATION discussed in Chap. 8. There, we also explain how recent algorithmic developments actually also place the dual of MAXIMUM CUT into $\mathcal{FPT}$, see Cor. 8.5.

In fact, it might be that it is possible to find parameterizations so that both primal and dual problems show the same quality from a parameterized perspective, namely $\mathcal{FPT}$-membership: the typical *phase transition* scenario (as described in [295], also see [208]) might be paraphrased as follows: if there are very few (even relatively few) clauses in a 3-SAT instance, then the corresponding instance is almost certainly satisfiable, and a satisfying assignment can be rapidly found; conversely, if there are very many clauses in a 3-SAT instance, then such an instance tends to be unsatisfiable, and this can be quickly shown, too. This observation may be a fruitful future research branch.

As an aside, we mention that in [295] it is also discussed for 3-SAT (CLAUSE) how the fraction of clauses that have three literals influences the hardness of typical instances. This might motivate the following problem:

Problem name: SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) (3-SAT)
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X, each clause having at most three literals
Parameter: a positive integer k, upperbounding the number of clauses of size three
Output: Is there a satisfying assignment $\alpha : X \to \{0, 1\}$ for F?

In fact, this problem can be also seen as a further example of the distance from triviality approach to parameterizing problems. If one could find a theoretical justification of the observation that, if the fraction of clauses with three variables is less than .4 when compared with the number of all clauses in a given SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) instance, then the instance can be quickly solved, then one might even think about re-parameterizing the problem SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION), by taking as parameter only the number of clauses that excess the mentioned fraction of "large clauses" that can be still tolerated; this would surely make sense from a practical point of view.

Remark 3.28 *We also like to mention that basically the notion of duality has been also developed in the area of approximation algorithms under the name of* differential approximation*.*

Recall that, in the case of minimization problems like MINIMUM VERTEX COVER*, the quality of a solution m_A found by algorithm A is usually measured against the value m^* of an optimal solution by the* performance ratio *$m_A(I)/m^*(I)$ for an instance I. So, in the case of* MINIMUM VERTEX COVER*, (rather trivial) algorithms A provide a performance ratio that is upperbounded by two, i.e., $m_A(I)/m^*(I) \leq 2$ for all graph instances I. Differential approximation offers an alternative way to measure the quality of a solution. Let $\omega(I)$ denote the value of the worst feasible solution of instance I. In the case of* MINIMUM VERTEX COVER*, for a graph instance $G = (V, E)$, the value of a worst feasible solution is $|V|$. Then, the* differential approximation ratio *δ_A is measured as follows:*

$$\delta_A(I) = \frac{|\omega(I) - m_A(I)|}{|\omega(I) - m^*(I)|}$$

In the case of MINIMUM VERTEX COVER*, observe that A could be easily turned into an algorithm A' for* MAXIMUM INDEPENDENT SET*, if instead of solution C, the complement $V \setminus C$ is returned. Clearly, the size of $V \setminus C$ is exactly $|V \setminus C| = |V| - |C|$. In other words, the differential approximation ratio of A is nothing else than the (inverse of the) performance ratio of A' (the inversion is due to the "inversed definition" of performance ratio for maximization problems). Results on (the more general)* MINIMUM HITTING SET *can be found in [39], where also pointers to the relevant literature can be seen.*

From a parameterized perspective, this means that measuring the differential approximation ratio instead of the performance ratio means to consider the dual of the standard parameterization.

This also gives an idea how to properly define the dual of a weighted problem, as WEIGHTED VERTEX COVER*: the dual parameter would be $k_d = \sum_{v \in V} \omega(v) - k$, where ω is now the vertex weight function. In fact, the value of a maximum independent set of instance $(G = (V, E), \omega)$ is then again the weight of the complement of a minimum vertex cover of that instance.*

We would finally like to point once more to [208], as well as to [236], where the concept of duality is discussed in a very broad sense; this might indeed give additional insights how to find suitable parameterizations for concrete problems.

Chapter 4

Kernels

In [134], the idea to get small problem kernels is somehow depreciated by calling it an *ad hoc method*. Although in some respects this idea still deserves this title (there is still no standard way of arriving at small kernels, but rather a collection of ways to get there), it could be also called the *core methodology* for arriving at parameterized algorithms.[1]

Mathematically, this is justified by Theorem 2.4: a problem is parameterized tractable if and only if it can be kernelized. So, while other methods, especially the search tree method as detailed in Chapter 5, might fail to produce parameterized algorithms for problems in $\mathcal{FPT}$, we can be *sure* that the kernelization technique will always work, at least in principle. Unfortunately, the proof of Theorem 2.4 does not provide efficient kernelizations, i.e., the resulting kernels could be rather huge.

From a practical point of view, kernelization could be also called the core methodology, although in a different sense: most (efficient) kernelizations are based on simple *data reduction* rules, i.e., local heuristics meant to simplify the problem. This can also open up a potentially fruitful dialog between practitioners and mathematicians, since kernelization may be a way to explain why certain heuristics do work so well in practice. This actually justifies the assertion "Eureka, you shrink" by Woeginger [378].

How do we measure the quality of a kernelization algorithm? Formally, this depends on the choice of the function "size" with which we measure the size of an instance. In this chapter, we mostly prefer to give this function in an implicit form. However, if we think ambiguities might arise, we will be more precise.

How can we turn a kernelization algorithm into an $\mathcal{FPT}$ algorithm? Usu-

[1]Notice, however, that in particular for maximization problems that are parameterized in the standard way, ideas from extremal combinatorics do now provide a certain methodological approach for getting small kernels, as exhibited in [323].

ally, after the kernelization phase, there will be an *exhaustive search* phase. This exhaustive phase can be either based on (often quite elaborated) search tree techniques (as detailed in the following Chapter 5) or on *brute force.* Brute force means in our case (i.e., mostly dealing with graph problems that are either vertex or edge selection problems) to try all possible subsets of size k of vertices or of edges. Here, the following combinatorial lemma comes in quite handy. More precisely, for algorithms derived from linear kernels, the following lemma (based on Stirling's formula) is quite handy and often gives a considerably smaller bound than the "naive" 2^{ak}:

Lemma 4.1 *For any $a > 1$,*

$$\binom{ak}{k} \approx a^k \left(\frac{a}{a-1}\right)^{(a-1)k} \leq (ea)^k$$

Proof.

$$\begin{aligned}
\binom{ak}{k} &= \frac{(ak)!}{k!((a-1)k)!} \\
&\approx \frac{\sqrt{2\pi ak}}{\sqrt{2\pi k}\sqrt{2\pi(a-1)k}} \cdot \left(\frac{ak}{e}\right)^{ak} \cdot \left(\frac{e}{k}\right)^{k} \cdot \left(\frac{e}{(a-1)k}\right)^{(a-1)k} \\
&= \frac{\sqrt{a}}{\sqrt{2\pi(a-1)k}} a^k \left(\frac{a}{a-1}\right)^{(a-1)k} \\
&\leq a^k \left(\frac{a}{a-1}\right)^{(a-1)k} \\
&\leq (ea)^k
\end{aligned}$$

for large enough k. (If a is not an integer, we can use general binomial coefficients based on the Gamma function.) ∎

A nice exposition of the current state of the art with many examples can be found in the recent PhD thesis of E. Prieto [323].

In the rest of this Habilitationsschrift, we are mostly sloppy when it comes to the issue of defining kernelizations. The issue of *reduction rule*s and how they lead to kernelizations is discussed in Sec. 4.1. How to actually obtain reduction rules is shown in various examples in the following sections. One of the simplest heuristic strategies is based on greedy approaches. Sometimes, this strategy also works to devise reduction rules, as explained in Sec. 4.2. Another idea to obtain reductions is to look if certain entities are particularly big or small. This is discussed in Sec. 4.3, in particular referring to FACILITY LOCATION and to MAXIMUM KNAPSACK, MINIMUM WEIGHT. Sec. 4.4

describes how to use (rather) deep mathematical results to obtain small kernels, mainly based on variants of coloring theorems. We will also discuss two problems related to DOMINATING SET: NONBLOCKER SET and DOMINATING SET OF QUEENS. Sec. 4.5 revisits again kernelization rules for VERTEX COVER. In Sec. 4.6, we discuss a kernelization example that is far from trivial: PLANAR DOMINATING SET. Sec. 4.7 discusses a novel, still tentative idea in kernelization algorithmics that might be fruitful for future research: kernelization schemes. We conclude this chapter with Sec. 4.8, where we discuss the differences of the approach to preprocessing taken by parameterized algorithmics to the one favored by the theory of compilability.

4.1 Reduction rules

In fact, it will be quite an exception if the kernelization function is explicitly given. This is due to the fact that kernelizations are usually given by a collection $\mathcal{R}$ of so-called *reduction rule*s. Reduction rules are usually simple processing rules that somehow simplify the given instance by local modifications; usually, they can be computed even in linear time. A *reduction rule* R takes the following form, when applied to the instance (I, k): if I satisfies some conditions $c_R(I, k)$, then modify (I, k), yielding $(I', k') = R(I, k)$. We also say that R is an *applicable rule* if $c_R(I, k)$ is true.

An instance to which none of the reduction rules from $\mathcal{R}$ is applicable is also called a *reduced instance* (with respect to $\mathcal{R}$).

Given such a collection $\mathcal{R}$, these rules implicitly define a kernelization by a procedure similar to the one listed in Alg. 5.

Algorithm 5 A generic kernelization algorithm from reduction rules

```
Input(s): a collection $\mathcal{R}$ of reduction rules
Output(s): kernelization function $K[\mathcal{R}]$
  Let $(I', k') = (I, k)$.
  while $\exists R \in \mathcal{R} : c_R(I', k')$ is true do
    Choose some $R \in \mathcal{R}$ such that $c_R(I', k')$ is true
    Update $(I', k') := R(I, k)$
  end while
  return $(I', k')$
```

Let us call a reduction R a *proper reduction* if, for all instances (I, k) on which R is applicable, the resulting instance $(I', k') = R(I, k)$ satisfies

- size$(I') <$ size(I) and $k' = k$, or

- size(I') = size(I) and $k' < k$.

For collections $\mathcal{R}$ of proper reductions, it is clear that the kernelization $K[\mathcal{R}]$ can be computed in polynomial time. It can be also seen that in that case, only *proper kernelization*s will be produced.

Actually, we can relax the condition on $\mathcal{R}$ a bit, still getting polynomial time computable kernelizations $K[\mathcal{R}]$: we only have to ensure that reduction rules that are not proper will be only applied at most some predefined fixed number of times.

Let us make the point that it is crucial that, given a set $\mathcal{R}$ of reduction rules, a successful application of one reduction rule may enable the application of another reduction rule. A good example for this mutual triggering of reduction rules is given in Ex. 2.21 for HITTING SET. This possibility is the reason why special care has to be taken when proving the correctness of reduction rules. Related to HITTING SET, we mention from the literature that the original algorithm for simplifying so-called *Hitting Set trees* as introduced by Reiter [331] turned out to be flawed [214], since the possible interaction between the proposed reduction rules was not properly taken into consideration.

4.2 Greedy approaches

In the case of a standard parameterization (for maximization problems), *greedy algorithms* have turned out to be quite useful.

As a very simple example, let us consider the following problem. Recall that an *independent set* $I \subseteq V$ of a graph $G = (V, E)$ satisfies:

$$\forall u, v \in I : \{u, v\} \notin E.$$

Problem name: PLANAR INDEPENDENT SET (PIS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Furthermore, recall that each planar graph has at least one vertex of degree at most five (this is a consequence of Euler's formula, see [Chapter 11, Site 6]).

Therefore, the following algorithm either finds a sufficiently large independent set or the scanned graph has no more than $6k - 6$ vertices.

The loop invariant [+] can be easily proved by induction, so that the correctness of the algorithm is clear.

Algorithm 6 Greedy kernelization for PLANAR INDEPENDENT SET

Input(s): planar graph $G = (V, E)$, positive integer k

Output(s): either independent set I with $|I| = k$ or $|V| \leq 6(k-1)$

```
    $G' := G$
    $i := 0$
    $I := \emptyset$
    while $V(G') \neq \emptyset$ and $i < k$ do
5:    pick vertex $v \in V(G')$ of smallest degree
      $G' := (G' - N[v])$ $\{|N[v]| \leq 6$ $(*)\}$
      $i := i + 1$
      $I := I \cup \{v\}$
      $\{I$ is an independent set of size $i$ in $G$. [+]$\}$
10: end while
    if $i = k$ then
      $I$ is an independent set of size $k$ in $G$.
    else $\{V(G') = \emptyset$ and $i < k\}$
      {Due to $(*)$, no more than $6(k-1)$ have been taken out before "ex-
      hausting" $V$.}
15: end if
```

Without giving explicit formulations of the algorithms, we mention that a couple of maximization problems, when restricted on planar graphs and parameterized in the standard way, can be solved by greedy kernelization, amongst them MAXIMUM MINIMAL VERTEX COVER and MAXIMUM MINIMAL DOMINATING SET, a problem defined as follows:

Problem name: MAXIMUM MINIMAL DOMINATING SET (MMDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a minimal dominating set of cardinality $\geq k$?

Observe that, in the case of MMDS, we do *not* need a "small degree guarantee" on so-called black-white graphs that are treated in detail in Sec. 5.3.2. Rather, Alg. 6 could be literally seen as a kernelization algorithm for MAXIMUM MINIMAL VERTEX COVER and for MMDS, as well. Namely, if that algorithm (run on graph G) comes up with an independent set I after k steps, then this set can be greedily augmented to give a minimal vertex cover or a minimal dominating set. The only crucial observation is that I does not contain a proper subset already dominating $G[I]$, since $E(G[I]) = \emptyset$. Hence, the answer YES can be justified in that case. In the other case, we have seen that there are less than $6k$ vertices in G.

More uses of the greedy idea can be found in the literature; more specifically, it has been successfully applied, e.g., in [110], to show certain properties of a problem kernel by means of arguments based on so-called *greedy localization.*

4.3 Large is no good (or too good)?!

In fact, the previous section could also take this headline, because there, for maximization problems, a problem that is too large meant it was a trivial YES-instance. (We will more examples along these lines below.) In the case of minimization problems, we rather have a local kernelization rule; then, something *must* happen in order to prevent the instance becoming a NO-instance.

In this spirit, we can also reconsider VERTEX COVER: Buss' rule can be seen as implementing a local version of the idea of tackling "large instances." Namely, here a local version of the idea of looking at large instances applies; a vertex is "large" if its degree is larger than k; such vertices must belong to any k-vertex cover (if such a small cover exists at all). Due to the simplicity of this idea, the same rule basically also applies to enumeration and counting versions of VERTEX COVER, as discussed in Chap. 8. Interestingly, this rule is also valid for different problems, as for the (parametric) dual of the MAXIMUM IRREDUNDANT SET problem, see [135] for details. It is also instructive to observe how Buss' rule is changed for the maximization version MAXIMUM MINIMAL VERTEX COVER: whenever we encounter a vertex v of degree at least k, we now *know* we have a YES-instance of MMVC, since putting $N(v)$ (but not v) into the cover C (and then greedily supplementing C to actually become a valid vertex cover) will justify this answer. However, the small-degree rule would stay the same, since putting an isolated vertex into a cover would render it non-minimal. We summarize our observations below:

Reduction rule 12 *Delete isolated vertices (and leave the parameter unchanged).*

Reduction rule 13 *If v is a vertex of degree greater than k in the given graph instance (G, k), then answer* YES.

Theorem 4.2 MAXIMUM MINIMAL VERTEX COVER *admits a kernel of size k^2 (measured in terms of number of vertices).*

As an aside, let us mention that this puts MAXIMUM MINIMAL VERTEX COVER into $\mathcal{FPT}$, while it is known [134, page 464] that its parameterized dual INDEPENDENT DOMINATING SET is W[2]-complete, i.e., most likely not in $\mathcal{FPT}$.

The possibly simplest example for this principle for obtaining kernels is given by:

Problem name: MINIMUM PARTITION (PART)
Given: A finite set $X = \{x_1, \dots, x_n\}$, a weight function $w : X \to \mathbb{R}_{\geq 1}$
Parameter: $k \in \mathbb{N}$
Output: Is there a set $Y \subset X$ such that

$$\max\{\sum_{y \in Y} w(y), \sum_{z \notin Y} w(y)\} \leq k \text{ ?}$$

The problem is therefore to partition the set X into two bins Y, $X \setminus Y$ such that the accumulated weights in both bins is as close to one half of the overall weight

$$w(X) = \sum_{x \in X} w(x)$$

as possible. Note that we are using the standard parameterization of the minimization problem MINIMUM PARTITION.

Now, consider the following rule designed after the principle of looking at large instances:

Reduction rule 14 *If $k < w(X)/2$, then* NO.

The correctness of this rule is easily seen: if it would be possible to put fractions of items into the bins, then it would be possible to put equal weights in both bins, i.e., $w(X)/2$ weight in each bin. If fractions are disallowed (as according to the problem definition), then the obtained values

$$\max\{\sum_{y \in Y} w(y), \sum_{z \notin Y} w(y)\} \leq k$$

could not decrease.

Lemma 4.3 *If (X, w, k) is an instance of* MINIMUM PARTITION *that is reduced according to Rule 14, then $|X| \leq 2k$ and $w(X) \leq 2k$. Hence, there is a small problem kernel for* PART.

Let us now reconsider now FACILITY LOCATION, using the matrix notation introduced in Sec. 3.2.[2] The following observation is easy but crucial:

Reduction rule 15 (FL-R0) *If a given instance (M, k) with $M \in \mathbb{N}^{(n+1)\times m}$ obeys $n > k$, then* NO.

Lemma 4.4 *Rule 15 is sound.*

Proof. Each customer must be served. Since edges have positive integer weights, each customer thus incurs "serving costs" of at least one unit. Hence, no more than k customers can be served. ∎

Lemma 4.5 *After having exhaustively applied Rule 15, the reduced instance (M, k) will have no more than k rows.*

Proof. By the definition of the problem, each customer has to be served, and serving some customer will cost at least one "unit." Hence, a budget of k units helps serve at most k customers. ∎

A facility location f is described by the vector $v_f = M[0 \dots n][f]$. These vectors can be componentwisely compared.

Reduction rule 16 (FL-R1) *If for two facility locations f and g, $v_f \leq v_g$, then delete g; the parameter stays the same.*

Lemma 4.6 *Rule 16 is sound.*

Proof. Obviously, a solution to the FL-R1-reduced instance is also a solution to the original instance. If we had a solution S to the originally given instance which contains a facility g and there is another facility f such that $v_f \leq v_g$, then a solution S' obtained from S by choosing facility f instead of g (and choosing to serve any customer served by f in S to be served by g in S') will also be a valid solution which comes at no greater cost. This way, we can gradually transform S into a solution which is also a valid solution to the FL-R1-reduced instance and has no larger costs than S. ∎

Central to the complexity class $\mathcal{FPT}$ is the concept of kernelization, since it characterizes $\mathcal{FPT}$. We first provide a "quick classification" of FACILITY LOCATION in $\mathcal{FPT}$ based on well-quasi-orderings.

Theorem 4.7 FACILITY LOCATION *is fixed-parameter tractable.*

[2]What follows is part of unpublished work with M. Fellows.

Proof. We show that, after having exhaustively applied FL-R0 and FL-R1, we are left with a problem of size $f(k)$. Let (M, k) be a (FL-R0,FL-R1)-reduced instance. By Lemma 4.5, we know that M contains no more than k rows, since (M, k) is FL-R0-reduced. Each facility is therefore characterized by a $(k + 1)$-dimensional vector. Since (M, k) is FL-R1-reduced, all these vectors are pairwise uncomparable. According to Dickson's Lemma [127] these could be only finitely many, upperbounded by some function $g(k)$. Hence, M is a matrix with no more than $(k + 1)g(k)$ entries. ∎

The function $f(k)$ derived for the kernel size in the previous theorem is huge, yet it provides the required classification. So, we would not claim that we actually arrived at an efficient algorithm for FL.

To obtain a better algorithm, observe that a solution can be viewed as a partition of the set of all customers into groups such that customers within the same *group* get served by the same facility. In actual fact, a solution specified by the selected facilities and selected serving connections can be readily transformed into this sort of partition. Also the converse is true: given a partition of the set of customers, we can compute in polynomial time which the cheapest way to serve this group by a certain facility is, so that an optimal solution (given the mentioned partition) in the sense of specifying the selected facilities and the chosen serving connections can be obtained.

This model immediately allows to derive the following result:

Lemma 4.8 FACILITY LOCATION *can be solved in time $\mathcal{O}(k^k p(g(k)) + |M|)$ (where p is some polynomial and $g(k)$ bounds the number of facilities) given some instance (M, k).*

Proof. The kernelization rules FL-R0 and FL-R1 can be applied in time $\mathcal{O}(|M|)$. Then, we have to check all partitions (there are actually $o(k^k)$ many of them; more precisely, this is described by Bell's number whose asymptotics is due to de Brujn (1958), see [Chapter 11, Site 2]) and for each partition, we have to compute its cost incurred by the assumption that each group in the partition is served by one facility. Hence, per partition $p(g(k))$ computations have to be performed. ∎

Still, this algorithm is practically useless due to the huge constants. Let us now develop a better algorithm.

Let us first focus on kernelization.

Reduction rule 17 (FL-R2) *If in an instance (M, k) there is a customer such that the total cost for serving this customer c (given by the sum of installation cost of a particular facility f and the serving cost of c via f) is larger than k, no matter which facility is picked, then* NO.

The following is obvious.

Lemma 4.9 *FL-R2 is sound.*

Lemma 4.10 *An instance (M, k) of* FACILITY LOCATION *that is reduced according to FL-R0, FL-R1, and RL-R2 obeys $|M| \leq k^{3k+1}$.*

Proof. Let (M, k) be an instance reduced according to FL-R0, FL-R1 and FL-R2. Each customer can assign, from its personal perspective, a two-dimensional vector to each facility, consisting of the costs of the facility and the costs of the corresponding service connection. The sum of both costs is bounded by k due to FL-R2. In the view of a particular customer, two facilities are indistinguishable if they incur the same vector. There are no more than k^3 many of these vectors. Therefore, k^{3k} many groups of facilities may occur, such that two facilities in each group are indistinguishable by any of the customers. Since (M, k) is FL-R1-reduced, each such group contains at most one facility. Hence, M has no more than k^{3k+1} many entries. ∎

Let us now turn our attention to one of the two variants of MAXIMUM KNAPSACK discussed in Chap. 3:

Problem name: MAXIMUM KNAPSACK, MINIMUM WEIGHT (KSMW)
Given: n items $\{x_1, \ldots, x_n\}$ with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k. All numbers are natural numbers encoded in binary.
Parameter: b
Output: Is there a subset of items which yield a profit larger than k and has an overall size of less than b?

We show by elaborating on MAXIMUM KNAPSACK, MINIMUM WEIGHT how some relatively simple combinatorial observations can be used to obtain a kernel. Notice that, along with the combinatorics, we will develop some reduction rules of the form:

```
if a certain entity is too big then
  NO
end if
```

We will not specify these rules in explicit form.

First of all, one can get rid of all items whose size is larger than b, because they will never belong to any feasible solution.

Then, there can be at most one item of size b, one item of size $b-1$, ..., and at most one item of size $\lceil (b+1)/2 \rceil$. Moreover, there can be at most two items of size $\lfloor b/2 \rfloor$, ..., up to size $\lceil (b+2)/3 \rceil$, etc. Hence, if there are say more than two items of size $\lfloor b/2 \rfloor$, then we only have to keep the two most profitable of those. This rule leaves us with a collection of $f(b)$ items in the reduced instance.

However, this does not directly give us a problem kernel, since there are two more things to be specified: the sizes of the items and their profits. Since we already removed all items of size larger than b, the sizes of the items can be specified with $\log(b)$ bits each.

As to the profits, the following two observations are helpful:

- An instance I' of MAXIMUM KNAPSACK, MINIMUM WEIGHT that is obtained from another instance I of MAXIMUM KNAPSACK, MINIMUM WEIGHT by simple scaling of all profits by a constant scaling factor c has an optimal solution of value p' iff I has an optimal solution of value p with $p' = cp$. The number of bits needed to specify I' is only linearly dependent on the number of bits needed to specify I and of c. We can therefore assume that all profits are integer values greater than or equal to one.

- If the largest profit p_i among all items is bigger than the sum of the profits of all other items, then the item x_i must be put into any solution of maximum profit. We can hence continue with the task of solving the "remaining" instance, where the parameter b would be reduced by the size s_i of x_i.

  So, if $p_1, p_2, \dots, p_\ell$ with $\ell \leq f(b)$ are sorted increasingly according to size, then $p_1 = 1$, and $p_j \leq \sum_{i=1}^{j-1} p_i$ for $j = 2, \dots, \ell \leq f(b)$. Hence, also the largest profit is upperbounded by some function $g(b)$.

This allows us to conclude:

Theorem 4.11 MAXIMUM KNAPSACK, MINIMUM WEIGHT *admits a problem kernel; hence, it is solvable in $\mathcal{FPT}$-time.*

The opposite strategy is often also helpful: look into local situations that are small and therefore easy to solve. For example, returning again to the case of VERTEX COVER, we developed reduction rules for vertices of degree zero. Further rules for vertices of degree one or two are useful for the development of search tree algorithms, see Chap. 5; a good overview on VC reduction rules is contained in [3].

4.4 The mathematical way

4.4.1 Colorful kernelizations

Mathematical knowledge sometimes provide kernelization results that are in a certain sense cheating, since they are not requiring any reduction rules but rather tell you that "large instances are trivially solvable" (in case of parameterizations deduced from maximization problems) or that "large instances are impossible." Hence, this section also provides more examples along the lines of the previous one: large is good (or bad).

Let us first turn toward PLANAR INDEPENDENT SET, again. One of the best-known non-trivial results on planar graphs is the following one:

Theorem 4.12 (Four Color Theorem) *Every planar graph is 4-colorable.*

The best WWW resource on this famous theorem is surely [Chapter 11, Site 7]. There, the new proof of that theorem published in [338] is outlined, which is considerably simpler than the original one that was derived by Appel, Haken and Koch in [25, 26]. Besides being a simplification, [338] also contains a rather explicit formulation of a quadratic-time algorithm to construct a 4-coloring, given a planar graph as input.

By definition of a coloring, every vertex set that has the same color is an independent set. This is the basic idea for Algorithm 7:

Algorithm 7 Color-based kernelization for PLANAR INDEPENDENT SET

```
Input(s): planar graph $G = (V, E)$, positive integer $k$
Output(s): either independent set $I$ with $|I| = k$ or $|V| \leq 4(k-1)$
    Find a 4-coloring of $G$, formalized as a mapping $c : V \to \{1, 2, 3, 4\}$.
    Let $V_i = \{v \in V \mid c(v) = i\}$.
    Let $V'$ be the largest among the $V_i$.
    {Each coloring is an independent set.}
 5: if $|V'| \geq k$ then
      return $I \subseteq V'$ with $|I| = k$
    else
      {$\forall$ colors $i$: $|V_i| \leq k$; hence, $|V| \leq 4(k-1)$}
    end if
```

Hence, we can conclude (using as a size function for the (kernel) instance the number of vertices):

Corollary 4.13 *There is a quadratic-time algorithm for deriving a kernel for* PLANAR INDEPENDENT SET *of size at most* $4k - 4$.

Proof. Algorithm 7 satisfies the claim. In particular, since we would choose the biggest amongst the colorings, $|V| \leq 4(k-1)$ whenever the algorithm does not already produce a sufficiently large independent set. ∎

Observe the *either-or structure* of the kernelization algorithm 7:

- either: there is a large enough coloring which we can return as a solution to our problem,
- or: all colorings are small, which means that there are only a few vertices in the graph.

Notice one special feature of this kernelization: if we only aim at a the decision version of PLANAR INDEPENDENT SET (as according to the definition of the problem), we can substitute Alg. 7 by the following rule:

Reduction rule 18 *If (G, k) is an instance of* PLANAR INDEPENDENT SET *and if $|V(G)| > 4k$, then* YES.

This shows that the fact that there is an efficient algorithm for getting a four-coloring is not important to this problem, so that the kernel (as such) can be even derived in linear time.

Observe that Alg. 7 (or the reduction rule just discussed) can again be also read as a $4k$-kernelization algorithm both for MAXIMUM MINIMAL VERTEX COVER and for MAXIMUM MINIMAL DOMINATING SET, following the reasoning given above. More precisely, the idea would be (say for MMVC) to greedily extend the largest color class C to a minimal vertex cover (if $|C| \geq k$, where k is the parameter of the MMVC instance).

Corollary 4.14 *There is a quadratic-time algorithm for deriving a kernel for* MAXIMUM MINIMAL VERTEX COVER*, restricted to planar instances, of size at most $4k - 4$.*

Observe that Theorem 4.2 only delivered a quadratic kernel for general graph instances.

Corollary 4.15 *There is a quadratic-time algorithm for deriving a kernel for* MAXIMUM MINIMAL DOMINATING SET*, restricted to planar instances, of size at most $4k - 4$.*

Other coloring theorems can be used to derive similar results. For example, [3] Ringel [335] considered graphs that corresponds to maps embedded on

[3]This notion is different from so-called map graphs that have been made popular by the papers [92, 93].

a sphere that are defined as follows: the vertices of these graphs correspond to either regions or points that do not belong to the open edges separating to regions. Two regions or points are neighbored if they share an edge; and this gives the edge relation of the corresponding graph.

Theorem 4.16 *Graphs that are derived from maps in the way described above can be colored with at most seven colors.*

In fact, Ringel showed a stronger result by allowing certain crossings in the drawing of the map. We omit details here. Anyhow, Alg. 7 can be easily modified to get a $7k$ kernel for INDEPENDENT SET, restricted to graphs derived from maps as described by Ringel.

Ringel (and later Kronk and Mitchem [265]) also considered a different kind of graph derivable from maps on the sphere, where also edge (boundaries) are considered as vertices of the derived graph; again, a coloring theorem is derived.

This schematics is typical for kernelization algorithms based on theorems known from combinatorics. We will encounter more examples along these lines in what follows.

A similar, color-based linear kernel can be shown for VERTEX INDUCED FOREST, which is:

Problem name: VERTEX INDUCED FOREST IN PLANAR GRAPHS (PViF)
Given: a (simple) planar graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *vertex-induced forest* of size at least k_d, i.e.,

$$\exists F \subseteq V, |F| \geq k_d, C(G[F]) = \emptyset?$$

Namely, Borodin [59] managed to prove the following (former) conjecture of Grünbaum:

Theorem 4.17 *[Borodin] Every planar graph is acyclically 5-colorable.*

To properly understand this theorem, let us introduce the following notion according to Borodin:

Definition 4.18 A coloring of a graph is called an *acyclic coloring*, if every bichromatic subgraph that is induced by this coloring is a forest (i.e., an acyclic graph).

The proof of Theorem 4.17 is constructive and gives a polynomial-time algorithm, but this is not essential, since we could also cast Theorem 4.17 into a reduction rule:

Reduction rule 19 *If an instance (G, k_d) of* VERTEX INDUCED FOREST *has more than $2.5k_d - 2.5$ vertices, then* YES*.*

Algorithm 8 Color-based kernelization for VERTEX INDUCED FOREST

```
Input(s): planar graph $G = (V, E)$, positive integer $k_d$
Output(s): either induced forest $F$ with $|F| = k_d$ or $|V| \leq 2.5(k_d - 1)$
    Find an acyclic 5-coloring of $G$, formalized as a mapping $c : V \to \{1, 2, 3, 4, 5\}$.
    Let $V_{\{i,j\}} = \{v \in V \mid c(v) = i \vee c(v) = j\}$.
    Let $V'$ be the largest among the $V_{\{i,j\}}$.
    {Each bichromatic subgraph induces a forest.}
 5: if $|V'| \geq k_d$ then
      return $F \subseteq V'$ with $|F| = k_d$
    else
      {$\forall$ colors $i, j$: $|V_{\{i,j\}}| \leq k_d$; hence, $|V| \leq 5/2 \cdot (k_d - 1)$}
    end if
```

Algorithm 8 (or likewise the mentioned reduction rule) provides a proof for the following assertion:

Corollary 4.19 *There is a polynomial-time algorithm for deriving a kernel for* VERTEX INDUCED FOREST*, given instance (G, k_d), of size (measured in terms of number of vertices of the reduced graph) at most $2.5k_d - 2.5$.*

Remark 4.20 *There exist different bounds on the size of vertex-induced forests for other classes of graphs, as well. Alon [19] mentions the following bounds and classes:*

- *If G is an n-vertex graph with average degree of at most $d \geq 2$, then it contains a vertex-induced forest with at least $2n/(d + 1)$ vertices.*
- *There is a positive constant b such that, if G is a bipartite n-vertex graph with average degree of at most $d \geq 1$, then it contains a vertex-induced forest with at least $(.5d + e^{-bd^2})n$ vertices.*
- *Furthermore, he reproduces a conjecture of Albertson and Haas who supposed that every planar, bipartite n-vertex graph contains a vertex-induced forest with at least $5n/8$ vertices, a result that would improve*

on Cor. 4.19 in the case of bipartite *planar graph considerably, yielding a kernel of* $1.6k_d$ *for* PViF *restricted to bipartite graphs.*

4.4.2 NONBLOCKER SET

The results of this section are based on unpublished work with F. Dehne, M. Fellows, E. Prieto and F. Rosamond.

We now consider the problem NONBLOCKER SET:

Problem name: NONBLOCKER SET (NB)
Given: A graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq k_d$?

Ore [314] has shown (using different terminology) that NB admits a kernel of size $2k_d$ on graphs of degree at least one. This was improved by McCuaig and Shepherd [291] for graphs with minimum degree two; in fact, their result was a sort of corollary to the classification of graphs that satisfy Ore's inequality with equality. Independently, this result was already discovered by the Russian mathematician Blank [50] more than fifteen years ago, as noticed by Reed in [329].

More precisely, their results reads as follows:

Theorem 4.21 *If a connected graph* $G = (V, E)$ *has miniminum degree two and it not one of seven exceptional graphs (each of them having at most seven vertices), then the size of its minimum dominating set is at most* $2/5 \cdot |V|$.

How can we use the mentioned results to get a small kernel? Obviously, we have to design reduction rules to cope with vertices of small degree. Unfortunately, this is not quite sufficient in our case; more precisely, we don't know how to cope with vertices of degree one. Intuitively, it looks as if one should take (if possible) vertices of degree one into the nonblocker set. In other words, its neighbor should go into the dominating set. But how can we make this sure by means of a reduction? Here, the idea of introducing a so-called *catalytic vertex* comes into play. In our example, a catalytic vertex is a vertex that we assume to go into the dominating set (not to go into the nonblocker set). So, we are rather dealing with the following problem:

Problem name: NONBLOCKER SET WITH CATALYTIC VERTEX (NBCAT)
Given: A graph $G = (V, E)$, a catalytic vertex c
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq k_d$ such that $c \notin N$?

We now propose the following reduction rules:

Reduction rule 20 *Let (G, c, k_d) be an instance of* NONBLOCKER SET WITH CATALYTIC VERTEX*. If C is a complete graph component of G that does not contain c, then reduce to $(G - C, c, k_d - (|C| - 1))$.*

Observe that the previous rule is in particular applicable to isolated vertices. It also applies to instances that don't contain a catalytic vertex. The soundness of the rule is easy to see; a formal proof is contained in [323]. Notice that this rule alone gives a $2k_d$ kernel for general graphs with the mentioned result of Ore (details are shown below).

Reduction rule 21 *Let (G, c, k_d) be an instance of* NONBLOCKER SET WITH CATALYTIC VERTEX*. Whenever you have a vertex $x \in V(G)$ whose neighborhood contains a non-empty subset $U \subseteq N(x)$ such that $N(U) \subseteq U \cup \{x\}$ and $c \notin U$, then you can merge x with the catalytic vertex c and delete U (and reduce the parameter by $|U|$).*

This rule is a bit more tricky, but the basic idea that c is used to annotate a vertex of the dominating set should be clear: it simply makes no sense to put any of the vertices from U into the (minimum) dominating set (that we implicitly construct when building up a (maximum) nonblocker set), since x will dominate at least those vertices as any of the vertices from U would do. In fact, we shouldn't apply that rule for very large neighborhoods, since it might be quite time-consuming to test all subsets as required (in order to make sure that the rule is no longer applicable). However, only small neighborhoods will suffice for our analytic purposes, as we will see.

We have actually found a reduction rule that can also cope with two consecutive vertices of degree two and hence eliminate all exceptional graphs of Theorem 4.21 (since all of them have this property).

However, from the point of view of systematics, it is more interesting how to first of all introduce a catalytic vertex (to produce a genuine NBCAT instance from the original NONBLOCKER SET instance) and then to get rid of the catalysts again (to be able to apply Theorem 4.21:

catalyzation rule If (G, k_d) is an instance of NONBLOCKER SET with $G = (V, E)$, then (G', c, k_d) is an equivalent instance of NONBLOCKER SET WITH CATALYTIC VERTEX, where $c \notin V$ is a new vertex, and $G' = (V \cup \{c\}, E)$.

decatalyzation rule Let (G, c, k_d) be an instance of NONBLOCKER SET WITH CATALYTIC VERTEX. Then, perform the following surgery to get a new instance (G', k'_d) of NONBLOCKER SET:

Connect the catalytic vertex c to three new vertices u, v, and w by edges; moreover, introduce new edges uv and vw. All other vertices and edge relations in G stay the same. This describes the new graph G'. Set $k'_d = k_d + 3$. You may then forget about the special role of the catalyst. The surgery is described in Fig. 4.1.

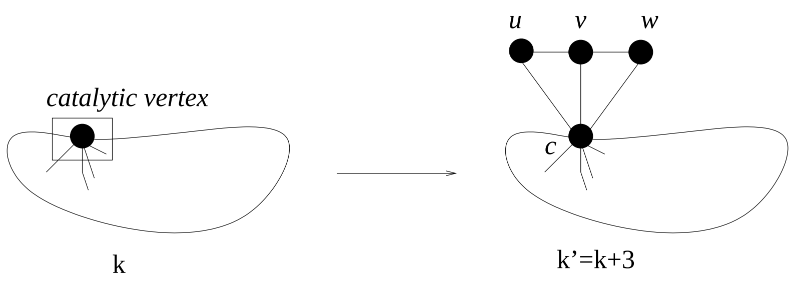


Figure 4.1:

The overall kernelization algorithm for a given NONBLOCKER SET instance (G, k_d) is then the procedure listed in Alg. 9.

Without further discussion, we only mention those reduction rules that can be used to get rid of all consecutive degree-2-vertices in a graph:

Reduction rule 22 *Let (G, c, k_d) be an instance of* NBCAT. *Let u, v be two vertices of degree two in G such that $u \in N(v)$ and $|N(u) \cup N(v)| = 4$, i.e., $N(u) = \{u', v\}$ and $N(v) = \{v', u\}$ for some $u' \neq v'$. If $c \notin \{u, v\}$, then merge u' and v' and delete u and v to get a new instance $(G', c', k_d - 2)$. If u' or v' happens to be c, then c' is the merger of u' and v'; otherwise, $c' = c$.*

Reduction rule 23 *Let (G, c, k_d) be an instance of* NBCAT, *where $G = (V, E)$. Assume that c has degree two and a neighboring vertex v of degree two, i.e., $N(v) = \{v', c\}$. Then, delete the edge vv'. Hence, we get the new instance $((V, E \setminus \{vv'\}), c, k_d)$.*

Algorithm 9 A kernelization algorithm for NONBLOCKER SET

Input(s): an instance (G, k_d) of NONBLOCKER SET

Output(s): an equivalent instance (G', k'_d) of NONBLOCKER SET with $V(G') \subseteq V(G)$, $|V(G')| \leq 5/3 \cdot k'_d$ and $k'_d \leq k_d$ OR YES

```
if $G$ has more than seven vertices then
  Apply the catalyzation rule.
  Exhaustively apply Rules 20 and 21 for neighborhoods $U$ up to size two.
  Apply the decatalyzation rule.
  {This leaves us with a reduced instance $(G', k'_d)$.}
  if $|V(G')| > 5/3 \cdot k'_d$ then
    return YES
  else
    return $(G', k'_d)$
  end if
else
  Solve by table look-up and answer accordingly.
end if
```

Notice that really all cases of two subsequent vertices u, v of degree two are covered in this way:

- If u or v is the catalytic vertex, then Rule 23 applies.
- Otherwise, if u and v have a common neighbor x, then Rule 21 is applicable; x will be merged with the catalytic vertex.
- Otherwise, Rule 22 will apply.

Let us finally mention that the kernelization rules we developed for NONBLOCKER SET are rather independent of the parameter, so that they can be also applied when solving MAXIMUM NONBLOCKER SET or even MINIMUM DOMINATING SET. Hence, they can be read as important practical heuristics for MINIMUM DOMINATING SET algorithms, although we cannot expect $\mathcal{FPT}$ results for DOMINATING SET in the general case, see Chap. 9.

Corollary 4.22 *Alg. 9 provides a kernel of size at most* $5/3 \cdot k_d$ *for a given instance* (G, k_d) *of* NONBLOCKER SET, *where the problem size is measured in terms of the number of vertices.*

Let us do our computations a bit more generally, as a plug-in. Assume we have a theorem telling us that all graphs in a graph class $\mathcal{G}$ obey that the domination number $\gamma(G)$ satisfies $\gamma(G) \leq c|V(G)|$ for all $G \in \mathcal{G}$. Then, given a graph G from our class and a parameter k, we can answer YES to the question if G has a dominating set of size at most k whenever $k > c|V(G)|$. Hence, given G from our class and a parameter k_d, we can answer YES to the question if G has a nonblocker set of size at least k_d whenever $(|V(G)| - k_d) > c|V(G)|$, or if $k_d < (1-c)|V(G)|$. Otherwise, we don't know the answer, but we know that $k_d/(1-c) \geq |V(G)|$. So, we have a kernel size of $k_d/(1-c)$, measured in terms of the number of vertices of G. With the result of McCuaig and Shepherd, we have $c = 2/5$, so that the corollary follows.

Together with Lemma 4.1, we can conclude:

Corollary 4.23 *By testing all subsets of size k_d of a reduced instance (G, k_d) of* NONBLOCKER SET*, this problem can be solved in time $\mathcal{O}^*(3.0701^{k_d})$.*

A slightly improved running time analysis will be presented in Chap. 10.

From an algorithmic perspective, the fact that the domination number of a graph without isolated vertices is bounded by its edge independence number [372, Theorem 1] could be also useful to get a smaller kernel for NONBLOCKER SET; observe that the edge independence number is just the size of a maximum matching which can be computed in polynomial time.

Observe that Rule 21 may destroy planarity, since merging vertices from different parts of a graph may create a K_5 or a $K_{3,3}$ as a substructure that has not been there before. Therefore, we cannot rely anymore on Cor. 4.22 if we insist on having a planar reduced graph. However, as the reader may verify, all other rules for NONBLOCKER SET do preserve planarity. As already mentioned above, Ore proved in [314, Theorem 13.1.3] c=.5 for graphs of minimum degree one. Of course, we can deal with vertices of degree zero by a reduction rule that does not affect planarity, namely Rule 20. Hence, we can conclude:

Corollary 4.24 NONBLOCKER SET*, restricted to the planar case, admits a planar kernel graph with at most $2k_d$ vertices.*

Ore's construction is quite instructive and can be found in Alg. 10. We believe that spanning tree techniques can be used in other kernelization problems, as well. We mention one more application in the following: MAXIMUM MINIMAL VERTEX COVER. Observe that we have already treated this problem for the special case of planar graph inputs. Hence, the kernelization based on spanning trees is yielding not only smaller kernels (as we will see) but also more general results regarding the $\mathcal{FPT}$ membership of these problems.

We basically have to see how to treat isolated vertices.

Algorithm 10 A kernelization algorithm for NONBLOCKER SET, also applicable to the planar case

Input(s): an instance (G, k_d) of NONBLOCKER SET
Output(s): an equivalent instance (G', k'_d) of NONBLOCKER SET with $V(G') \subseteq V(G)$, $|V(G')| \leq 2k'_d$ and $k'_d \leq k_d$ OR YES; in the latter case, also a solution A is constructed.

```
Exhaustively apply reduction rule 20. Call the resulting instance $G$ again.
Compute a minimum spanning tree for each component of $G$.
for all component $C$ do
  Pick an arbitrary root $r_C$ of the spanning tree $T_C$ belonging to $C$.
  Collect in $L_C^i$ all vertices of $T_C$ having distance $i$ from $r_C$, where the distance is measured in $T_C$.
  Define, for $j = 0, 1$: $A_j = \bigcup_{i \geq 0} L_C^{2i+j}$.
  Choose the larger of the two and call it $A_{NB(C)}$, where ties are broken arbitrarily.
end for
Form $A = \bigcup \{A_{NB(C)} \mid C$ is component of $G\}$.
if $|A| > k_d$ then
  return YES
else
  {$G$ is the returned instance.}
end if
```

Reduction rule 24 *Let (G, k) be an instance of* MAXIMUM MINIMAL VERTEX COVER. *If C is a complete graph component of G, then reduce to $(G - C, k - (|C| - 1))$.*

The soundness of the rulecan be easily seen: A vertex cover of a complete graph C needs $|C| - 1$ vertices, and putting all vertices in the cover would render it non-minimal.

If Alg. 10 returns YES, it implicitly selects either of two disjoint vertex sets A_0 and A_1 (basically, every second level of the spanning tree). However, A_0 or A_1 need not be vertex cover sets. Notice that both A_0 and A_1 are minimal in the sense that none of them contains a closed neighborhood of any vertex. Then, we would have to supplement say the larger of the two to get a valid minimal vertex cover. Hence, answering YES in the indicated case is o.k., so that we conclude:

Corollary 4.25 MAXIMUM MINIMAL VERTEX COVER *admits a kernel graph with at most $2k$ vertices.*

Let us once more return to NONBLOCKER SET: Unfortunately, we were not able to use the stronger result provided by B. Reed [329] for the kernelization algorithm, upperbounding the domination number of a graph that has minimum degree of three. More precisely, he showed $c = 3/8$, which gives a kernel of size $1.6k_d$ for graphs of minimum degree two. This graph class appears to be a bit strange, admittedly, but we simply don't know how to get rid of all vertices of degree up to two by reduction rules.[4] There is another result due to Matheson and Tarjan [288]. We can conclude from their reasoning that graphs $G = (V, E)$ that possess a triangulated planar subgraph $G' = (V, E')$ have a dominating set with at most $\frac{|V|}{3}$ vertices. This gives a kernel of $1.5k_d$ for this graph class for NONBLOCKER SET.

Coming back to the "cheating argument" of the introductory paragraph of this section, let us reconsider NONBLOCKER SET: In a certain sense, the theorems we used to provide small kernels are in fact telling us that, whenever we assume the existence of a small nonblocker set, then the graph cannot be big. This may mean that in actual fact the parameterization we chose is not appropriate. Similar to the suggestion in [307], this might imply we should *reparameterize.* One such reparameterization was already discussed in [284] for a logical problem. Transferred to NONBLOCKER SET, we might ask questions like:

Problem name: δ-NONBLOCKER SET (δ-NB)
Given: A graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq \delta|V| + k_d$?

Here, $\delta \in (0, 1)$ is a constant that steers the reparameterization. It is not part of the input, but rather essential for creating a whole family of parameterized problems.

Theorem 4.26 *If* $\delta < 3/5$, *then* δ-NONBLOCKER SET *is in* $\mathcal{FPT}$.

Proof. We are going to show that δ-NB has a problem kernel, see Theorem 2.4.

Let G be a graph instance and k_d the parameter. We exhaustively apply the same reduction rules as developed before for NONBLOCKER SET to get a reduced graph G' with parameter k'_d. $G' = (V, E)$ has minimum degree of

[4]It might be worthwhile noting that there is a kind of complementary result for graphs with *maximum* degree three by Fisher, Fraughnaugh and Seager [182]; however, the corresponding bound also includes the number of vertices in the graph, so that the kernel size bound looks a bit strange; for cubic graphs, however, the result of Reed is exactly matched.

two, and therefore it has a minimum dominating set of size at most $2/5 \cdot |V|$. Hence, $|V| - |N| \leq 2/5 \cdot |V|$ for a maximum nonblocker set N, i.e., $|N| \geq 3/5 \cdot |V|$. If we ask for an N that is smaller than this, we can safely answer YES, see Alg. 9. Hence, given the question if $|N| \geq \delta|V| + k_d$, we can answer YES if $\delta|V| + k_d \leq 3/5 \cdot |V|$, i.e., if $k_d \leq (3/5 - \delta)|V|$. Otherwise, we have arrived at a kernel of size $|V| \leq (3/5 - \delta)^{-1}k_d$. ∎

Observe that similar reparameterizations can be performed (while keeping $\mathcal{FPT}$-membership) for problems like VERTEX INDUCED FOREST, PLANAR INDEPENDENT SET, etc.

4.4.3 Dominating Queens

Finally, let us reconsider the following variant of a domination problem already introduced in the introduction:

Problem name: DOMINATING SET OF QUEENS (QDS)
Given: An $n \times n$ chessboard C
Parameter: a positive integer k
Output: Is it possible to place k queens on C such that all squares are dominated ?

Theorem 4.27 QDS *has a problem kernel of size* $(2k + 1)^2$.

Proof. Indeed, we are going to verify the following "cheating" reduction rule:

Reduction rule 25 *If C is an $n \times n$ board and k a parameter with $n > 2k + 1$, then* NO.

A proof of this result is shown in [100, Theorem 3]; however, that proof contains a non-trivial "hole," and therefore we produce a proof in what follows.[5]

Given an $n \times n$ board C, fix a dominating set of k queens, where k is minimal. The rule is surely o.k. for $n = 1, 2$, since there is no fitting parameter value. If $n > 2$, we can show $k \leq n-2$, since placing $n-2$ queens on the main diagonal of the chessboard, leaving free the first and last square in that diagonal, will dominate the whole chessboard. Therefore, the following rows and columns exist: Let column a (b, resp.) be the leftmost (rightmost, respectively) column that is not occupied by any queen in our minimum dominating

[5]Thanks to my class in approximation algorithms (Tübingen in 2004/2005), in particular to Bernd Lutz, who pointed this hole to me.

set. Correspondingly, let row c (d, resp.) be the topmost (bottom-most, respectively) unoccupied row. Let $\delta_1 = b - a$ and $\delta_2 = d - c$. By symmetry of the argument, we can assume that $\delta_1 \geq \delta_2$.

Let S_a (S_b, resp.) be the set of squares in column a (b, resp.) that lie between rows c and $d' := \min\{n, c + \delta_1\}$ inclusive. Set $S := S_a \cup S_b$.

Claim 0: Any queen can diagonally dominate at most two squares of S.
Proof of Claim. Since $\delta_1 \geq \delta_2$, no diagonal intersects both S_a and S_b. Hence, every queen can dominate at most one square of S on a southwest-northeast diagonal and one square of S on a northwest-southeast diagonal. $\diamondsuit$

Claim 1: Any queen placed in the first $c - 1$ rows or in the last $n - d'$ rows can only dominate at most two squares of S.
Proof of Claim. By definition, no queen that is placed on the mentioned squares can dominate any squares from S by row or column. The preceding claim hence proves Claim 1. $\diamondsuit$

Claim 2: Any queen can only dominate at most four squares of S.
Proof of Claim. Due to the preceding claim, we consider a queen that is placed on the rows c through d' (inclusive). By definition of S, such a queen cannot dominate any squares of S by column. It can obviously dominate at most two squares of S by row. Hence, Claim 0 finishes this proof. $\diamondsuit$

Let o be the number of "occupied" rows to which Claim 1 applies, i.e., $o = c - 1 + n - d'$. Summarizing our claims, we can state:

- At least o queens dominate at most two squares of S.
- At most $k - o$ queens dominate at most four squares of S.

Hence, the k queens together dominate at most $2o + 4(k - o) = 4k - 2o$ squares of S.

We now distinguish two subcases:

- If $o \geq c$ (or, equivalently, if $c + \delta_1 \leq n$, i.e., $d' = c + \delta_1$), then $|S| = 2\delta_1$, since δ_1 is the number of rows that is pertaining to S. Moreover, Claim 1 applies to the remaining $o = n - (\delta_1 + 1)$ many rows. This gives the following inequality, since we are discussing how to dominate all squares in S:
$$4k - 2n + 2\delta_1 + 2 \geq 2\delta_1 \quad \rightsquigarrow 2k + 1 \geq n.$$
- If $o = c - 1$ (or, equivalently, if $c + \delta_1 > n$, i.e., $d' = n$), then $|S| = 2(n - (c - 1)) = 2n - 2o$. Hence,
$$4k - 2o \geq 2n - 2o \quad \rightsquigarrow 2k \geq n.$$

Since we don't know which of the two cases applies, we have to take the less stricter bound in our theorem. ∎

So, on the one hand, this shows membership of DOMINATING SET OF QUEENS in $\mathcal{FPT}$, but on the other hand, it is also sort of questionable if the parameterization is appropriate.

4.5 VERTEX COVER: our old friend again

The reason for having an own section on VERTEX COVER in the context of kernelization are manifold:

- In the introduction, we already saw how to design data reduction rules that provide a quadratic kernel for VERTEX COVER.
- There exist by now *two* (at first glance) completely different ways of obtaining a small linear kernel for VERTEX COVER, one being based on nice structural properties of vertex covers, as discovered by Nemhauser and Trotter [304]; the other one being based on generalizing Rule 1 for isolated vertices to a so-called *crown rule* [110, 164]. Having two ways to obtain the same result is in fact an interesting situation on its own. Computer experiments indicate, however, that the kernelization based on crown reduction rules is better, not only in terms of running time (see [3]).
- Finally, we will discuss a generalization of VERTEX COVER and see how we can build up a set of data reduction rules that can be read either as a reduction to the formerly treated case of VERTEX COVER or as enabling to read reduction rules for VERTEX COVER as being valid for the more general case, as well, and hence enabling us to state a small kernel result for this case, too.

As indicated above, the first part of this section will be devoted to discussing the following result:

Theorem 4.28 VERTEX COVER *has a kernel of size upperbounded by* $2k$.

Remark 4.29 *Can we possibly hope for a smaller kernel? The difficulty of getting a smaller kernel for* VERTEX COVER *seems to be related with the (old) problem of getting approximation algorithms that provide a quality ratio essentially better than two. More precisely, the best known approximation [255] gives a ratio of*

$$2 - \Theta(\frac{1}{\sqrt{\log n}}).$$

More precisely, although not formally required by the definition of kernelization, it seems likely that the development of a kernelization algorithm that gives a kernel of say $1.7k$ *would also mean that we get a* 1.7 *approximation for* VERTEX COVER, *by simply taking all kernel vertices (plus possibly some that are provided by the reduction itself) into the (approximate) cover.*

Similar comments also apply to the VERTEX COVER *problem on hypergraphs, likewise known as* d-HITTING SET, *where the trivial* d*-approximation algorithm is hard to beat. Regarding positive approximability results in some particular cases, we refer to the recent papers of Halperin and Okun [226, 313]. However, in that case no linear kernel is known, see [310], which in itself can be regarded as an interesting research topic.*

4.5.1 The result of Nemhauser and Trotter

Let us first state the theorem of Nemhauser and Trotter according to the formulation given in [35] and in [320]; the original formulation of Nemhauser and Trotter [304] uses connections with Integer Linear Programming that are not central to this Habilitationsschrift. A concise proof of the result is contained in [35].

If $G = (V, E)$ is a graph, let $G_{BP} = (V \times \{1, 2\}, E_{BP})$ be its *bipartite graph variant*, where

$$E_{BP} = \{\{(u,1),(v,2)\} \mid \{u,v\} \in E\}\,.$$

Given a vertex set C_{BP} of G_{BP}, let

$$C_{BP}^{\text{AND}}(G) = \{v \in V \mid (v,1) \in C \wedge (v,2) \in C\}\,,$$

$$C_{BP}^{\text{XOR}}(G) = \left\{v \in V \setminus C_{BP}^{\text{AND}}(G) \mid (v,1) \in C \vee (v,2) \in C\right\}.$$

One point of Alg. 11 needs further clarification: how can we efficiently compute a vertex cover in bipartite graphs? The key to this lies in the following theorem, due to König and Egerváry (see [264]):

Theorem 4.30 *If* G *is a bipartite graph, then the size of a minimum vertex cover of* G *equals the size of a maximum matching of* G.

It is worth noting that one direction of that theorem is close to trivial; let us state it in a weakened form:

Lemma 4.31 *If* M *is a matching in a graph* G*, then* G *has no vertex cover of size less than* $|M|$.

Algorithm 11 A kernelization algorithm for VERTEX COVER, called VCNT

Input(s): an instance (G, k) of VERTEX COVER

Output(s): an equivalent instance (G', k') of VERTEX COVER with $V(G') \subseteq V(G)$, $|V(G')| \leq 2k'$ and $k' \leq k$, as well as a set $C \subseteq V(G)$ that must appear in any vertex cover for G (and that is not accounted into the budget k').

```
    Create $G_{BP}$ from $G$.
    Compute a minimum vertex cover $C_{BP}$ of $G_{BP}$.
    Compute $C := C_{BP}^{\text{AND}}(G)$ and $C_{BP}^{\text{XOR}}(G)$.
    Let $k' := k - |C|$.
 5: if $k' < 0$ then
      Pick an edge $e = \{u, v\}$ from $G$.
      Define $G' = (\{u, v\}, \{e\})$ and $k' = 0$.
      Let $C := \emptyset$.
    else
10:   Let $G' = G[C_{BP}^{\text{XOR}}(G)]$.
    end if
    return $(G', k')$ as well as $C$.
```

Proof. All edges of M need to be covered. Each edge can be covered by either of its endpoints. Covering $e \in M$ this way will not cover any other edge from M, since M is a matching. Hence, we need $|M|$ vertices to cover M, so that we need (at least) $|M|$ vertices to cover G. ∎

From an algorithmic point of view, two things are worth mentioning:

- A maximum matching $M \subseteq E$ of a graph $G = (V, E)$ can be computed in time $\mathcal{O}(|E|\sqrt{|V|}$; such an algorithm is usually based on the notion of *alternating paths* and can be found in any introductory standard text on algorithms.

- From M, one can efficiently construct a vertex cover $C \subseteq V$ if G is bipartite. Such a construction can be based on an alternating paths construction, as they are already used in the standard construction showing that maximum matchings can be found in polynomial time, see [Chapter 11, Site 13]. An alternative way using Dinic's algorithm is explained in [3].

Nemhauser and Trotter then showed the following properties:

Theorem 4.32 *Let $G = (V, E)$ be a graph.*

1. *Let C' be a cover of $G[C_{BP}^{XOR}(G)]$. Then $C := C' \cup C_{BP}^{AND}(G)$ is a cover of G.*
2. *There is a minimum cover $C^*(G)$ of G that contains $C_{BP}^{AND}(G)$.*
3. *$|C_{BP}^{XOR}(G)| \leq 2|C^*(G)|$, where $C^*(G)$ is a minimum cover of G.*

From the mentioned results, the following was observed in [86, 251]:

Corollary 4.33 *To each* VERTEX COVER *instance (G,k), Alg. 11 computes a kernel (G',k') with $|V(G')| \leq 2k'$ and $k' \leq k$ in time $\mathcal{O}\left(|E(G)|\sqrt{|V(G)|}\right)$.*

Remark 4.34 *Hochbaum [233] used the result of Nemhauser and Trotter to give a factor-2 approximation algorithm for* MINIMUM VERTEX COVER*: this algorithm would output*

$$C_{BP}^{OR}(G) = C_{BP}^{AND}(G) \cup C_{BP}^{XOR}(G)$$

for an input graph G (using the abbreviation introduced above).

Remark 4.35 *By first kernelizing according to Buss' rule (see Chap. 2) and then according to Alg. 11 (in fact, all these rules could and should be applied to exhaustion before starting the search-tree part as explained in the next chapter), the overall complexity for the kernelization phase could be reduced to $\mathcal{O}(nk + k^3)$.*

Remark 4.36 *A similar algorithm works for the kernelization of the weighted version of* VERTEX COVER*, i.e., of* WEIGHTED VERTEX COVER*. Then, however, the computation of the kernel would be based on maximum flow computations, which are slower than the maximum matching computations in the unweighted case.*

When the input graph is planar, the kernelization computations could be considerably sped up. All these complexities are detailed in [35].

We also mention the following properties of a decomposition of the vertex set V of a graph $G = (V,E)$ according to Theorem 4.32 that are lists in [320, Sec. 5.1]:

1. $C_{BP}^{\mathrm{NOR}}(G) = V \setminus (C_{BP}^{\mathrm{XOR}}(G) \cup C_{BP}^{\mathrm{AND}}(G))$ forms an independent set in G.
2. All neighbors of $C_{BP}^{\mathrm{NOR}}(G)$ are in $C_{BP}^{\mathrm{AND}}(G)$.

Remark 4.37 *In [96, 95], the theorem of Nemhauser & Trotter got sharpened so that,* for all minimum vertex covers *of G, they are contained in $C_{BP}^{AND}(G)$.*

It is finally worthwhile noticing that the kernelization due to the Nemhauser-Trotter theorem can be called *global* in contrast to all other kernelization procedures we have seen up to now; this is also the reason for its comparatively huge complexity.

4.5.2 Crown reductions: another way to a small kernel

We follow the definition of a crown as given in [95] for the more general weighted case. Otherwise, this section is mostly based on [3, 110, 247].

Definition 4.38 Let $G = (V, E)$ be a graph. An independent set I in G is called a *crown* if, for all nonempty sets $U \subseteq N(I)$, $|U| \leq |N(U) \cap I|$.

That definition is equivalent to the one given in [3, 110, 164] due to Hall's marriage theorem (apart from the requirement that $I \neq \emptyset$ according to those definitions) that can be found in any textbook on graph theory. Hence, a crown I has the following property:

- $H := N(I)$ is matched into I. (H is usually referred to as the *head* of the crown.)

In other words, the bipartite graph with vertex set $H \cup I$ and edge set $\{uv \mid u \in Hv \in I\}$ has a *upper perfect matching.*

Observe that a decomposition á la Nemhauser and Trotter is "nearly" a crown (more precisely, the set $C_{BP}^{\text{NOR}}(G)$).

M. Chlebík and J. Chlebíková [95] have shown that there is always a specific Nemhauser-Trotter decomposition that is actually a crown.

Fellows has introduced the following reduction rule:

Reduction rule 26 *If* $(G = (V, E), k)$ *a* VERTEX COVER *instance and* $I \subseteq V$ *a crown with head* $H = N(I)$*, then reduce to* $(G - (I \cup H), k - |H|)$*.*

The question is then how to find crowns. Juedes[6] devised Alg. 12 to this purpose. How can we use this algorithm to get a small kernel for VERTEX COVER? In fact, the kernelization algorithm has several ways to use the information collected in Alg. 12; therefore, we state the corresponding kernelization algorithm in an explicit form in Alg. 13.

Lemma 4.39 *Alg. 13 combined with Rule 26, when run on input* (G, k)*, either correctly answers* NO *or yields a reduced instance* (G', k') *of* VERTEX COVER *such that* $|V(G')| \leq 3k$*.*

[6]personal communication in Oct. 2002; this idea can be found in [3, 247]

Algorithm 12 An efficient algorithm to find a crown in a graph, called GETCROWN

Input(s): an graph G
Output(s): a crown $I \subseteq V(G)$

```
Greedily find a maximal matching $M_1$ of $G$.
Let $O$ (outsiders) be the set of vertices from $G$ that are not in the matching.
if $O \neq \emptyset$ then
  Let $I' := \emptyset$.
  Find a maximum matching $M_2$ in the bipartite graph $B$ with vertex set
  $O \cup N(O)$ and edge set $\{uv \mid u \in O \land v \in N(O)\}$.
  Let $I$ collect all vertices in $O$ that are not matched by $M_2$.
  while $I' \neq I$ do
    Let $I' := I$.
    Let $H := N(I)$.
    Let $I := I \cup \{u \in O \mid \exists v \in H(uv \in M_2)\}$.
  end while
else
  $I := \emptyset$.
end if
```

The proof of this lemma is based on the following observations:

- If we find a matching M (i.e., M_1 or M_2) in G such that $|M| > k$, then (G, k) is a NO-instance due to Lemma 4.31.

- If M_1 and M_2 contain no more than k edges, the graph induced by $M_1 \cup M_2$ has at most $3k$ vertices. Has, after applying Rule 26 following the execution of Alg. 13, the remaining graph has at most $3k$ vertices.

Abu-Khzwam *et al.* have demonstrated [3] that a kernelization based on Alg. 13 runs much faster than directly using Alg. 11. This is probably due to the fact that the Nemhauser-Trotter reduction has to compute a maximum matching in a relatively large graph, while the crown-based reduction only computes a maximal matching in a large graph and then maximum matchings in much smaller graphs. More precisely, it is shown in [247] that this algorithm runs in time $\mathcal{O}(|V|+|E|)$ on an input graph $G = (V, E)$ by looking again into how maximum matching algorithms work in our case.

However, it is clear that one could certainly afford running Alg. 11 after having obtained a $3k$ kernel with Alg. 13. The experiments reported in [3] also indicate that this might not actually lead to a smaller kernel (as could be expected due to theory).

Algorithm 13 Using crowns for kernels in the case of VERTEX COVER; the algorithm is called kernelfromcrown

Input(s): an instance (G, k) of VERTEX COVER
Output(s): NO if the VERTEX COVER instance has no solution OR a crown $I \subseteq V(G)$ such that $|V(G - N[I])| \leq 3k$

```
Greedily find a maximal matching $M_1$ of $G$.
if $|M_1| > k$ then
  return NO
else
  Let $O$ (outsiders) be the set of vertices from $G$ that are not in $M_1$.
  if $O \neq \emptyset$ then
    Find a maximum matching $M_2$ in the bipartite graph $B$ with vertex
    set $O \cup N(O)$ and edge set $\{uv \mid u \in O \wedge v \in N(O)\}$.
    if $|M_2| > k$ then
      return NO
    else
      Let $I' := \emptyset$.
      Let $I$ collect all vertices in $O$ that are not matched by $M_2$.
      while $I' \neq I$ do
        Let $I' := I$.
        Let $H := N(I)$.
        Let $I := I \cup \{u \in O \mid \exists v \in H(uv \in M_2)\}$.
      end while
    end if
  else
    $I := \emptyset$.
  end if
  return $I$
end if
```

There is alternative way of obtaining a kernel of size $2k$, known as *iterative compression*, see [110]. However, getting the kernel that small comes at a price: one has to optimally solve k (smaller) instances of VERTEX COVER, so that the overall kernelization does not need polynomial time anymore (as required by the kernelization definition) but rather in $\mathcal{FPT}$-time. We therefore omit details here.

The idea of crown reductions have been successfully applied to other circumstances, as well. We only mention the following papers: [110, 166, 247, 289, 324, 325].

4.5.3 A generalization of VERTEX COVER

Let us now consider the following generalization of VC:

> **Problem name:** GENERALIZED VERTEX COVER (VCGEN)
> **Given:** A graph $G = (V, E)$, a subset V' of vertices
> **Parameter:** a positive integer k
> **Output:** Is there a *vertex cover* $C \subseteq V'$ with $|C| \leq k$?

It is clear that this is actually a generalization of VC, since it is possible to have $V' = V$. Therefore, $\mathcal{NP}$-hardness of VCGEN is trivially concluded.

We propose the following reduction rules for this problem. In these rules, let $(G = (V, E), V', k)$ be an instance of VCGEN.

Reduction rule 27 *If there is an edge in $G[V \setminus V']$, then* NO.

Reduction rule 28 *If there is an edge $\{u, v\}$ with $u \in V'$ and $v \in V \setminus V'$, then put u into the cover, i.e., reduce to $(G - u, V', k - 1)$.*

The soundness of both rules is clear, since we are still aiming at a cover. Hence, edges between vertices from $V \setminus V'$ cannot be covered at all (leading to a NO-instance), and edges that contain exactly one vertex u from V' can only be covered by taking that vertex u into the cover.

After having exhaustively applied these two reduction rules, in the case that we did not already discover that the given instance was not solvable at all, we are left with an instance that contains only vertices from $V \setminus V'$ that are isolates. Since such isolates would not be put into a cover even if they were belonging to V', we can conclude:

Lemma 4.40 *If $(G = (V, E), V', k)$ is an instance of* GENERALIZED VERTEX COVER *that is reduced with respect to Rules 27 and 28. Then, (G, V', k) is a* YES*-instance of* VCGEN *iff (G, V, k) is a* YES*-instance of* VCGEN *iff (G, k) is a* YES*-instance of* VC.

This means that we have reduced VCGEN to VC. In view of the trivial reduction for the converse direction, we may conclude:

Corollary 4.41 GENERALIZED VERTEX COVER *and* VERTEX COVER *are parameterized interreducible.*

However, we can also interpret this reducibility as providing a complete set of data reduction rules for GENERALIZED VERTEX COVER that have to be applied in a certain order to get an overall kernelization algorithm that

Algorithm 14 A kernelization algorithm for GENERALIZED VERTEX COVER, called VCgen-kern

Input(s): an instance (G, V', k) of GENERALIZED VERTEX COVER
Output(s): YES iff the given instance has a solution

```
if Reduction rule $R$ out of rules 27 and 28 is applicable then
  return VCgen-kern($R(G, V', k)$)
else if $V(G) \neq V'$ then
  {This can only happen if $G[V(G) \setminus V']$ contains no edges.}
  return VCgen-kern($G, V(G), k$)
else
  {Apply your favorite VERTEX COVER-reduction rules to get a $2k$-kernel,
  interpreted as rules for GENERALIZED VERTEX COVER.}
end if
```

deviates a bit from the usual scheme (see Alg. 5) for kernelization algorithms, so that we explicitly state it in Alg. 4.5.3. The indicated interpretation of VC-rules as VCGEN-rules is very straightforward: a VC instance (G, k) is simply understood as VCGEN instance $(G, V(G), k)$. Since (according to the case distinction in Alg. 4.5.3), we are only left with VCGEN instances (G, V', k) such that $V' = V(G)$, this translation gives a sound reduction rule for GENERALIZED VERTEX COVER. Hence, we can conclude:

Theorem 4.42 GENERALIZED VERTEX COVER *has a* $2k$ *problem kernel.*

4.6 PLANAR DOMINATING SET: a tricky example

In the following, we present results on PLANAR DOMINATING SET; this is an example of quite complex kernelization rules derived in [12, 82].

Let us first state the problem.

Problem name: PLANAR DOMINATING SET (PDS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *dominating set* $D \subseteq V$ with $|D| \leq k$?

For a graph G, we denote by $\gamma(G)$ the size of a minimum dominating set in G. We will color the vertices of the graph G with two colors: black and white. Initially, all vertices are colored black. Informally speaking, white vertices will be those vertices that we know for sure when we color them that there exists a minimum dominating set for the graph *excluding* all of them. The black vertices are all other vertices.[7]

For a vertex v in G, we partition its set of neighbors $N(v)$ into three sets:

1. $N_1(v) = \{u \in N(v) \mid N(u) \setminus N[v] \neq \emptyset\}$;
2. $N_2(v) = \{u \in N(v) \setminus N_1(v) \mid N(u) \cap N_1(v) \neq \emptyset\}$; and
3. $N_3(v) = N(v) \setminus (N_1(v) \cup N_2(v))$.

To understand these notions, have a look at the following example:

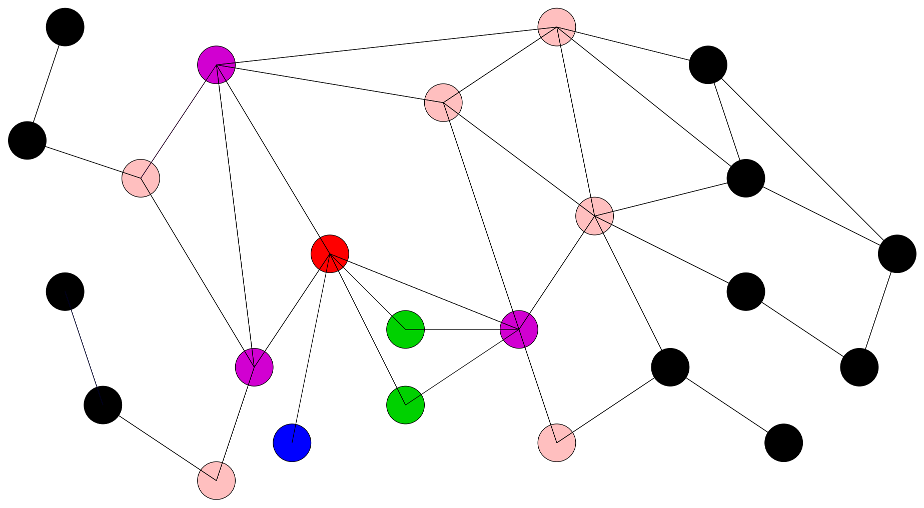


Figure 4.2: Explaining the division of the neighborhood of v.

Example 4.43 Have a look at the red vertex v in Fig. 4.2. Its neighborhood is colored in three different colors. For the convenience of the reader, we repeat the definitions of the various divisions of the neighborhood in the following, using matching colorings with respect to the example graph.
$N_1(v) = \{u \in N(v) \mid N(u) \setminus N[v] \neq \emptyset\}$
$N_2(v) = \{u \in N(v) \setminus N_1(v) \mid N(u) \cap N_1(v) \neq \emptyset\}$
$N_3(v) = N(v) \setminus (N_1(v) \cup N_2(v))$

For two vertices v and w we define $N(v,w) = N(v) \cup N(w)$ and $N[v,w] = N[v] \cup N[w]$. Similarly, we partition $N(v,w)$ into three sets:

[7]In Chap. 5, we will present a search tree algorithm for PLANAR DOMINATING SET that is also using a black and white graph: note that the colors there have different semantics from the usage here.

1. $N_1(v,w) = \{u \in N(v,w) \mid N(u) \setminus N[v,w] \neq \emptyset\}$;

2. $N_2(v,w) = \{u \in N(v,w) \setminus N_1(v,w) \mid N(u) \cap N_1(v,w) \neq \emptyset\}$; and

3. $N_3(v,w) = N(v,w) \setminus (N_1(v,w) \cup N_2(v,w))$.

Definition 4.44 Let $G = (V, E)$ be a plane graph. A *region* $R(v,w)$ between two vertices v and w is a closed subset of the plane with the following properties:

1. The boundary of $R(v,w)$ is formed by two simple paths P_1 and P_2 in V which connect v and w, and the length of each path is at most three.

2. All vertices that are strictly inside (i.e., not on the boundary) the region $R(v,w)$ are from $N(v,w)$.

For a region $R = R(v,w)$, let $V[R]$ denote the vertices in R, i.e.,

$$V[R] := \{u \in V \mid u \text{ sits inside or on the boundary of } R\}.$$

Let $V(R) = V[R] - \{v,w\}$.

Definition 4.45 A region $R = R(v,w)$ between two vertices v and w is called a *simple region* if all vertices in $V(R)$ are common neighbors of both v and w, that is, $V(R) \subseteq N(v) \cap N(w)$.

We introduce the following definitions.

Definition 4.46 A region $R = R(v,w)$ between two vertices v and w is called a *quasi-simple region* if $V[R] = V[R'] \cup R^+$, where $R' = R'(v,w)$ is a simple region between v and w, and R^+ is a set of white vertices satisfying the following conditions:

1. Every vertex of R^+ sits strictly inside R'.

2. Every vertex of R^+ is connected to v and not connected to w, and is also connected to at least one vertex on the boundary of R' other than v.

A vertex in $V(R)$ is called a *simple vertex*, if it is connected to both v and w, otherwise it is called a *non-simple vertex*. The set of vertices R^+, which consists of the non-simple vertices in $V(R)$, will be referred to as $R^+(v,w)$.

For a vertex $u \in V$, denote by $B(u)$ the set of black vertices in $N(u)$, and by $W(u)$ the set of white vertices in $N(u)$. We describe next the reduction and coloring rules to be applied to the graph G. The reduction and coloring rules are applied to the graph in the order listed below until the application of any of them does not change the structure of the graph nor the color of any vertex in the graph. Observe that—in contrast with many kernelization scenarios we find in this chapter—none of the eight reduction rules will change the value of the parameter. Consequently, we omit the parameter when describing the reduction rules.

Reduction rule 29 *If $N_3(v) \neq \emptyset$ for some black vertex v, then remove the vertices in $N_2(v) \cup N_3(v)$ from G, and add a new white vertex v' and an edge (v, v') to G.*

Let us visualize at least this simple first rule by our example.

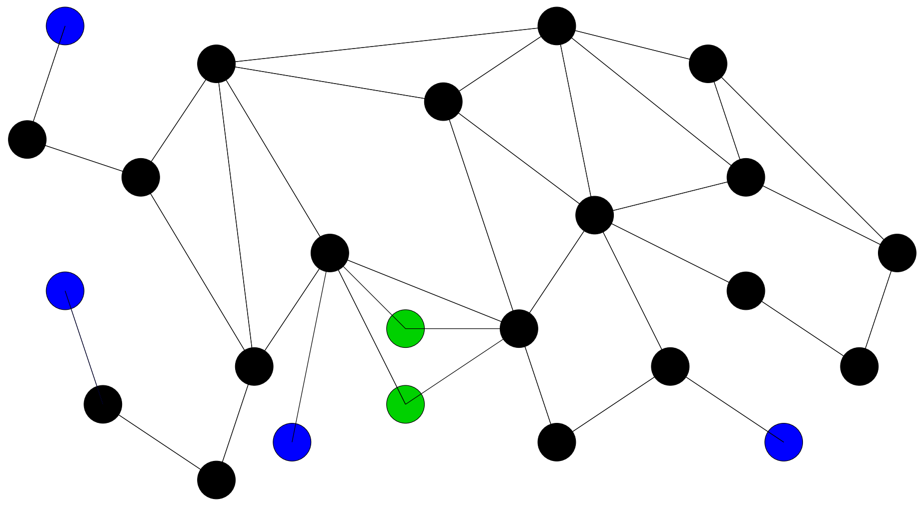


Figure 4.3: Where Rule 29 can be applied.

Example 4.47 Let us consider again the graph from Ex. 4.43, assuming that all vertices are (initially) black. Then, Fig. 4.3 shows where Rule 29 can be applied. This means, all the vertices colored blue belong to $N_3(v)$ for some v. Hence, some black vertices will be replaced by white vertices (and some of the vertices will even disappear) according to Rule 29. The result of applying that rule is shown in Fig. 4.4.

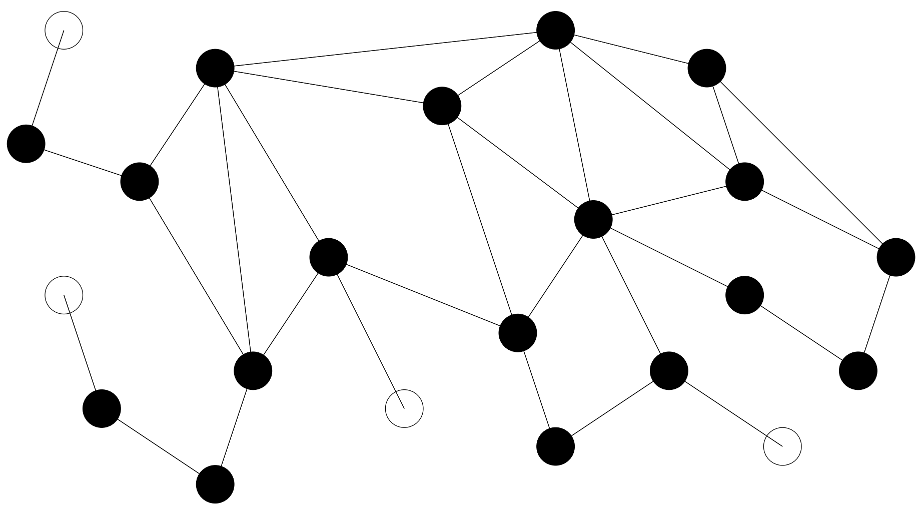


Figure 4.4: Some vertices are white now.

Reduction rule 30 *If $N_3(v, w) \neq \emptyset$ for two black vertices v, w, and if $N_3(v, w)$ cannot be dominated by a single vertex in $N_2(v, w) \cup N_3(v, w)$, then we distinguish the following two cases.*

Case 1. *If $N_3(v, w)$ can be dominated by a single vertex in $\{v, w\}$ then:*

(1.1) *if $N_3(v, w) \subseteq N(v)$ and $N_3(v, w) \subseteq N(w)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(v) \cap N(w)$ from G and add two new white vertices z, z' and the edges $(v, z), (w, z), (v, z'), (w, z')$ to G;*

(1.2) *if $N_3(v, w) \subseteq N(v)$ and $N_3(v, w) \not\subseteq N(w)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(v)$ from G and add a new white vertex v' and the edge (v, v') to G; and*

(1.3) *if $N_3(v, w) \subseteq N(w)$ and $N_3(v, w) \not\subseteq N(v)$, remove $N_3(v, w)$ and $N_2(v, w) \cap N(w)$ from G and add a new white vertex w' and the edge (w, w') to G.*

Case 2. *If $N_3(v, w)$ cannot be dominated by a single vertex in $\{v, w\}$, then remove $N_2(v, w) \cup N_3(v, w)$ from G and add two new white vertices v', w' and the edges (v, v'), (w, w') to G.*

Reduction rule 31 *For each black vertex v in G, if there exists a black vertex $x \in N_2(v) \cup N_3(v)$, color x white, and remove the edges between x and all other white vertices in G.*

Reduction rule 32 *For every two black vertices v and w, if $N_3(v, w) \neq \emptyset$, then for every black vertex $x \in N_2(v, w) \cup N_3(v, w)$ that does not dominate all vertices in $N_3(v, w)$, color x white and remove all the edges between x and the other white vertices in G.*

Reduction rule 33 *For every quasi-simple region $R = R(v, w)$ between two vertices v and w, if v is black, then for every black vertex $x \in N_2(v, w) \cup N_3(v, w)$ strictly inside R that does not dominate all vertices in $N_2(v, w) \cup N_3(v, w)$ strictly inside R, color x white and remove all the edges between x and the other white vertices in G.*

Reduction rule 34 *For every two white vertices u and v, if $N(u) \subseteq N(v)$, and $u \in N_2(w) \cup N_3(w)$ for some black vertex w, then remove v.*

Reduction rule 35 *For every black vertex v, if every vertex $u \in W(v)$ is connected to all the vertices in $B(v)$, then remove all the vertices in $W(v)$ from G.*

Reduction rule 36 *For every two black vertices v and w, let $W(v, w) = W(v) \cap W(w)$. If $|W(v, w)| \geq 2$ and there is a degree-2 vertex $u \in W(v, w)$, then remove all vertices in $W(v, w)$ except u, add a new degree-2 white vertex u', and connect u' to both v and w.*

Theorem 4.48 *Let G be a graph with n vertices. Then in time $\mathcal{O}(n^3)$ we can construct a graph G' from G such that:*

1. *G' is reduced,*
2. *$\gamma(G') = \gamma(G)$,*
3. *there exists a minimum dominating set for G' that excludes all white vertices of G', and*
4. *from a minimum dominating set for G', a minimum dominating set for G can be constructed in linear time.*

Given any dominating set D in a graph G, a *D-region decomposition* of G is a set $\mathfrak{R}$ of regions between pairs of vertices in D such that:

1. For any region $R = R(v, w)$ in $\mathfrak{R}$, no vertex in D is in $V(R)$. That is, a vertex in D can only be an endpoint of a region in $\mathfrak{R}$.
2. No two distinct regions $R_1, R_2 \in \mathfrak{R}$ intersect. However, they may touch each other by having common boundaries.

Note that all the endpoints of the regions in a D-region decomposition are vertices in D. For a D-region decomposition $\Re$, define $V[\Re] = \bigcup_{R \in \Re} V[R]$. A D-region decomposition is *maximal*, if there is no region R such that $\Re' = \Re \cup R$ is a D-region decomposition with $V[\Re] \subsetneq V[\Re']$.

For a D-region decomposition $\Re$, associate a planar graph $G_{\Re}(V_{\Re}, E_{\Re})$ with possible multiple edges, where $V_{\Re} = D$, and such that there is an edge between two vertices v and w in $G_{\Re}$ if and only if $R(v, w)$ is a region in $\Re$. A planar graph (possibly with multiple edges) is called a *thin planar graph* if there is a planar embedding of the graph such that for any two edges e_1 and e_2 between two distinct vertices v and w in the graph, there must exist two more vertices which sit inside the disjoint areas of the plane enclosed by e_1 and e_2.

Alber, Fellows and Niedermeier [12] showed that the number of edges in a thin planar graph of n vertices is bounded by $3n - 6$. They also showed that for any reduced plane graph G and a dominating set D of G, there exists a maximal D-region decomposition for G such that $G_{\Re}$ is thin. Since the maximal D-region decomposition in [12] starts with any dominating set D and is not affected by the color a vertex can have, the same results hold true for our reduced graph G whose vertices are colored black/white, and with a minimum dominating set D consisting only of black vertices. Our discussion is summarized in the following proposition.

Proposition 4.49 *Let G be a reduced graph and D a dominating set of G consisting of black vertices. Then there exists a maximal D-region decomposition $\Re$ of G such that $G_{\Re}$ is thin.*

Corollary 4.50 *Let G be a reduced graph with a minimum dominating set D consisting of k black vertices, and let $\Re$ be a maximal D-region decomposition of G such that $G_{\Re}$ is thin. Then the number of regions in $\Re$ is bounded by $3k - 6$.*

In the remainder of this discussion, $\Re$ will denote a maximal D-region decomposition of G such that $G_{\Re}$ is thin. Let u and v be two vertices in G. We say that u and v are *boundary-adjacent vertices* if (u, v) is an edge on the boundary of some region $R \in \Re$. For a vertex $v \in G$, denote by $N^*(v)$ the set of vertices that are boundary-adjacent to v. Note that for a vertex $v \in D$, since v is black, by Rule 31, all vertices in $N_2(v) \cup N_3(v)$ must be white.

Proposition 4.51 *Let $v \in D$. The following are true.*

(a) Every vertex $u \in N_1(v)$ is in $V[\Re]$.

(b) The vertex v is an endpoint of a region $R \in \Re$. That is, there exists a region $R = R(x, y) \in \Re$ such that $v = x$ or $v = y$.

(c) Every vertex $u \in N_2(v)$ which is not in $V[\Re]$ is connected only to v and to vertices in $N^*(v)$.

Let x be a vertex in G such that $x \notin V[\Re]$. Then by part (b) in Proposition 4.51, $x \notin D$. Thus, $x \in N(v)$ for some black vertex $v \in D \subseteq V[\Re]$. By part (a) in Proposition 4.51, $x \notin N_1(v)$, and hence, $x \in N_2(v) \cup N_3(v)$. By Rule 31, the color of x must be white. Let $R = R(v, w)$ be a region in $V[\Re]$ of which v is an endpoint (such a region must exist by part (b) of Proposition 4.51). We distinguish two cases.

Case A. $x \in N_3(v)$. Since v is black, by Rule 29, this is only possible if $\deg(x) = 1$ and $N_2(v) = \emptyset$ (in this case x will be the white vertex added by the rule). In such case it can be easily seen that we can flip x and place it inside R without affecting the planarity of the graph.

Case B. $x \in N_2(v)$. Note that in this case $N_3(v) = \emptyset$, and x is only connected to v and $N^*(v)$ by part (c) in Proposition 4.51. If $\deg(x) = 2$, by a similar argument to **Case A** above, x can be flipped and placed inside R.

According to the above discussion, it follows that the vertices in G can be classified into two categories: (1) those vertices that are in $V[\Re]$; and (2) those that are not in $V[\Re]$, which are those vertices of degree larger than two that belong to $N_2(v)$ for some vertex $v \in D$, and in this case must be connected only to vertices in $N^*(v)$. To bound the number of vertices in G we need to bound the number of vertices in the two categories. We start with the vertices in category (2).

Let O denote the set of vertices in category (2). Note that all vertices in O are white, and no two vertices u and v in O are such that $N(u) \subseteq N(v)$. To see why the latter statement is true, note that every vertex in O must be in $N_2(w)$ for some black vertex $w \in D$. So if $N(u) \subseteq N(v)$, then by Rule 34, v would have been removed from the graph. To bound the number of vertices in O, we will bound the number of vertices in O that are in $N_2(v)$ where $v \in D$. Let us denote this set by $N^\dagger(v)$. Let $N^*_\dagger(v)$ be the set of vertices in $N^*(v)$ that are neighbors of vertices in $N^\dagger(v)$. Note that every vertex in $N^\dagger(v)$ has degree ≥ 3, is connected only to v and to $N^*_\dagger(v)$, and no two vertices x and y in $N^\dagger(v)$ are such that $N(x) \subseteq N(y)$.

Proposition 4.52 $|N^\dagger(v)| \leq 3/2|N^*_\dagger(v)|$.

Lemma 4.53 *The number of vertices in category (2) (i.e., the number of vertices not in $V[\Re]$) is bounded by $18k$.*

To bound the number of vertices in category (1), fix a region $R(v, w)$ between $v, w \in D$. We have the following lemma.

Lemma 4.54 *Let $R = R(v, w)$ be a region in $V[\Re]$. The number of vertices in $V(R)$ is bounded by 16.*

Theorem 4.55 *The number of vertices in the reduced graph G is bounded by $67k$.*

Proof. By Lemma 4.53, the number of category-(2) vertices in G is bounded by $18k$. According to the discussion before, if we use the $18k$ upper bound on the number of category-(2) vertices, then we can assume that each region in $\Re$ is nice (if this is not the case we obtain a better upper bound on the total number of vertices in G). By Corollary 4.50, the number of regions in $\Re$ is bounded by $3k - 6$. According to Lemma 4.54, the number of vertices in $V(R)$, where $R \in \Re$ is a nice region, is bounded by 16. It follows that the number of vertices in $V(\Re)$ is bounded by $48k - 96$. Thus, the number of vertices in $V[\Re]$, and hence, in category (1), is bounded by $48k - 96$ plus the number of vertices in D which are the endpoints of the regions in $\Re$. Therefore the number of vertices in $V[\Re]$ is bounded by $49k - 96$, and the total number of vertices in G is bounded by $67k - 96 < 67k$. ∎

Corollary 4.56 *Let G be a planar graph with n vertices. Then in time $O(n^3)$, computing a dominating set for G of size bounded by k can be reduced to computing a dominating set of size bounded by k, for a planar graph G' of $n' < n$ vertices, where $n' \leq 67k$.*

Observe that, although the reasoning that shows the upperbound on the kernel size in the case of PLANAR DOMINATING SET is quite complex, the resulting algorithm is again quite simple: it is an instantiation of the general blueprint outlined in Alg. 5.

4.7 Kernelization schemes

In the design of approximation algorithms, it has been quite popular to design *algorithm schemes*, i.e., families of algorithms that, in the case of approximation, provide better and better approximation guarantees, at the expense of higher and higher running times. In the case of approximation algorithms, such schemes were called *approximation schemes.* Is it possible to translate this idea into the realm of parameterized algorithmics?

To this end, let us return to a problem that we already discussed in Chap. 3: LINEAR ARRANGEMENT. We have already seen through Theorem 3.11 that LINEAR ARRANGEMENT is fixed parameter tractable. However, observe that we also considered the problem LINEAR ARRANGEMENT restricted to connected graphs and showed $\mathcal{FPT}$-membership by providing a kernel with at most $k+1$ vertices (the more general kernel was measured in terms of edges).

We have also observed that components can be treated separately. Obviously, a component of size one or two (measured in terms of number of vertices) can be trivially solved at the end of the algorithm:

Reduction rule 37 *If G has two or less vertices, then construct an arbitrary arrangement onto neighbored numbers.*

In combination with Lemma 3.4, this shows the following:

Proposition 4.57 *A reduced instance of* LA *has at most* $1.5k$ *vertices.*

Proof. Since the instance is reduced and since each component can be treated separately, the number of vertices n in a reduced instance is the sum of the vertices n_c in the components c. Each n_c can be upperbounded by k_c+1, where k_c is the proportion of the parameter that is spent on component c. Hence:

$$n = \sum_c n_c \le \sum_c (k_c + 1) = \sum_c k_c + \#_c = k + \#_c$$

where $\#_c$ denotes the number of components of the instance. However, since we are dealing with a reduced instance, there are at most $n/3$ components in the instance: namely, each component has at least three vertices. Therefore:

$$k \ge n - \frac{n}{3} = \frac{2}{3}n.$$

This shows the claim. ∎

In fact, our reduction rule can be generalized: optimal arrangements for all graphs up to $q-1$ vertices may be precomputed (taking exponential time, measured against q); this shows that each component then has at least q vertices. After formulating an appropriate reduction rule, we may deduce:

Proposition 4.58 *For each fixed q, a reduced instance of* LA *has at most* $\frac{q}{q-1}k$ *vertices.*

This certainly has the flavor of an algorithmic scheme: at the expense of larger and larger running times, we may get smaller and smaller kernels (let q grow).

In fact, we have already seen a similar example before: Rule 21 is a kernelization scheme, since we would have to test larger and larger neighborhoods, although for our analysis of the kernel size for NONBLOCKER SET, we can only exploit the rule for neighborhoods up to size two. Similarly, a naive search for crowns can be viewed as a kernelization scheme, since again larger and larger (potential) crowns should be examined.

In Chap. 6, we will encounter further such examples.

4.8 Compilability and kernels

The idea of using preprocessing rules is not new but has been around in the heuristics community for years: it is in fact a very natural approach to solve hard problems by making them as small as possible to be able to attack them. The main interesting thing of parameterized algorithmics is that it allows a mathematical analysis of this heuristic approach and hence somehow explains the success of preprocessing in practice.

In fact, there exists an alternative way of mathematically analyzing preprocessing rules, mainly developed by P. Liberatore and his colleagues: they term the preprocessing *compilation* and analyze the *compilability* of intractable problems. This approach is based on the following observation: fast algorithms can be realized when part of the input data (called *fixed input part*) is known long before the rest of the input (called *varying input part*). Then, it makes sense to spend some (more) computation time to analyze the fixed part to speed up the solution when the varying part arrives. More technically speaking, a problem is *compilable* if the fixed input part x can be transformed (by arbitrarily complex preprocessing computations) into a part x' such that the size $|x'|$ is upperbounded by a polynomial in $|x|$ and there is a polynomial-time algorithm that solves the overall problem in time measured in $|x'|$ and the varying input part size $|y|$. To a certain extent, this approach also allows to separate an *on-line behavior* from an *off-line behavior*.

However, this idea is a bit contradicting the basic idea of parameterized algorithmics in the sense that in many concrete applications, the fixed input part is in fact large while the varying input part is rather small. This might be the main intuitive reason why results on compilability do significantly differ from results in computational complexity.

As an example, consider the following problem taken from [272]:

Problem name: MINIMAL DIAGNOSIS (MD)
Given: A finite set of faults F, a set of effects M, a function $e : F \to 2^M$ relating faults and effects, a set of observed effects $M' \subseteq M$, an integer k
Parameter: the (size of the) relating function e
Output: Is there a set $F' \subset F$ with $|F'| \leq k$ such that $M' \subseteq \bigcup_{f \in F'} e(f)$?

The intuition is to model a diagnosis system:[8] each fault may cause several effects (as modelled by the function e), and e is known before (as fixed part). At a certain time, several (unwanted) effects M' are observed. Then, it would be nice to quickly give a minimal-size diagnosis; the quest for a diagnosis of minimal size is basically motivated by Occam's razor as philosophical background; from a practical viewpoint, it would be of course also desirable to repair as few faults as necessary to save costs.

By viewing the faults as vertices of a hypergraph and the elements of M' as hyperedges, it is obvious that MINIMAL DIAGNOSIS naturally relates to HITTING SET: the main difference is the different paramaterization (an issue also discussed in Chapter 3), since within MINIMAL DIAGNOSIS, the parameter is basically the size of the super-hypergraph from which M' selects a hyperedge-induced sub-hypergraph. Therefore (as already observed in [272]), MD is fixed-parameter tractable by building a two-column table in which each row contains, in the first column, the set of effects corresponding to a particular set of failures (e.g., as a bit-vector) and in the second column, the size of the minimal diagnosis. After this preprocessing (which yields a table that is exponentially large, measured in terms of the parameter), the solution to a concrete instance (as specified by the observation M') is a simple table look-up.

Observe that although the sketched kernelization-like algorithm shows that the problem is parameterized tractable, it does not show compilability, since the table is not of polynomial size. In fact, most papers on compilability focus on negative results. For example, the compilability of MINIMAL DIAGNOSIS would entail unexpected collapses in classical complexity theory.

Another "natural" parameterization of MINIMAL DIAGNOSIS would be the size of the observed effects (in the hope there are not too many of these unwanted observations). Again, membership in $\mathcal{FPT}$ is close to trivial.

More details on compilation, including discussions on the relations between compilability and parameterized tractability can be found in [66, 272] and in related papers quoted therein.

[8]Observe that this is very much related to the *theory of diagnosis* treated elsewhere in this Habilitationsschrift. However, the parameterization is different.

Unfortunately, there are only few natural examples of compilable problems around (as far as we observed). In the context of paramaterized algorithmics, the *revised knowledge base* problems from [65] seem to be quite interesting, since M. Cadoli *et al.* showed that, under many forms of "belief revisions," given a set of beliefs T and some additional (possibly contradictory) information P, a "revised belief" T' can be computed in time polynomial in $|T|$ but exponential in $|P|$. This gives an $\mathcal{FPT}$ algorithm if the problem is parameterized by the size of the additional information $|P|$. Note that this parameterization is different from the viewpoint of compilability, which would allow large computations in terms of $|T|$. Therefore, these results are listed under the headline of *bounded revision.* However, these examples do perfectly fit into the landscape of parameterized algorithmics, since in practice the size of the background knowledge database, i.e., $|T|$, would be rather large, while the additional information P would be comparatively small.

It would not be surprising if the future shows more fruitful dialog between compilability and parameterized tractability.

Chapter 5

Search trees

Again, R. Downey and M. R. Fellows [134] classify the *search tree method* we are going to exhibit in this section as an *ad hoc method.* And again, they are right. However, besides kernelization this is the other standard methodology to get *efficient* parameterized algorithms. The downside is here that we know of no guarantee that actually all problems in $\mathcal{FPT}$ admit some sort of search tree algorithm —if we rule out the fact that the kernel that can be always obtained according to Theorem 2.4 may be "searched" by brute force; this does not yield the kind of search trees we are after.

From a practical point of view, this approach has lots of similarities to branch-and-cut algorithms and similar paradigms, so that again parameterized algorithmics might turn out as a way of mathematically analyzing and justifying (pruning) heuristics applied by practitioners anyhow to solve combinatorially hard problems.

As such, search tree algorithms are nothing new. They have been along since a long time. Robson's work [342] on MAXIMUM INDEPENDENT SET and Fomin, Kratsch and Woeginger's recent work [187] on MINIMUM DOMINATING SET are examples showing that the techniques we present here likewise apply to the non-parameterized setting. More can be found in Chap. 10.

For logical (satisfiability) type problems, many variants of search tree algorithms have been also proposed. A very elaborate example of this approach can be found in [266], albeit the earlier mentioned (more "parameterized") papers [212, 285] also follow the same basic methodology, which is a search tree that is build along ideas stemming from Davis-Putnam algorithms for dealing with satisfiability problems.

As a warm-up to this chapter, let us therefore reconsider the problem SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) as introduced in the end of Chap. 3.

Observe that the number of clauses of size three decreases in both re-

Algorithm 15 A simple search tree algorithm, called 3SAT

Input(s): a set F of clauses, at most k of them having three literals
Output(s): YES if there is a satisfying assignment for F NO if no such satisfying assignment exists.

```
    if $k \leq 0$ then
      {there are no more clauses of size three in the formula $F$}
      return 2-SAT($F$)
      {2-SAT can be solved in polynomial time}
 5: else
      Choose variable $x$ that occurs in some clause $c$ with three literals.
      Let $\ell$ be the number of clauses of $F$ in which $x$ occurs either as positive
      or as negative literal.
      Let $F'$ be the formula obtained from $F$ by setting $x$ true.
      Let $F''$ be the formula obtained from $F$ by setting $x$ false.
10:   if 3SAT($F', k - \ell$) then
        return YES
      else
        return 3SAT($F'', k - \ell$)
      end if
15: end if
```

cursive calls in Alg. 15, irrespectively of whether x is assumed to be true or false; in the first case, the clause c under consideration will be satisfied; in the latter case, the clause c will become a clause with at most two literals, and therefore k will be also decremented in that case. In fact, our algorithm can be further improved, as it is sketched for other problems in what follows. There, we will also learn how to actually analyze the running time of such search trees.

This chapter is structured as follows: we first turn our attention to cover problems. Embedded in that section, the reader can find some more detailed description how to prove upperbounds on the running times of search tree algorithms. As already mentioned in the foreword, we consider as one of the issues of this Habilitationsschrift to show how to develop improvements for parameterized algorithms, not being content with mere classifications. In Sec. 5.2, we show a somewhat traditional way of doing this, namely by intricate case analysis of local situations; we also call this the bottom-up approach to algorithm improvements. We rather advocate a top-down approach to this improvement problem, which should lead to simpler algorithms (in terms of implementation work) than the previous approach; however, the mathematics that need to be analyzed might be more tricky. More on this latter approach can be found in Sec. 5.3.

The reader who wishes to first see a concrete example of a search tree algorithm together with a worked-out search tree might skip to Sec. 6.4.2, where a particular graph drawing problem is studied.

5.1 Cover problems

The classical and surely most simple cover problem is VERTEX COVER.

5.1.1 VERTEX COVER

Problem name: VERTEX COVER (VC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $|C| \leq k$?

To our knowledge, the very first parameterized algorithm for this problem, together with its analysis, was published by K. Mehlhorn in his book on graph algorithms [292], much predating the formal advent of parameterized algorithmics.

Algorithm 16 A simple search tree algorithm, called VCMH

Input(s): a graph $G = (V, E)$, a positive integer k
Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

```
if $k \leq 0$ and $E \neq \emptyset$ then
  return NO
else if $k \geq 0$ and $E = \emptyset$ then
  return YES
else
  Choose edge $e = \{x, y\} \in E$
  if VCMH($G - x, k - 1$) then
    return YES
  else
    return VCMH($G - y, k - 1$)
  end if
end if
```

In fact, this is a very simple search tree algorithm. This simplicity has two immediate mathematical consequences:

- The correctness of the algorithm can be verified in a nearly trivial yet formal manner.
- The running time can also be easily estimated.

Due to the simplicity of this algorithm, we will perform both analyses in the following.

Let us remark that the way this search tree algorithm was obtained can be seen as a specific technique for designing search tree algorithms: the choice of an appropriate, provably small *candidate set*; in each branching step, simply all possible candidates are considered one after the other. The correctness of a search tree algorithm based on candidate set branching can be basically seen by showing that, in every solution (or at least in every solution that is obeying the constraint imposed by the given parameter value), at least one of the elements in the candidate set has to show up.

The correctness of Mehlhorn's algorithm:
Let us consider the following variant VCMH-C of Mehlhorn's algorithm (the pseudocode is listed in Alg. 17) that actually produces a cover if possible. In a certain sense, this algorithm can be seen as an extension of the previous algorithm, making the produced cover explicit.

This is indeed a fairly generally applicable technique to transform search tree algorithms for pure decision problems into search tree algorithms that actually produce the desired solution. In order to keep our algorithm presentation as simple as possible, we will refrain from actually constructing the solutions if ever possible, but the reader should be able to do this simple transformation on her/his own.

The proof of the correctness of this algorithm can be formally seen by induction on the number of edges in G.

First observe that, if $k \leq 0$ but $E(G) \neq \emptyset$, there cannot be any valid cover for G.

If $E(G) = \emptyset$, then the algorithm correctly returns $\emptyset$ as minimum cover set.

Otherwise, the algorithm chooses an arbitrary candidate edge $e = \{x, y\}$ and then branches according to which of the two incident vertices goes into the cover. This is valid, since any edge must be covered by a feasible solution. In other words, an edge is a valid candidate set. In the recursion call, the graph is modified by deleting, e.g., x and its incident edges from G (which are the edges that are covered by taking x into the cover), i.e., the procedure is called with arguments $G - x$ and $k - 1$, since obviously the "budget" of vertices that can still go into a cover is decremented by putting x into the cover.

Algorithm 17 A simple search tree algorithm, called VCMH-C

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): a vertex cover $C \subseteq V$, $|C| \leq k$, (if possible) or NO if no vertex cover of size at most k exists.

```
   if $k \leq 0$ and $E \neq \emptyset$ then
     return NO
   else if $k \geq 0$ and $E = \emptyset$ then
     return $\emptyset$
5: else
     Choose edge $e = \{x, y\} \in E$
     if $C :=$ VCMH-C$(G - x, k - 1) \neq$ NO then
       return $C \cup \{x\}$
     else if $C :=$ VCMH-C$(G - y, k - 1) \neq$ NO then
10:    return $C \cup \{y\}$
     else
       return NO
     end if{branching}
   end if
```

So, if we assume by induction hypothesis that the procedure works correct for all input graphs of up to m edges, and we consider an input graph with $m + 1$ edges, then either (according to our first observation) the routine returns NO due to an insufficient budget, or it will branch as described above. In each branch, the number of edges is decreased at least by one, so that the induction hypothesis applies, so that either NO is forwarded along the recursion or to the cover set produced by the recursive call x (or y) is correctly added and handed back to the calling procedure. Therefore, the procedure works correctly for all input graphs.

Time complexity:
Let us work out the time spent when processing a graph with n vertices and m edges, given a parameter k.

The tests to see if k has a specific value or if $E(G)$ is empty can be done in $\mathcal{O}(1)$ time. By appropriate data structures, choosing an arbitrary edge is also doable in time $\mathcal{O}(1)$. However, producing $G - x$ in takes $\mathcal{O}(n)$ time. By induction on the depth k of the search tree, a running time of $\mathcal{O}(2^k n)$ follows.

After having developed a simple and correct search tree algorithm, it is of course desirable to *improve* on the estimates of its running time. To this end, it turns out to consider another variant of Mehlhorn's simple algorithm:

Algorithm 18 A simple search tree algorithm, called VCMH′

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

```
if $k \leq 0$ and $E \neq \emptyset$ then
  return NO
else if $k \geq 0$ and $E = \emptyset$ then
  return YES
else
  Choose edge $e = \{x, y\} \in E$
  if VCMH′$(G - x, k - 1)$ then
    return YES
  else
    return VCMH′$(G - N(x), k - \deg(x))$
  end if
end if
```

The correctness of this variant is based on the observation that if x is *not* taken into the cover, then *all* neighbors of x (not only y) must be put into the cover. So, in a sense, this means that we rather branch on vertices here than on edges: either x goes into the cover or not. In a certain sense, this *binary branching* is not based on the selection of the edge e at all. We only need e to guarantee that the chosen vertex x has any neighbors.

If $\deg(x) = 1$, then actually nothing changes with respect to the previous analysis: in both branches, the parameter decreases by one. But this is obviously a very special case. More specifically, if we *knew* that all vertices of the graph instance had degree at most one, then the whole graph would be a collection of isolated vertices and isolated edges. It is clear that such an instance can be deterministically solved by a simple greedy algorithm; actually, we can use algorithm VCMH without any further branching.

This gives the variant VCMH-TL.

The strategy to improve on the running time (analysis) could be hence paraphrased as “Leave the simple cases till the end.” We will therefore call this design technique *triviality last.*

We can actually deal with vertices up to degree two this way: namely, observe that a graph with maximum degree of two is a forest of cycles and paths, and these structures can be resolved by trivial dynamic programming, where in a first step to resolve cycles, an arbitrary vertex is put into the cover. In VCMH-TL, only two lines need to be updated to this situation:

Algorithm 19 A still simple search tree algorithm, called VCMH-TL

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

```
    if $k \leq 0$ and $E \neq \emptyset$ then
      return NO
    else if $k \geq 0$ and $E = \emptyset$ then
      return YES
 5: else if possible then
      Choose vertex $x \in V$ such that $\deg(x) \geq 2$.
      if VCMH-TL$(G - x, k - 1)$ then
        return YES
      else
10:     return VCMH-TL$(G - N(x), k - \deg(x))$
      end if
    else
      {vertex selection not possible $\rightsquigarrow$ maximum degree is 1}
      resolve deterministically
15: end if
```

- line 6 becomes "Choose vertex $x \in V$ such that $\deg(x) \geq 3$" and
- line 13 becomes "{ ... maximum degree is 2 }."

Hence, we observe the following branching behavior:

1. If x is put into the cover, the parameter is reduced by one.
2. If x is not put into the cover, the parameter is reduced by at least three.

Therefore, the recurrence for the number of leaves in the search tree of the worst case (when $\deg(x) = 3$) can be expressed as follows:

$$\begin{aligned} T(k) &= \mathcal{O}(1) \text{ for } k < 3 \\ T(k) &= T(k-1) + T(k-3) \text{ for } k \geq 3 \end{aligned}$$

Solving this recurrence with the ansatz $T(k) = c^k$ gives: $T(k) \leq 1.4656^k$. In terms of parameterized algorithm analysis, this is indeed a huge progress when compared to the earlier analysis leading to the estimate 2^k.

Lemma 5.1 *The modified Algorithm 19 solves k-VERTEX COVER in time $\mathcal{O}(1.4656^k n)$ on an instance with n vertices.*

Further improvements on VERTEX COVER will be reported in the following sections.

Let us remark that also the simple improvement we observed (in comparison with Mehlhorn's original algorithm) is only possible if we assume (as it is the usual case with graph algorithms) that the whole graph instance is input to the algorithm at the very beginning. In other words, we nearly exclusively tackle the problem of developing *off-line algorithm*s in the parameterized framework. If, in this example, the graph would be given edge after edge and decisions on how the specific edge is going to be covered are to be immediately made, then we would face the task of developing a parameterized *off-line algorithm.* This area has only be superficially scratched in [23], as it will be explained in Sec. 6.2.2. In the specific case of VERTEX COVER, where edges are given one after the other, the algorithm of Mehlhorn can be used to obtain a 2^k branching behavior, and the example of k disjoint copies of $K_{1,1}$ shows that there is no way to avoid this 2^k branching, since at no stage it is known that all vertices have indeed a degree of only one.

We have just described how to improve on search tree algorithms with the triviality last principle. This name alone indicates that we can also think about the opposite strategy, namely *triviality first.* In this approach, we basically make use of reduction rules, again. More specifically, consider the following two rules for VERTEX COVER:

Reduction rule 38 (Rule 1) *If* $\deg(x) = 0$*, then remove* x*.*

Reduction rule 39 *If* $\deg(x) = 1$*, then place* $N(x)$ *into the vertex cover set and remove* $N[x]$*.*

It is clear that both rules are sound:

- An isolated vertex cannot cover any edge by definition, and therefore it is not useful to put it into any small vertex cover set.
- A vertex of degree one only covers one edge by definition, so that taking the other endpoint of that edge would never be worse. However, by doing so we could possibly cover other edges, as well.

Observe that both rules can be seen as special cases of Rule 26.

This leads us to the following algorithm VCMH-TF, which is rather generic, since there is no specific implementation of what is meant by "reduction rules." Of course, to be able to conclude with a reasonable time analysis, at least the mentioned two reduction rules are addressed.

Algorithm 20 Yet another simple search tree algorithm, called VCMH-TF

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

```
    Exhaustively apply the reduction rules.
    {The reduced instance is also called $G = (V, E)$.}
    if $k \leq 0$ and $E \neq \emptyset$ then
      return NO
5:  else if $k \geq 0$ and $E = \emptyset$ then
      return YES
    else
      Choose some vertex $x \in V$
      if VCMH-TF$(G - x, k - 1)$ then
10:     return YES
      else
        return VCMH-TF$(G - N(x), k - \deg(x))$
      end if
    end if
```

Since we only "destroy" vertices of degree zero and one by our reduction rules 1 and 39, in the worst case we branch at vertices of degree two. Therefore, the estimate of the running time becomes worse:

$$\begin{aligned} T(k) &= \mathcal{O}(1) \text{ for } k < 2 \\ T(k) &= T(k-1) + T(k-2) \text{ for } k \geq 2 \end{aligned}$$

Solving this recurrence with the ansatz $T(k) = c^k$ gives: $T(k) \leq 1.6181^k$.

Lemma 5.2 *Algorithm 20 solves k-*VERTEX COVER *in time $\mathcal{O}(1.6181^k n)$ on an instance with n vertices.*

The two mentioned ways to deal with triviality are not that far off from each other as it might appear at first glance; the algorithm based on triviality first can be even transformed into an algorithm based on triviality last (at least in this specific case). More precisely: whenever one succeeds in developing reduction rules for an otherwise trivial search tree algorithm purely based on the triviality first principle, then one can also define the class of instances $\mathcal{I}$ that can be completely resolved by these reduction rules and then possibly take a subclass of instances $\mathcal{I}' \subseteq \mathcal{I}$ (for which membership can be easily detected for a given instance), so that the trivial branching is only done as long as the instance under consideration is not in $\mathcal{I}'$. This way, one obtains an algorithm based on the triviality last principle.

In our concrete example of the reduction rules 1 and 39, the class of graphs immediately recognizable as being completely resolvable by these rule is the class of graphs with maximum degree one. Let us remark here that the two reduction rules—together with a trivial commitment (i.e., one branching) of one vertex per cycle component—can be actually used to completely resolve all graphs of maximum degree two.

However, it is in general not possible to describe a sufficiently large class of instances of the class of instances that can be efficiently resolved (and hence taken as a base for a simple search tree algorithm based on the triviality last principle) merely by reduction rules, so that a conversion into a simple search tree algorithm suitable for a triviality first approach is not always possible. In our example of VERTEX COVER, it is not clear how to always deal with vertices of degree two, at least not at first glance. However, in this example, Chen, Kanj and Jia [86] (also see [251]) found a reduction rule that deals with vertices of degree two:

Reduction rule 40 *Let (G, k) be an instance of* VERTEX COVER*. Let $v \in V(G)$ such that $N(v) = \{u, w\}$, i.e.,* $\deg(v) = 2$.

- *Assume that $u \notin N(w)$. Construct the graph $G' = G[u = w] - v$. Then, the reduced instance is $(G', k - 1)$.*
- *If $u \in N(w)$, then put u, w into the vertex cover and reduce to $(G - (\{u, v, w\}), k - 2)$.*

In fact, only the first part of Rule 40 is known as *folding rule.* The second part can be actually generalized as follows:

Reduction rule 41 *If $C \subseteq V(G)$ is a clique in a graph instance (G, k) of* VC *such that C contains a vertex v with $N[v] \subseteq C$, then reduce to $(G - C, k - (|C| - 1))$.*

Observe that Rule 41 can be again seen as a kind of kernelization scheme rule, since the running time for detecting large cliques scales up exponentially with the size of the clique, although we do not know of an exploit for this rule (in terms of analysis) if $|C| > 3$.

In other words, also Alg. 20 can be improved to run in time $\mathcal{O}(1.4656^k n)$.

The difference between the two approaches will become more transparent when we do more involved subcase analysis in the following section. There, it makes a difference whether certain vertices have been actually removed from the graph instance with the help of reduction rules, i.e., with the trivial-first approach, or whether these vertices are still around and may interact with other vertices in the graph.

Put in another way: while the triviality last approach is suitable for quickly developing search tree algorithms with reasonable running times, it is not easy (and actually, we don't know of any such example) to further improve on such search tree algorithms by any form of "clever branching." By way of contrast, the triviality first approach (where the trivial cases are usually resolved by reduction rules) tends to lead to running times that are worse than the ones from the triviality last approach. However, they might be improvable by clever branching, as seen below in many instances.

At this stage, let us mention that it is also possible to combine both approaches, leading to the following, typical general scheme detailed in Alg. 21 that first applies reduction rules and then only branches at non-trivial cases:

Algorithm 21 A simple search tree algorithm, called VCMH-TD

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or NO if no vertex cover of size at most k exists.

```
    Exhaustively apply the reduction rules.
    {The reduced instance is also called $G = (V, E)$.}
    if $G$is trivial then
      resolve $G$ deterministically in polynomial time
5:  else
      Choose some vertex $x \in V$
      if VCMH-TD($G - x, k - 1$) then
        return YES
      else
10:     return VCMH-TD($G - N(x), k - \deg(x)$)
      end if
    end if
```

As within algorithm VCMH-TL (see Alg. 20), triviality basically means to have maximum degree two. Observe that we now could subsume the other trivial cases (basically, when there are no edges left over) under this headline. The analysis of VCMH-TL basically transfers, so that the exponential part of the running time can be estimated as $T(k) \leq 1.4656^k$.

Lemma 5.3 *Algorithm 21 solves k-VERTEX COVER in time $\mathcal{O}(1.4656^k n)$ on an instance with n vertices.*

This scheme is typical for the top-down approaches discussed in Sec. 5.3. The point is that the branching part itself is still rather trivial, where this part becomes really tricky with algorithms developed by a rather bottom-up approach. We will see a couple of examples for this from the following subsection on.

One peculiar important property shared by all three algorithms VCMH-TL, VCMH-TF and VCMH-TD is that their correctness immediately transfers from the correctness of the basic algorithm VCMH as long as

- the reduction rules are sound and
- the algorithms for solving the trivial cases are correct and properly called from the main algorithm.

This is different from the bottom-up approach we present in the next subsection(s), where the correctness of the over-all algorithm must be proven, in addition.

Let us finally mention that it is actually hard to pin down the sources of this subsection: VERTEX COVER is simply the standard example for parameterized algorithmics. What might be new, however, is the stress of the strategies (triviality first versus triviality last) how to improve on parameterized search tree algorithms.

5.1.2 The time analysis of search tree algorithms

In this section, we will first try to generalize our observations on how to analyze the running time of search tree algorithms and then apply our results to the analysis of SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) algorithms.

In the cases we have seen so far, it was not hard to obtain recurrences of the form:

$$T(k) \leq \alpha_1 T(k-1) + \alpha_2 T(k-2) + \cdots + \alpha_\ell T(k-\ell) \tag{5.1}$$

for the size $T(k)$ of the search tree (which can be measured in terms of the number of leaves of the search tree, since that number basically determines the running time of a search tree based algorithm).

More specifically, α_i is a natural number that indicates that in α_i of the $\sum_j \alpha_j$ overall branches of the algorithm, the parameter value k got decreased by i. Notice that, whenever $\ell = 1$, it is quite easy to find an estimate for $T(k)$, namely α_1^k. But how can we deal with the more general case? A recipe is contained in Alg. 22.

Algorithm 22 Simple time analysis for search tree algorithms, called ST-simple

```
Input(s): a list $\alpha_1, \ldots, \alpha_\ell$ of nonnegative integers, viewed as coefficients of
  the inequality (5.1)
Output(s): a tight estimate $c^k$ upperbounding $T(k)$
  Consider inequality (5.1) as equation:
      $T(k) = \alpha_1 T(k-1) + \alpha_2 T(k-2) + \cdots + \alpha_\ell T(k-\ell)$
  Replace $T(k-j)$ by $x^{k-j}$, where $x$ is still an unknown to be determined.
  Divide the equation by $x^{k-\ell}$.
  {This leaves a polynomial $p(x)$ of degree $\ell$.}
  Determine the largest positive real zero (i.e., root) $c$ of $p(x)$.
  return $c^k$.
```

Why does that algorithm work correctly? Please observe that in the simplest case (when $\ell = 1$), the algorithm does what could be expected. We only mention here that

$$p(x) = x^\ell - \alpha_1 x^{\ell-1} - \cdots - \alpha_\ell x^0$$

is also sometimes called the *characteristic polynomial* of the recurrence given by Eq. 5.1, and the base c of the exponential function that Alg. 22 returns is called the *branching number* of this recurrence. Due to the structure of the characteristic polynomial, c is the only positive root.

Alternatively, such a recursion can be also written in the form

$$T(k) \leq T(k-a_1) + T(k-a_2) + \cdots + T(k-a_r).$$

Then, $(a_1, \ldots, a_r)$ is also called the *branching vector* of the recurrence.

As detailed in [209, pp. 326ff.], a general solution of an equation

$$T(k) = \alpha_1 T(k-1) + \alpha_2 T(k-2) + \cdots + \alpha_\ell T(k-\ell)$$

(with suitable initial conditions) takes the form

$$T(k) = f_1(k)\rho_1^k + \cdots + f_\ell(k)\rho_\ell^k,$$

where the ρ_i are the distinct roots of the characteristic polynomial of that recurrence, and the f_i are polynomials (whose degree corresponds to the degree of the roots (minus one)). As regards asymptotics, we can conclude $T(k) \in \mathcal{O}^*(\rho_1^k)$, where ρ_1 is the dominant root.

The exact mathematical reasons can be found in the theory of polynomial roots, as detailed in [209, 183, 210, 266]. It is of course also possible to check the validity of the approach by showing that $T(k) \leq \rho^k$ for the obtained solution ρ by a simple mathematical induction argument.

There are two more general settings which we will encounter in this Habilitationsschrift:

- Due to case distinctions that will play a key role for designing refined search tree algorithms, the recurrences will then take the form

  $$T(k) \leq \max\{f_1(k), \dots, f_r(k)\},$$

  where each of the $f_i(k)$ is of the form

  $$f_i(k) = \alpha_{i,1}T(k-1) + \alpha_{i,2}T(k-2) + \cdots + \alpha_{i,\ell}T(k-\ell).$$

  Such a recurrence can be solved by r invocations of Alg. 22, each time solving $T(k) \leq f_i(k)$. This way, we get r upperbounds $T(k) \leq c_i^k$. Choosing $c = \max\{c_1, \dots, c_r\}$ is then a suitable upperbound.

- We might have systems of equations. In its most general form, these will again include maximum operators that can be treated as explained in the previous point. Then, we are left with solving systems like

  $$T^\ell(k) \leq \sum_{i=1}^{r} f_i^\ell(k),$$

  where each of the $f_i^\ell(k)$ is of the form

  $$f_i^\ell(k) = \alpha_{i,\ell,1}T^i(k-1) + \alpha_{i,\ell,2}T^i(k-2) + \cdots + \alpha_{i,\ell,q_\ell}T^i(k-q_\ell),$$

  where $1 \leq i, \ell \leq r$. We shall follow the approach of first transforming this set of mutual recurrences into a single recurrence for the main component (usually T^1) only, by treating all involved inequalities as equalities. Then, we again apply Alg. 22.

  We should like to remark that this is not the only way of dealing with sets of linear recurrence equations that involve an *auxiliary parameter ℓ*. For instance, Kullmann and Wahlström included this auxiliary parameter by deriving a single recurrence equation from a set of recurrence equations in their algorithm analysis [266, 373], trading off ℓ against k, so to speak. Sometimes, even more involved methods tailored to the specific analysis of a certain algorithm may be worthwhile pursuing, as can be seen with the algorithm analysis of Monien and Speckenmeyer, see [296]. A more general and abstract approach called quasiconvex method was recently exhibited by Eppstein [155].

Let us finally return to SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION), the warm-up problem of this chapter. By the techniques developed so far for VC, the following result can be easily seen:

Theorem 5.4 *Alg. 15 solves a* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) *instance* (F, X, k) *in time* $\mathcal{O}^*(2^k)$.

We already mentioned that there are simple strategies to improve on this running time. One simple observation relies on the fact that, whenever a particular variable x always only occurs as say a positive literal in the formula F, then we can set x to true (without any further need to branch), this way simplifying the formula (possibly decreasing the number of clauses with three variables). This motivates the following reduction rule:

Reduction rule 42 *Let* (F, X, k) *be an instance of* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) *and* $x \in X$.

- *If* x *only appears as a positive literal in the formula* F*, then set* x *to true, remove all clauses in which* x *appears from* F*, remove* x *from* X *and reduce the parameter by the number of clauses of size three in* F *in which* x *appeared.*

- *If* x *only appears as a negative literal in the formula* F*, then set* x *to false, remove all clauses in which* $\bar{x}$ *appears from* F*, remove* x *from* X *and reduce the parameter by the number of clauses of size three in* F *in which* $\bar{x}$ *appeared.*

Similarly, the following observation is helpful (although it will not directly affect the analysis of the modified search tree algorithm for 3-SAT (CLAUSE)):

Reduction rule 43 *Let* (F, X, k) *be an instance of* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) *and* $x \in X$*. If* C *is a clause that contains only the variable* x*, then do one of the following things:*

- *If* x *and* $\bar{x}$ *are contained in* C*, then remove* C *from* F *(only affecting the parameter if* C *contained three literals).*

- *If only* x *is contained in* C*, then set* x *to true, remove* C *from* F *and* x *from* X *(only affecting the parameter if* C *contained three literals).*

- *If only $\bar{x}$ is contained in C, then set x to false, remove C from F and x from X (only affecting the parameter if C contained three literals).*

In fact, the first case of the previous rule can be formulated more generally as follows:

Reduction rule 44 *Let (F, X, k) be an instance of* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) *and $x \in X$. Assume that Rule 43 does not apply. If x and $\bar{x}$ are contained in C, then remove x and $\bar{x}$ from C, this way modifying F. Decrement the parameter (by one).*

The rules themselves are not new but can be found in basically any algorithm dealing with SATISFIABILITY problems. The correctness of the proposed modified search tree procedure for SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) relies on the soundness of the reduction rules. The proof of the following lemma is left to the reader.

Lemma 5.5 *Let (F, X, k) be an instance of* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION)*. Let (F', X', k') be obtained from (F, X, k) by applying either of the rules 42 or 43 or 44 to (F, X, k). Then, (F, X, k) is satisfiable if and only if (F', X', k') is satisfiable.*

If we now modify Alg. 15 by adding inbetween lines 5 and 6 of that procedure the following line of code:

Exhaustively apply rules 42, 43 and 44;
the resulting instance is also named (F, k).[1]

then we arrive at an algorithm for which we can prove:

Theorem 5.6 *The modified version of Alg. 15 solves a* SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) *instance (F, X, k) in time $\mathcal{O}^*(1.6182^k)$.*

Proof. (Sketch) The correctness of the procedure directly follows from the correctness of Alg. 15 and the soundness of the reduction rules.

Before branching, the instance (F, k) is reduced. In particular, this means that the variable x we choose is contained both as a positive and a negative literal; more precisely, there is one clause C in which x occurs as a positive literal and one clause C' in which x occurs as a negative literal $\bar{x}$ (otherwise, one of the reduction rules would trigger).

[1] Observe that in the search tree algorithm, we do not explicitly specify the set of variables X as an argument, since this can be implicitly deduced from the set of clauses.

Now, we consider two cases:

- The two clauses C and C' mentioned above both contain three variables. Then, we will arrive at the following recursion for limiting the size of the search tree:

  $$T(k) \leq 2T(k-2).$$

  Namely, irrespectively of whether we set x to true or false, at least two clauses of size three will vanish.

- W.l.o.g., consider the case that C contains three variables but C' contains two variables. Now, in the branch that x is set to true, $C' = \bar{x} \vee \ell$ can only be satisfied by setting the variable y of ℓ accordingly. According to the reduction rules, y will appear in another clause C_y; more precisely, if $\ell = y$, $\bar{y}$ will show up in C_y, and if $\ell = \bar{y}$, y will show up in C_y. If C_y contains only one other literal z, the argument will repeat, since the fixing of y will also fix z: possibly, the whole instance will be resolved this way. If not, finally a clause C'' will be reached in this sequence (in the simplest case, this is already C_y) that contains three literals. This clause C'' will be turned into a clause with only two literals.

  In the branch that x is set to false, nothing is gained. Altogether, this branch leads to the following recursion for limiting the size of the search tree:

  $$T(k) \leq T(k-1) + T(k-2).$$

Due to $T(k-1) \geq T(k-2)$, the worst case is obviously given by the second recursion, which gives the claimed result. ∎

5.1.3 WEIGHTED VERTEX COVER

Problem name: WEIGHTED VERTEX COVER (WVC)
Given: A graph $G = (V, E)$ with vertex weights $\omega : V \to \mathbb{R}_{\geq 1}$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $\omega(C) \leq k$?

In the following, the weight function ω will also be used for sets of vertices as argument.

The interesting thing of this weighted version of VERTEX COVER—and in contrast with the unweighted case—is that of the two simple reduction rules for VC we listed above, only the rule 1 for isolates is still valid. The problem with the degree-1 rule 39 is that there may be degree-1 vertices of relatively small weight whose inclusion in the vertex cover is actually better than taking its heavy-weight neighbor.

Yet, the simple VCMH algorithm is also valid in this case, since still each edge needs to be covered. Moreover, even the more sophisticated Algorithm 19, called VCMH-TL, based on the *triviality last* principle, together with the analysis is still valid, although a little care has to be applied to the trivial cases, i.e., the graph with maximum degree of two. Now, for paths actually some little dynamic programming must be performed (due to the inavailability of the second reduction rule for the weighted case as explained above), starting from one end of the path, working through towards the other. The according easily derived algorithm is left as an exercise, see [309]. Similarly, we can also develop an analogue to Alg. 21: again, the proof of the correctness of the algorithm itself and its running time upperbounds are easy to obtain.

Lemma 5.7 *Variants of Algorithm 19 and Algorithm 21 solve k-WEIGHTED VERTEX COVER in time $\mathcal{O}(1.4656^k n)$ on an instance with n vertices.*

However, the triviality first principle is harder to apply, since only Rule 1 is valid in the weighted case. Nonetheless, by using a bottom-up approach, the currently best exact algorithm for WEIGHTED VERTEX COVER was developed in [309], the variant of which that uses polynomial space is explicitly shown in Alg. 26 and the subsequently listed subroutines.

5.1.4 CONSTRAINT BIPARTITE VERTEX COVER

Let us reconsider the problem CONSTRAINT BIPARTITE VERTEX COVER introduced in Chap. 3. We already mentioned that an $\mathcal{O}(2^{k_1+k_2}k_1k_2 + (k_1 + k_2)|G|)$ has been developed by people working in that area, much predating the systematic development of $\mathcal{FPT}$ algorithms. This algorithm—as the stated running time indicates, was a combination of a kernelization and a search tree algorithm. Let us focus here on the search tree part.

In fact, it is not hard to see how to obtain a $\mathcal{O}^*(2^{k_1+k_2})$ algorithm: the basic observation for the development of Alg. 16 was that every edge $\{u, v\}$ needed to be covered, so branching into the to cases "u comes into the cover" or "v comes into the cover" is a complete case distinction. The same observation is also true for CONSTRAINT BIPARTITE VERTEX COVER. Since in either branch, k_1 or k_2 is decreased, the claimed running time follows.

How can this simple algorithm be improved? Let us (again) try the *triviality last approach.* To this end, we need to be able to deal with simple cases, where "simple" means graphs with a low maximum degree. If these simple cases can be dealt with in polynomial time, then it is clear that the simple branching strategy exhibited in Alg. 20 transfers to this case, as well.

Lemma 5.8 CONSTRAINT BIPARTITE VERTEX COVER *can be solved in polynomial time on forests of cycle and path components.*

Proof. (Sketch) First observe that this assertion is not completely trivial in this two-parameter VERTEX COVER variant. For example, a cycle C_8 of length eight would need every second vertex to be put into the cover, seen as an instance of VERTEX COVER. Hence, 4 vertices would need to be put into a cover. But this would not be possible if say $k_1 = k_2 = 2$. Hence, if $k_1 < 4$ and $k_2 < 4$, three vertices of one kind and two of the other would need to be put into a cover for C_8.

This is more generally true: if the parameter values k_1 and k_2 should suffice to cover a cycle component C_m, then if $m \leq 2k_1$ or $m \leq 2k_2$, the cycle can be optimally covered (as if it were a VERTEX COVER instance); otherwise, if $m-2 \leq 2(k_1+k_2)$, then such a cover can be constructed for C_m (as CBVC instance), and how this "change" between taking one sort of vertices or the other one is performed is arbitrary (where only "using" at least two vertices from each sort makes sense). Finally, if $m-2 > 2(k_1+k_2)$, then no solution is possible. This gives the following strategy for solving forests of cycles (with current parameter (k_1, k_2)):

```
Select a cycle $C$ of smallest length $m$.
Let $k_\ell$ be the smaller among $k_1$ and $k_2$.
**if** $m \leq 2k_\ell$ **then**
  Cover $C$ by taking $m/2$ vertices from $C$ from vertex set $V_\ell$.
**else if** $m - 2 \leq 2(k_1 + k_2)$ **then**
  Use $k_\ell$ vertices from $V_\ell$ and $(m+1)/2 - k_\ell$ from the other sort to cover
  $C$.
**else**
  The instance has NO solution.
**end if**
```

Why did we choose to first treat small cycles? As we argued above, when it is not possible to cover a cycle by vertices from one type, we would need (overall) one more vertex to cover that cycle. This observation is independent of the size of the cycles. Hence, leaving the large cycles for the end is better than the converse strategy. Moreover, it shows that at most one cycle will be covered non-optimally.

Similar considerations can be made for paths, although here the situation is slightly more intricate: there are basically three types of paths that need different considerations: (1) paths of odd length, (2) paths of even length that begin and end with vertices from V_1, and (3) paths of even length that begin and end with vertices from V_2. The additional problem is that paths of even length can be covered optimally (as a VC instance) by taking every second vertex into the cover, starting with the second vertex on a path; for a path of length m, $m/2$ vertices are needed this way. If there are, however, not "sufficiently" many vertices of a particular kind available (in the bipartite case), one more vertex will be needed in the cover. The situation is different with paths of odd length: there, the additional constraints put within the CBVC problem (in contrast with VC) do not incur additional costs; more precisely, a path of odd length as a vertex cover of size k if and only if it is solvable as part of a CONSTRAINT BIPARTITE VERTEX COVER instance, with any parameter values k_1 and k_2 that satisfy $k_1 + k_2 = k$.

To get a really efficient algorithm, we would therefore stick to the following strategy:

1. Solve all paths of even length, first solving short path components.
2. Solve cycles, first solving short cycles (see details above).
3. Solve paths of odd length.

Details are left to the reader. The claimed polynomial running time is then obvious. In fact, the polynomial is pretty small; assuming that the components are already sorted by size, it is even linear. ∎

This gives us the Algorithm listed in Alg. 23. We can conclude:

Theorem 5.9 CONSTRAINT BIPARTITE VERTEX COVER *can be solved in time* $\mathcal{O}^*(1.4656^{k_1+k_2})$.

By a much more involved case analysis, a search tree algorithm was developed for CONSTRAINT BIPARTITE VERTEX COVER that runs in time $\mathcal{O}^*(1.3999^k)$, see [180]. We omit stating the rather complicated algorithm here. However, let us mention that it would be another perfect example of the bottom-up approach illustrated in the next section.

When it comes to the analysis of search tree algorithms, there is one ingredient of [180] that is worth mentioning, since it might be applicable to other search tree algorithms, as well, the so-called *bonus point trick*. Namely, the search tree algorithm is basically an intricate implementation of the triviality-last principle, combined with a sophisticated analysis of many different local situations. In this sophisticated analysis, from time to time it happens that components consisting of at least two vertices, each of degree at most two,

Algorithm 23 A still simple search tree algorithm for CBVC, called CBVC-TL

Input(s): a bipartite graph $G = (V_1, V_2; E)$, positive integers k_1 and k_2
Output(s): YES iff there is a vertex cover $(C_1, C_2) \subseteq V_1 \times V_2$, $|C_1| \leq k_1$ and $|C_2| \leq k_2$

```
    if $k_1 + k_2 \leq 0$ and $E \neq \emptyset$ then
       return NO
    else if $k_1 + k_2 \geq 0$ and $E = \emptyset$ then
       return YES
 5: else if possible then
       Choose vertex $x \in V_1 \cup V_2$ such that $\deg(x) \geq 3$.
       if $x \in V_1$ then
          $d = (1, 0)$
       else
10:       $d = (0, 1)$
       end if
       if CBVC-TL$(G - x, (k_1, k_2) - d)$ then
          return YES
       else
15:       return CBVC-TL$(G - N(x), (k_1, k_2) - \deg(x)((1, 1) - d))$
       end if
    else
       {vertex selection not possible $\rightsquigarrow$ maximum degree is 2}
       resolve deterministically according to Lemma 5.8
20: end if
```

are split off. When this happens, we can (already) reduce the parameter by one, since we might (as well) already at that point completely resolve that component, and we would need at least one vertex to come into the cover for such a solution. Hence, we can count this as a "bonus point," although we will only later actually "use up" this bonus point for covering that component. Hence, formally the concrete parameter value no longer corresponds to the number of vertices put into the (partial) cover (implicitly) constructed so far on some branching path.

If one further requires that a solution to CONSTRAINT BIPARTITE VERTEX COVER is only accepted if it is also a minimum vertex cover (an assumption accepted in [279] to get a better solving strategy (heuristic) based on the *Dulmage-Mendelsohn decomposition* for bipartite graphs), then even an $\mathcal{O}^*(1.26^k)$ algorithm was derived [85] (their algorithm also makes use of the Dulmage-Mendelsohn decomposition); however, it appears to be questionable if this is the correct modelization of the original chip repair problem: why should a repair be rejected on the ground that it is not minimum (when seen as a VC instance)? A more authorative critique of this assumption from people that actually write programs that are used in reconfiguration lasers can be found in [227].

More interesting variants appear to be, from the practical point of view:

- weighted variants: in particular, it can be assumed that there is a different cost incurred for repairing rows or columns, and a solution that only uses say row repairs is preferrable to a solution that uses both row and column repairs, since changing between different repair modes takes additional time.
- variants that use more parameters (see Chap. 3)

Both main variants will be shortly discussed in the context of parameterized enumeration, see Chap. 8.

Let us only remark in this place that the variant which prefers to repair with only one sort of spare lines can be solved with what we already saw as follows. Let $(G = (V_1, V_2; E), k_1, k_2)$ be an instance of CONSTRAINT BIPARTITE VERTEX COVER. Recall the probem GENERALIZED VERTEX COVER that generalizes VERTEX COVER in the way that a specific set of vertices was prescribed out of which the cover vertices must be chosen.

- Try to solve VCGEN on the instance $((V_1 \cup V_2, E), V_1, k_1)$; if this is possible, this would be our repair proposal.
- Otherwise: Try to solve VCGEN on the instance $((V_1 \cup V_2, E), V_2, k_2)$; if this is possible, this would be our repair proposal.
- If the first two cases fail, we would have to solve the given CBVC instance in the "traditional way" (as indicated above).

Observe that VERTEX COVER and GENERALIZED VERTEX COVER are parameterized interreducible, see Cor. 4.41, so that the first two steps do not introduce additional running time cost in terms of $\mathcal{O}$-notation, since VERTEX COVER can be (currently) faster solved than CONSTRAINT BIPARTITE VERTEX COVER; moreover, assuming $k_1 \approx k_2$, the parameter values with which we call any VC algorithm are smaller than for CBVC.

5.2 Improved branching: a bottom-up approach

Although we already reported on a couple of algorithms for solving VERTEX COVER, the algorithms we listed are not among the best in terms of worst-case running times. Still, VCMH-TL and VCMH-TD are significantly better than the first improved branching algorithms, as reported, e.g., in [33, Theorem 1]. Note that it was the mentioned algorithm of Balasubramanian, Fellows and Raman that was chosen in the parallel implementation of VERTEX COVER algorithms as described in [80] in the dispatching phase, also see Sec. 8.5.2.

Let us now first present the second algorithm introduced in [33] as a kind of random example of the bottom-up approach, see Alg. 24.

The correctness of this algorithm must be argued on a case-by-case basis. For example, in lines 5 through 21, the case that x has degree two is discussed. In the corresponding subcases that lead to different branching scenarios, the subcase that y and z are both having no neighbor besides x is not mentioned. Nonetheless, the case distinction is complete, since this evidently means that bot y and z have degree one, and this case got treated in the very first case distinction. Observe that we could replace basically the first 21 lines of code by referring to the reduction rules we presented for getting rid of vertices of degree of at most two in the case of VERTEX COVER.

However, it is clear that in contrast to the simple algorithms we considered up to now, a formal correctness proof is pretty tedious. Moreover, observe that not all details of the algorithm have been given; rather, the lines of pseudo-code should give an impression of how complicated algorithms tend to be that are optimized according to the bottom-up approach.

The algorithm we presented here is not the one for which the smallest upper bound has been shown. When restricting oneself on algorithms that only use polynomial space, the currently best algorithm (that needs only polynomial space) is published in [86] (also see Kanj's PhD Thesis [251]) and runs in time $\mathcal{O}^*(1.2852^k)$. However, if exponential space is allowed, running times down to $\mathcal{O}^*(1.2745^k)$ are possible [77]. Those algorithms are not conceptually (and not from the point of view of implementing them) simpler than the one we listed to some detail. Moreover, it can be doubted if the exponential space requirement (of course, in the area of parameterized algorithmics, exponential space rather means "parameterized space") of the theoretically best algorithm pays off in practice, since time may be eaten up by page swapping.

As a second example of this approach (that is still prominent in the literature) we reconsider WEIGHTED VERTEX COVER. By using a complicated algorithm, Niedermeier and Rossmanith were able to prove the following theorem:

Theorem 5.10 WEIGHTED VERTEX COVER *can be solved in time* $\mathcal{O}(1.3954^k + kn)$.

Algorithm 24 An involved search tree algorithm, called VCBFR

Input(s): a graph $G = (V, E)$, a positive integer k

Output(s): YES if there is a vertex cover $C \subseteq V$, $|C| \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of size at most k exists.

```
if there is a vertex $x$ of degree zero then
  return VCBFR($G - x, k$)
else if there is a vertex $x$ of degree one then
  return VCBFR($G - N[x], k - 1$)
else if there is a vertex $x$ of degree two then
  {Let $y$ and $z$ be the two neighbors.}
  if $y$ and $z$ are neighbors then
    {$y$ and $z$ can go into vertex cover.}
    return VCBFR($G - \{x, y, z\}, k - 2$)
  else if $|(N(y) \cup N(z)) \setminus \{x\}| \geq 2$ then
    {Either $\{y, z\}$ are part of the cover or $N(\{y, z\})$.}
    if VCBFR($G - \{x, y, z\}, k - 2$) then
      return YES
    else
      return VCBFR($G - N[\{x, y\}], k - 3$)
    end if
  else
    {$y$ and $z$ have one other neighbor $a$ besides $x$.}
    {$x$ and $a$ can go into vertex cover.}
    return VCBFR($G - N[\{x, y\}], k - 2$)
  end if
else if there is a vertex $x \in V$ with $\deg(x) \geq 5$ then
  if VCBFR($G - x, k - 1$) then
    return YES
  else
    return VCBFR($G - N[x], k - \deg(x)$)
  end if
else if there is a vertex $x \in V$ with $\deg(x) = 3$ then
  {Let $x_1, x_2, x_3$ be the neighbors of $x$.}
  {See next algorithm.}
else
  {There is a vertex $x \in V$ with $\deg(x) = 4$.}
  {This case is similar and hence omitted.}
end if
```

Algorithm 25 What to do with a vertex x of degree three in VCBFR

```
    {Let $x_1, x_2, x_3$ be the neighbors of $x$.}
    if there are $1 \le i < j \le 3$ with $\{x_i, x_j\} \in E$ then
      {Branch at $x_\ell$ such that $\{x_i, x_j, x_\ell\} = N(x)$.}
      if VCBFR($G - N(x), k - 3$) then
5:      return YES
      else
        return VCBFR($G - N(x_\ell), k - 3$)
      end if
    else if there are $1 \le i < j \le 3$ with $N(x_i) \cap N(x_j) = \{x, y, \ldots\}$ then
10:   if VCBFR($G - N[x], k - 3$) then
        return YES
      else
        return VCBFR($G - \{x, y\}, k - 2$)
      end if
15: else if there is a $1 \le i \le 3$ with $|N(x_i)| \ge 4$ then
      if VCBFR($G - N[x], k - 3$) then
        return YES
      else if VCBFR($G - N(x_i), k - 4$) then
        return YES
20:   else
        return VCBFR($G - (\{x_i\} \cup N(N(x) \setminus \{x_i\})), k - 6$)
      end if
    else
      {There are no edges between $x_1, x_2, x_3$ and each $x_i$ has, apart from $x$,
      exactly two private neighbors: $N(x_i) = \{x, y_{i,1}, y_{i,2}\}$.}
25:   if There are no edges between $N[N(x)]$ and $V \setminus N[N(x)]$ then
        solve "tiny component" $G[N[N(x)]]$ by brute force and return ac-
        cording solution
      else
        further case distinctions omitted ...
      end if
30: end if
```

Actually, the additive $\mathcal{FPT}$ algorithm was obtained by combining the search tree algorithm described in the following with a kernelization algorithm based on a combination of Buss' reduction and Nemhauser-Trotter reduction, as described in Chapter 4 for the unweighted case.

The reader is encouraged to compare this running time with the one claimed in Lemma 5.7. Obviously, the complexity of Algorithm 26 in terms of

- proving the correctness of the algorithm,
- proving the running time of the algorithm and
- implementing the algorithm (without introducing any errors)

is much higher. It could be even questioned if the running time would be much better (compared to Algorithms 19 and 21) in practice, since much more work has to be done in each node of the search tree.

To simplify the notations when stating the algorithm of Niedermeier and Rossmanith, we write

WVC-NR-branch at x

to abbreviate:

```
**if** WVC-NR($G - x, k - \omega(x)$) **then**
  return YES
**else**
  return WVC-NR($G - N(x), k - \omega(N(x))$)
**end if**
```

The reader is encouraged to actually prove the correctness of the listed algorithm. Especially, this means that one has to argue why the many case distinctions are actually capturing all cases. Also, the quality of the branchings would have to be considered. The worst case branching can be described by the recurrence

$$T(k) \leq 2T(k-3) + T(k-4),$$

which gives the claimed upper bound on the running time.

Let us finally report on interesting automatic approaches that were recently developed to analyze local branching scenarios, see [211].

Algorithm 26 An elaborated search tree algorithm, called WVC-NR

Input(s): a graph $G = (V, E)$ with real vertex weights $\omega : V \to \mathbb{R}$ with $\min \omega(V) \geq 1$, a positive real k

Output(s): YES if there is a vertex cover $C \subseteq V$, $\omega(C) \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of weight at most k exists.

```
Resolve all graph components of maximum degree two in polynomial time. (*)
Resolve all graph components with no more than six vertices in constant time.
{The new (reduced) instance is also called $G = (V, E)$.}
if $k \leq 0$ and $E \neq \emptyset$ then
  return NO
else if $k \geq 0$ and $E = \emptyset$ then
  return YES
else if possible then
  Choose some vertex $x \in V$ with $\deg(x) \geq 4$
  WVC-NR-branch at $x$
else if possible then
  Choose some vertex $x \in V$ with $\deg(x) = 1$.
  Let $a$ denote the unique neighbor of $x$.
  return WVC-NR-deg1$(G, x, a, k)$
else if possible then
  Choose some triangle $a, b, c \in V$
  return WVC-NR-triangle$(G, a, b, c, k)$
else
  return WVC-NR-notriangles$(G, k)$
end if
```

Algorithm 27 A subroutine for WVC-NR, called WVC-NR-deg1

Input(s): a graph $G = (V, E)$ with real vertex weights $\omega : V \to \mathbb{R}$ with $\min \omega(V) \geq 1$ of maximum degree three without "easy components", a degree one vertex x with neighbor a, a positive real k

Output(s): YES if there is a vertex cover $C \subseteq V$, $\omega(C) \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of weight at most k exists.

```
    if $\omega(x) \geq \omega(a)$ then
      return WVC-NR($G - a, k - \omega(a)$)
    else if deg($a$) = 2 then
      {Let $y$ denote the first vertex on the path starting with $x, a, \ldots$ with
      deg($y$) $\neq 2$; due to step ($*$), deg($y$) $> 2$.}
 5:   WVC-NR-branch at $y$
    else
      {deg($a$) = 3}
      if at least one of $a$'s neighbors (say $y$) has degree three then
        WVC-NR-branch at $y$
10:   else if $a$'s other neighbors $y$ and $y'$ have degree two then
        {Let $N(y) = \{a, z\}$ and $N(y') = \{a, z'\}$.}
        if $\omega(a) \geq 2$ OR $\omega(y) \geq \omega(z)$ OR $\omega(y') \geq \omega(z')$ then
          WVC-NR-branch at $a$
        else
15:       {$\omega(y) < \omega(z)$ AND $\omega(y') < \omega(z')$}
          WVC-NR-branch at $x$
        end if
      else
        {One of $a$'s other neighbors, say $y$, has degree two and one, say $y'$,
        has degree one. This case is similar to deg($a$) = 2 and is hence
        omitted.}
20:   end if
    end if
```

Algorithm 28 A subroutine for WVC-NR, called WVC-NR-triangle

Input(s): a graph $G = (V, E)$ with real vertex weights $\omega : V \to \mathbb{R}$ with $\min \omega(V) \geq 1$ of maximum degree three without "easy components", with a triangle a, b, c, a positive real k

Output(s): YES if there is a vertex cover $C \subseteq V$, $\omega(C) \leq k$, (and it will implicitly produce such a small cover then) or NO if no vertex cover of weight at most k exists.

```
    {Let (w.l.o.g.) $\deg(a) \leq \deg(b) \leq \deg(c) = 3$.}
    if $\deg(b) = 2$ then
      {one vertex in the triangle has degree three}
      if $\omega(a) \leq \omega(b)$ then
5:      return WVC-NR$(G - N(a), k - \omega(N(a)))$
      else
        return WVC-NR$(G - N(b), k - \omega(N(b)))$
      end if
    else if $\deg(a) = 2$ then
10:   {two vertices in the triangle have degree two}
      WVC-NR-branch at $b$
    else
      {all vertices in the triangle have degree two}
      {assume, w.l.o.g., $\omega(a) \leq \omega(b) \leq \omega(c)$}
15:   WVC-NR-branch at $a$
    end if
```

Algorithm 29 A subroutine for WVC-NR, called WVC-NR-notriangles

Input(s): a graph $G = (V, E)$ with real vertex weights $\omega : V \to \mathbb{R}$ with $\min \omega(V) \geq 1$ of maximum degree three without "easy components", without triangles, a positive real k

Output(s): YES if there is a vertex cover $C \subseteq V$, $\omega(C) \leq k$, (and it will implicitly produce such a small cover then) or
NO if no vertex cover of weight at most k exists.

```
    if possible then
      Choose some vertex $x$ with $\omega(x) \geq 2$.
      if $\deg(x) = 3$ then
        WVC-NR-branch at $x$
5:    else
        $\{\deg(x) = 2$; let $N(x) = \{y, y'\}$ with $\deg(y) \leq \deg(y')\}$
        WVC-NR-branch at $y'$
      end if
    else
10:   {All vertices have weight less than two.}
      if possible then
        Choose two neighbors $a$ and $b$ with $\deg(a) = \deg(b) = 2$.
        {Let $x$ and $y$ be the two endpoints of a path $a$ and $b$ are part of such
        that $\deg(x), \deg(y) \geq 3$}
        if $x \in N[y]$ then
15:       WVC-NR-branch at $x$
        else
          if WVC-NR$(G - \{x, y\}, k - \omega(\{x, y\}))$ then
            return YES
          else if WVC-NR$(G - (\{x\} \cup N(y)), k - \omega(\{x\} \cup N(y)))$ then
20:         return YES
          else
            return WVC-NR$(G - N(x), k - \omega(N(x)))$
          end if
        end if
25:   end if
      return WVC-NR-notriangles-ctd$(G, k)$
    end if
```

Algorithm 30 A subroutine for WVC-NR-notriangles, called WVC-NR-notriangles-ctd

```
if possible then
  Choose three vertices $x, y, z$ with $\deg(x) = \deg(y) = \deg(z) = 3$ and two vertices
  $a, b$ with $N(a) = \{x, z\}$ and $N(b) = \{y, z\}$.
  if WVC-NR($G - \{x, y, z\}, k - \omega(\{x, y, z\})$) then
    return YES
  else if WVC-NR($G - (\{x\} \cup N(y)), k - \omega(\{x\} \cup N(y))$) then
    return YES
  else
    return WVC-NR($G - N(x), k - \omega(N(x))$)
  end if
else if possible then
  Choose a vertex $x$ with $\deg(x) = 3$ and a vertex $a \in N(x)$ with $\deg(a) = 2$ and
  $\omega(a) \geq \omega(x)$.
  WVC-NR-branch at $x$
else
  {We can find the following situation in $G$: vertices $x, y, z$ with $\deg(x) = \deg(y) =$
  $\deg(z) = 3$ and $y \in N(z)$, a vertex $a$ with $N(a) = \{x, z\}$ and $N(z) = \{a, b, y\}$.}
  if $\deg(b) = 2$ then
    {Let $N(b) = \{c, z\}$.}
    if $c = x$ then
      if $y \in N(x)$ then
        WVC-NR-branch at $y$
      else
        if WVC-NR($G - \{x, y\}, k - \omega(\{x, y\})$) then
          return YES
        else if WVC-NR($G - (\{y\} \cup N(x)), k - \omega(\{y\} \cup N(x))$) then
          return YES
        else
          return WVC-NR($G - N(y), k - \omega(N(y))$)
        end if
      end if
    else
      if WVC-NR($G - \{x, y, z, c\}, k - \omega(\{x, y, z, c\})$) then
        return YES
      else if WVC-NR($G - (N(y)), k - \omega(N(y))$) then
        return YES
      else
        return WVC-NR($G - N(z), k - \omega(N(z))$)
      end if
    end if
  else
    if WVC-NR($G - \{a, b, y\}, k - \omega(\{a, b, y\})$) then
      return YES
    else if WVC-NR($G - (\{b\} \cup N(y)), k - \omega(\{b\} \cup N(y))$) then
      return YES
    else
      return WVC-NR($G - N(b), k - \omega(N(b))$)
    end if
  end if
end if
```

5.3 Improved branching: simple algorithms

The aim of this section is to develop *very simple* search tree algorithms. Due to their simplicity, these algorithms are easily seen to be working correctly. Their time analysis, however, often involves either some thorough mathematical background knowledge or a possible tough mathematical analysis.

If an involved time analysis is necessary, the algorithm development often proceeds in a top-down rather than bottom-up manner. This means that the case analysis is guided by the need of the time analysis rather than by a collection of observations concerning local situations.

However, sometimes improved time analysis and actually slightly changing (tuning) algorithms goes hand in hand. This is the actual topic of our first example.

5.3.1 PLANAR INDEPENDENT SET

Recall that the greedy kernelization algorithm 6 was based on the well-known fact that each planar graph has at least one vertex of degree at most five. This result can be also used to create a very simple search tree algorithm for PLANAR INDEPENDENT SET, see Alg. 31.

The mentioned result on planar graphs shows that for any v that is picked by the algorithm, $|N[v]| \leq 6$, so that we can state:

Proposition 5.11 *Alg. 31 solves* PLANAR INDEPENDENT SET *in time* $\mathcal{O}(6^k|G|)$.

Fortunately, some mathematicians gave a deeper analysis of the structure of planar graphs regarding vertices of small degree. In [6, Theorem 2], the authors have shown the following result:

Theorem 5.12 *Every connected plane graph with at least two vertices has*

1. *two vertices with degree sum at most 5, or*
2. *two vertices of distance at most two and with degree sum at most 7, or*
3. *a triangular face with two incident vertices with degree sum at most 9, or*
4. *two triangular faces neighbored via an edge $\{u, v\}$ where the sum of the degrees of u and v is at most 11.*

Based on that theorem, we propose Algorithm 32:

Algorithm 31 A simple search tree algorithm for PLANAR INDEPENDENT SET, called PIS-ST-simple

Input(s): planar graph $G = (V, E)$, positive integer k

Output(s): YES if G has an independent set I with $|I| = k$; NO otherwise

```
if $k \leq 0$ then
  return YES
else if $V = \emptyset$ then
  return NO
else
  Let $v$ be a vertex of lowest degree in $G$.
  {One vertex from $N[v]$ will be in any maximal independent set.}
  for all $x \in N[v]$ do
    $V' := N[x]$
    if PIS-ST-simple($G[V \setminus V'], k-1$) then
      return YES
    end if
  end for
  {All trials have failed.}
  return NO
end if
```

The correctness of the algorithm immediately follows from the quoted theorem. The running time T of the new algorithm, measured in terms of the number of leaves of the search tree, satisfies

$$T(k) \leq 4T(k-1) + 6T(k-2).$$

This recurrence can be solved, showing $T(k) \leq 5.1623^k$ as claimed.

This shows Proposition 5.13.

Proposition 5.13 *Alg. 32 solves* PLANAR INDEPENDENT SET *within time* $\mathcal{O}(5.1623^k|G|)$.

Unfortunately, we have to remark in this place that this is not the best algorithm for PLANAR INDEPENDENT SET that we are aware of. As mentioned below in Chap. 10, there are very cute algorithms for solving (general) MAXIMUM INDEPENDENT SET that run in time approximately $\mathcal{O}(1.2^n)$. Since n can be limited to $4k$ (see Alg. 7), a combination of these algorithms would outperform the (admittedly still simple) algorithm that we presented here. However, note that even the full-fledged Alg. 32 is still close to trivial in comparison with the quite sophisticated algorithms that should be implemented to enjoy the better running times.

Algorithm 32 A more advanced search tree algorithm for PLANAR INDEPENDENT SET, called PIS-ST

Input(s): planar graph $G = (V, E)$, positive integer k
Output(s): YES if G has an independent set I with $|I| = k$; NO otherwise

```
    if $k \leq 0$ then
      return YES
    else if $V = \emptyset$ then
      return NO
5:  else
      Let $v$ be a vertex of lowest degree in $G$.
      {One vertex from $N[v]$ will be in any maximal independent set.}
      if $\deg(v) \leq 4$ then
        for all $x \in N[v]$ do
10:       $V' := N[x]$
          if PIS-ST($G[V \setminus V'], k-1$) then
            return YES
          end if
        end for
15:   else
        {The fourth case of Theorem 5.12 applies.}
        {Hence: $N(v) = \{x, y, u, v_1, v_2\}$
        such that $N(u) = \{x, y, v, u_1, u_2, u_3\}$}
        if PIS-ST($G[V \setminus N[v]], k-1$) then
          {Branch 1: $v$ in IS ?}
20:       return YES;
        else if PIS-ST($G[V \setminus N[u]], k-1$) then
          {Branch 2: $v$ is not in IS, but $u$ ?}
          return YES;
        else if PIS-ST($G[V \setminus N[x]], k-1$) then
25:       {Branch 3: $v, u$ is not in IS, but $x$ ?}
          return YES;
        else if PIS-ST($G[V \setminus N[y]], k-1$) then
          {Branch 4: $v, u, x$ is not in IS, but $y$ ?}
          return YES;
30:     else
          {Now $v, u, x, y$ is not in IS, so one of the $v_i$ and one of the $u_j$ must
          be in a maximal independent set.}
          for all $i = 1, 2; j = 1, 2, 3$ do
            if PIS-ST($G[V \setminus (N[v_i] \cup N[u_j])], k-2$) then
              return YES
35:         end if
          end for
          return NO
        end if
      end if
40: end if
```

The next subsection is devoted to PLANAR DOMINATING SET. However, there is one kind of intermediate problem, namely PLANAR INDEPENDENT DOMINATING SET:

Problem name: PLANAR INDEPENDENT DOMINATING SET (PIDS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *independent dominating set* $D \subseteq V$ with $|D| \leq k$?

Clearly, this problem can be likewise called MINIMUM MAXIMAL INDEPENDENT SET, restricted to planar graphs. Now, observe that the search tree algorithms we presented for PLANAR INDEPENDENT SET never actually rely on the fact that we actually look for a maximum independent set, since all branchings are based on the idea of finding a maximal independent set. We can therefore conclude:

Proposition 5.14 PLANAR INDEPENDENT DOMINATING SET *can be solved in time* $\mathcal{O}(5.1623^k|G|)$.

Proof. Alg. 32 would have to be modified: the first four lines of the code should be replaced by the following four lines:

```
if $k < 0$ then
  return NO
else if $V = \emptyset$ then
  return YES
else
  ...
end if
```

where the ... contains the actual branching program. $k < 0$ means that there is no small independent dominating set. If $k \geq 0$ and $V = \emptyset$, we have found an independent dominating set. Otherwise, there exists a vertex on which we can branch. ∎

This bound on the running time is actually better than the one of the alternative algorithm that looks amongst the dominating sets produced with the algorithm presented in the next subsection for a dominating set that happens to be independent. Moreover, the previous comments on producing better algorithms by relying on exact algorithms for MAXIMUM INDEPENDENT SET do not apply here.

Special graph classes that allow polynomial-time algorithms for INDEPENDENT DOMINATING SET are considered in [55].

5.3.2 PLANAR DOMINATING SET

The following exposition follows [11].

In the case of DOMINATING SET, the situation seems more intricate than with INDEPENDENT SET or INDEPENDENT DOMINATING SET, where we could simply branch at a low-degree vertex u (as existing in planar graphs). Clearly, again, either u or one of its neighbors can be chosen to be in an optimal dominating set. However, removing u from the graph leaves all its neighbors being already dominated, but still also being suitable candidates for an optimal dominating set. This consideration leads us to formulate our search tree procedure in a more general setting, where there are two kinds of vertices in our graph.

We stress this fact by partitioning the vertex set V of G into two disjoint sets B and W of *black* and *white* vertices, respectively, i.e., $V = B \uplus W$, where $\uplus$ denotes disjoint set union. We will also call this kind of graph a *black and white graph.*

Problem name: ANNOTATED DOMINATING SET (ADS)
Given: A black and white graph $G = (B \uplus W, E)$
Parameter: a positive integer k
Output: Is there a choice of at most k vertices $V' \subseteq V = B \uplus W$ such that, for every vertex $u \in B$, there is a vertex $u' \in N[u] \cap V'$? In other words, is there a set of at most k vertices (which may be either black or white) that dominates the set of black vertices?

In each step of the search tree, we would like to branch according to a low degree black vertex. This would work fine if we deal with a graph class, so that we can guarantee the existence of a vertex $u \in B \uplus W$ with $\deg(u) \leq d$. However, as long as *not all* vertices have degree bounded by d (as, e.g., the case for graphs of bounded genus g, where only *the existence* of a vertex of degree at most d_g is known), this vertex need not necessarily be black.

We consider the following reduction rules for simplifying ANNOTATED DOMINATING SET the on planar graphs. In developing the search tree, we will always assume that we are branching from a "reduced instance;" thus, we are constantly simplifying the instance according to the reduction rules given below (the details will be explained later). When a vertex u is placed in the dominating set D by a reduction rule, then the target size k for D is reduced to $k - 1$ and the neighbors of u are whitened.

(R1) Delete an edge between white vertices.

(R2) Delete a pendant white vertex, i.e., a vertex of degree one.

(R3) If there is a pendant black vertex w with neighbor u (either black or white), then delete w, place u in the dominating set, and lower k to $k-1$.

(R4) If there is a white vertex u of degree 2, with two black neighbors u_1 and u_2 connected by an edge $\{u_1, u_2\}$, then delete u.

(R5) If there is a white vertex u of degree 2, with black neighbors u_1, u_3, and there is a black vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G, then delete u.

(R6) If there is a white vertex u of degree 2, with black neighbors u_1, u_3, and there is a white vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G, then delete u.

(R7) If there is a white vertex u of degree 3, with black neighbors u_1, u_2, u_3 for which the edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ are present in G (and possibly also $\{u_1, u_3\}$), then delete u.

Lemma 5.15 *The reduction rules are sound.*

The proof is contained in the journal version of the quoted paper.

We say that that G is a *reduced* graph if none of the above reduction rules can be applied to G. If none of the rules (R1), (R2), (R4)–(R7) are applicable to G, we term G *nearly reduced.*

Let $H := G[B]$ denote the (plane embedded) subgraph of G induced by the black vertices. Let F denote the set of faces of H. Say that a face $f \in F$ is *empty* if, in the plane embedding of G, it does not contain any white vertices.

Lemma 5.16 *Let $G = (B \uplus W, E)$ be a plane black and white graph. If G is (nearly) reduced, then the white vertices form an independent set and every triangular face of $G[B]$ is empty.*

Proof. The result easily follows from the reduction rules (R1), (R2), (R4), and (R7). ∎

Let us introduce a further notion which is important in order to bound the running time of our reduction algorithm.[2] To this end, we introduce the following variants of reduction rules (R5) and (R6):

[2]In the conference version of this paper, it was stated that a graph can be reduced with respect to all rules in linear time. Thanks to Torben Hagerup (Augsburg) who pointed out a gap in our earlier argument and suggested the fix we here employ.

(R5′) If there is a white vertex u of degree 2, with black neighbors u_1, u_3 such that u_1 *has at most seven neighbors that have degree at least 4*, and there exists a black vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G, then delete u.

(R6′) If there is a white vertex u of degree 2, with black neighbors u_1, u_3 such that u_1 *has at most seven neighbors that have degree at least 4*, and there exists a white vertex u_2 and edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ in G, then delete u.

We say that that G is a *cautiously reduced* graph if (R1), (R2), (R4), (R5′), (R6′), and (R7) cannot be applied anymore to G. Observe that Lemma 5.16 is also valid for cautiously reduced graphs.

Lemma 5.17 *Applying reduction rules (R1), (R2), (R4), (R5′), (R6′), and (R7), a given planar black and white graph $G = (B \uplus W, E)$ can be transformed into a cautiously reduced graph $G' = (B' \uplus W', E')$ in time $\mathcal{O}(n)$, where n is the number of vertices in G.*

Again, we skip details of the proof and refer to the journal version.

The main theorem for proving the correctness and running time of the simple branching algorithm 33 is the following one.

Theorem 5.18 *If $G = (B \uplus W, E)$ is a planar black and white graph that is nearly reduced, then there exists a black vertex $u \in B$ with $\deg_G(u) \leq 7$.*

Since the proof of Theorem 5.18 is very technical, we will omit most parts of it and basically only give an overview of it. In Lemma 5.19, we specialize Euler's well-known formula for planar graphs to planar black and white graphs. This is a core tool within the proof of Theorem 5.18, which is done by contradiction. Lemma 5.20 sets up some additional information used in the proof of Theorem 5.18. For the involved technical details, we refer to our (journal) paper on the material.

The following technical lemma, based on an "Euler argument," will be needed.

Lemma 5.19 *Suppose $G = (B \uplus W, E)$ is a connected plane black and white graph with b black vertices, w white vertices, and e edges. Let the subgraph induced by the black vertices be denoted $H = G[B]$. Let c_H denote the number of connected components of H and let f_H denote the number of faces of H. Let*

$$z = \big(3(b + w) - 6\big) - e \tag{5.2}$$

measure the extent to which G fails to be a triangulation of the plane. If the criterion

$$3w - 4b - z + f_H - c_H < 7 \tag{5.3}$$

is satisfied, then there exists a black vertex $u \in B$ with $\deg_G(u) \leq 7$.

Proof. Let the (total) numbers of vertices, edges, and faces of G be denoted v, e, f, respectively. Let e_{bw} be the number of edges in G between black and white, and let e_{bb} denote the number of edges between black and black. With this notation, we have the following relationships.

$$\begin{aligned}
v - e + f &= 2 && \text{(Euler formula for } G\text{)} && (5.4)\\
v &= b + w && && (5.5)\\
e &= e_{bb} + e_{bw} && && (5.6)\\
b - e_{bb} + f_H &= 1 + c_H && \text{((extended) Euler formula for } H\text{)} && (5.7)\\
2v - 4 - z &= f && \text{(by Eq. (5.2), (5.4), and (5.5))} && (5.8)
\end{aligned}$$

If the lemma were false, then the minimum degree would be at least eight. Hence, we would have, using (5.6),

$$8b \;\leq\; 2e_{bb} + e_{bw} \;=\; e_{bb} + e. \tag{5.9}$$

We will assume this and derive a contradiction. The following inequality holds:

$$\begin{aligned}
3 + c_H &= v + b - (e_{bb} + e) + f + f_H && \text{(by (5.4) and (5.7))}\\
&\leq v + b - 8b + f + f_H && \text{(by (5.9))}\\
&= 3v - 7b + f_H - 4 - z && \text{(by (5.8))}\\
&= 3w - 4b + f_H - 4 - z. && \text{(by (5.5))}
\end{aligned}$$

This yields a contradiction to (5.3). ∎

We will prove Theorem 5.18 by contradiction. The reduction rules give us additional helpful properties of an assumed counterexample. This is stated in the following lemma.

Lemma 5.20 *If there is any counterexample to Theorem 5.18, then there is a connected counterexample where $\deg_G(u) \geq 3$ for all $u \in W$.*

In order to be able to conclude our stated running times, we in fact need a corollary of Theorem 5.18 first:

Corollary 5.21 *Let G be a cautiously reduced planar black and white graph. Then, G contains a black vertex of degree at most 7.*

Proof. Let G' be the graph obtained when reducing G further with respect to all reduction rules (R1)–(R7). In particular, each connected component of G' is nearly reduced. Hence, there exists a black vertex v with $\deg_{G'}(v) \leq 7$ (in one such component). The only difference between G' and G is that G may contain white vertices of degree two where both neighbors have more than seven neighbors that are of degree at least 4. We argue that $\deg_G(v) \leq 7$. If this were not the case, then v must have additional neighbors which are not present in G'. By the above observation an additional neighbor must be a white vertex u of degree two where both neighbors (in particular, the neighbor v) have more than seven neighbors that are of degree at least 4. Hence, there exist vertices $v_1, \ldots, v_\ell \in N_G(v)$ ($\ell \geq 8$) which are of degree at least 4. Since these vertices are not removed by any of the reduction rules, it follows that $v_1, \ldots, v_\ell \in N_{G'}(v)$ which implies $\deg_{G'}(v) > 7$, a contradiction. ∎

Theorem 5.22 ANNOTATED DOMINATING SET *on planar graphs can be solved in $\mathcal{O}(8^k n)$ time.*

Proof. Use Corollary 5.21 for the construction of a search tree as sketched in the beginning of the subsection. Algorithm 33 is initiated with the call PDS-ST$(V, \emptyset, E, k)$, where $((V, E), k)$ is the given planar graph instance.

Note that performing the reduction in each node of the search tree, by Lemma 5.17, can be done in time $\mathcal{O}(n)$. Moreover, it would be also possible to incorporate reduction rule (R3) to avoid further recursive calls; the time analysis is valid in this case, as well. ∎

Alternatively, using a reduction to a linear size problem kernel for PLANAR DOMINATING SET shown in [12, 82] (also see Chap. 4, we obtain the following result.

Theorem 5.23 PLANAR DOMINATING SET *can be solved in $\mathcal{O}(8^k k + n^3)$ time.*

Let us also remark that, albeit this result generalizes to graphs of bounded genus [154], further generalizations are unlikely, as examined in [87].

Let us now briefly discuss the following variant of DOMINATING SET, namely ROMAN DOMINATION, restricted to planar graphs:

Problem name: PLANAR ROMAN DOMINATION (pROMAN)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *Roman domination* function R such that $\sum_{x \in V} R(x) \leq k$?

Algorithm 33 A simple search tree algorithm for PLANAR DOMINATING SET, called PDS-ST

Input(s): planar black and white graph $G = (B, W, E)$, positive integer k
Output(s): YES if G has a dominating set D with $|D| \leq k$; NO otherwise

```
Exhaustively apply "cautious reduction rules" to $(B, W, k)$;
if $k < 0$ then
  return NO
else if $k = 0$ then
  return $B = \emptyset$
else
  $\{k > 0\}$
  pick some black vertex $v$ of minimum degree;
  $B' := B \cap N[v]$;
  $W' := W \cap N[v]$;
  for all $v' \in B'$ do
    $E' := \{\{u, v'\} \mid u \in B \cup W\}$;
    if PDS-ST$(B \setminus N[v'], W \cup N(v'), E \setminus E', k - 1)$ then
      return YES
    end if
  end for
  for all $v' \in W'$ do
    $E' := \{\{u, v'\} \mid u \in B \cup W\}$;
    if PDS-ST$(B \setminus N(v'), (W \setminus \{v'\}) \cup N(v'), E \setminus E', k - 1)$ then
      return YES
    end if
  end for
  return NO
end if
```

Here, a *Roman domination* of a graph $G = (V, E)$ is a function $R : V \to \{0, 1, 2\}$ such that

$$\forall v \in V : R(v) = 0 \Rightarrow \exists x \in N(v) : R(x) = 2.$$

$D_R = R^{-1}(\{1, 2\})$ is then the *Roman domination set.* The minimum $\sum_{x \in V} R(x) = \sum_{x \in D_R} R(x) =: R(V)$ over all valid Roman domination functions is also called the *Roman domination number* of a given graph.

This problem comes with an interesting history dating back to the location of armies in the times of Emperor Constantine. This is nicely explained in [Chapter 11, Site 17]; the journal version being [334]. This problem became popular by a note published in the *Scientific American* [356].

A nice overview on problems related to Roman domination can be found in [41, 141]. We assume that solving algorithms similar to the ones presented in this Habilitationsschrift can be also found for most of these variants, in particular regarding multi-attack variants [231].

We show how to solve PLANAR ROMAN DOMINATION with our reasoning about DOMINATING SET. Namely, a specific vertex x of low degree (as would be picked by Alg. 33) could be dominated in the following ways:

- The Roman domination function assigns 2 to $y \in N[x]$. This case is treated completely analogous to the case that puts y into the dominating set in the problem PLANAR DOMINATING SET described above.
- The Roman domination function (that is constructed along the search tree) assigns 1 to x. Then, x (together with its incident vertices) is deleted in the next recursive call.

This is a complete case distinction that leads to Alg. 34.

There is one point that still needs justification: are the reduction rules still sound for this modified problem? As the reader can observe, this is in fact the case for all rules but (R3), since the basic argument for showing correctness of deleting that particular white vertex always is that it is never worse to put another black (in one case also white) vertex into the dominating set (i.e., in this case, assign 2 to that vertex via the Roman domination function). In fact, deleting black vertices is not that easy, since it might be cheaper to put one army onto that vertex instead of putting two armies onto the neighboring vertex. Fortunately, observe that we actually don't use rule (R3) in the analysis for PLANAR DOMINATING SET. Consequently, the "Roman domination rules" exclude (R3). This justifies the (nice) branching that can be found in Alg. 34.

Let us analyze the running time of Alg. 34. Domination of a particular chosen vertex v can be accomplished in two ways:

- either the implicitly constructed Roman domination function R shows $R(v) = 1$; then the parameter is reduced by one,
- or R shows $R(u) = 2$ for some $u \in N[v]$; this branching reduces the parameter by two for each u; according to Thm. 5.18, $N[v]$ contains at most eight vertices.

Solving the corresponding recurrence $T(k) \leq T(k-1) + 8T(k-2)$ for the size of the search tree shows the following assertion:

Theorem 5.24 *Alg. 34 solves* PLANAR ROMAN DOMINATION *in* $\mathcal{O}(3.3723^k k + n^3)$ *time.*∎

Algorithm 34 A simple search tree algorithm for PLANAR ROMAN DOMINATION, called pROMAN-ST

Input(s): planar black and white graph $G = (B, W, E)$, positive integer k
Output(s): YES if G has a Roman domination function R with $|R(B \cup W)| \leq k$; NO otherwise

```
    Exhaustively apply “Roman domination reduction rules” to $(B, W, k)$;
    the resulting instance is also called $(B, W, k)$.
    if $k < 0$ then
      return NO
    else if $k = 0$ then
5:    return $B = \emptyset$
    else
      $\{k > 0\}$
      if there is a pendant black vertex $v$ with neighbor $u$ (either black or
      white) then
        if pROMAN-ST($B \setminus \{v\}, W, E \setminus \{e \in E \mid v \in e\}, k-1$) then
10:       return YES
        else
          return pROMAN-ST($B \setminus \{v, u\}, (W \setminus \{u\}) \cup (N(u) \setminus \{v\}), E \setminus \{e \in E \mid v \in e\}, k-2$)
        end if
      end if
15:   {Now, the instance is reduced w.r.t. all cautious reduction rules known
      for PDS.}
      pick some black vertex $v$ of minimum degree;
      if pROMAN-ST($B \setminus \{v\}, W, E \setminus \{e \in E \mid v \in e\}, k-1$) then
        return YES
      end if
20:   $B' := B \cap N[v]$;
      $W' := W \cap N[v]$;
      for all $v' \in B'$ do
        $E' := \{\{u, v'\} \mid u \in B \cup W\}$;
        if pROMAN-ST($B \setminus N[v'], W \cup N(v'), E \setminus E', k-2$) then
25:       return YES
        end if
      end for
      for all $v' \in W'$ do
        $E' := \{\{u, v'\} \mid u \in B \cup W\}$;
30:     if pROMAN-ST($B \setminus N(v'), (W \setminus \{v'\}) \cup N(v'), E \setminus E', k-2$) then
          return YES
        end if
      end for
      return NO
35: end if
```

Regarding running time, in the same spirit as discussed above with PLANAR DOMINATING SET, one should again replace (R5) and (R6) by their "cautious counterparts." We leave the obvious modifications to the reader.

5.3.3 HITTING SET

As observed in various places throughout this Habilitationsschrift, HITTING SET is a very important example, showing both the possibilities and limitations of the parameterized approach. Here, we focus on a top-down approach to develop search tree algorithms for d-HITTING SET, where d is fixed. The results presented in this section are based on mostly unpublished works, also see [175].

We are improving on Niedermeier and Rossmanith's results for d-HITTING SET, in particular for small d. Basically, we are doing a better analysis of a simple search tree algorithm with the guide of improved *heuristic priorities* and the use of *data reduction rules* within the search tree algorithm (analysis). In actual fact, the reduction rules we use have been already introduced in Chap. 2. The better analysis is based on the introduction of a second *auxiliary parameter,* a technique which can be useful in other analyses of parameterized algorithms, as we believe.

More precisely, we get the following table:

d	3	4	5	6	10	100
$T(k)$	2.18^k	3.12^k	4.08^k	5.05^k	9.02^k	99.0002^k

which favorably compares to the one obtained in [308], who obtained the following values:

d	3	4	5	6	10	100
$T(k)$	2.42^k	3.31^k	4.24^k	5.20^k	9.11^k	99.0101^k

The general formula is

$$\mathcal{O}\left(\left(\frac{d-1}{2}\left(1+\sqrt{1+\frac{4}{(d-1)^2}}\right)\right)^k + n\right),$$

since Niedermeier and Rossmanith also presented a simple linear-time kernelization, i.e., a preprocessing step typical for parameterized algorithms which leaves us with an instance of size $f(k)$ (instead of n). A smaller kernel was recently described in [310]. As can be seen, the exponential base tends to $d-1$ with growing d.

By an intricate case analysis of a comparatively complicated algorithm, they were able to arrive at an $\mathcal{O}(2.270^k + n)$ algorithm for 3-HITTING SET. This was improved in [175] to about $\mathcal{O}(2.179^k + n)$. We will also sketch this result below.

For the special case of 2-HITTING SET, likewise known as VERTEX COVER, in a kind of race (using more and more intricate case analysis) an $\mathcal{O}(1.285^k + n)$-algorithm [86] has been obtained. Our approach is not suitable to tackle that case.

Interestingly, a similar sort of analysis has been performed independently of the present work by Wahlström for the non-parameterized MINIMUM 3-HITTING SET problem [373]. Or if we like to put it into the parameterized framework, we can say that he used a different parameterization, namely by the number of vertices of the given MINIMUM 3-HITTING SET instance. More precisely, he basically used

- the same set of reduction rules and
- the same auxiliary parameter, but
- the mathematical way how to analyze the algorithms is different: while Wahlström is combining the auxiliary parameter with the ordinary parameter to get specific numbers as labels of the search tree (and hence the branching numbers), we will here derive explicit sets of recurrence equations.

We present a simplified version of his work in Chap. 10.

Simple algorithmics

Since each hyperedge must be covered, there exists a trivial $\mathcal{O}^*(d^k)$ algorithm for d-HITTING SET. The procedure HS-simple listed in Alg. 35 is initially called with arguments $(G, k, \emptyset)$, where (G, k) is the d-HITTING SET instance to be solved. The procedure returns a valid hitting set S of size at most k iff (G, k) is a YES-instance; otherwise, the procedure returns NO.

Obviously, the base of the exponential running time of this algorithm heavily depends on the necessary amount of branching. Observe that according to the problem specification, in a d-HITTING SET instance, there might be edges of degree *up to d*; in particular, there might be edges of smaller degree in the very beginning. "Small degree edges" may also be introduced later during the run of the algorithm, since the removal of a certain vertex x will reduce the edge degree of all edges incident with x. A natural heuristic would therefore first branch according to edges of small degree. We would therefore refine:

Algorithm 35 A simple search tree algorithm for d-HS, called HS-simple

Input(s): G, k: input instance,
S: partial solution constructed so far along the branch
Output(s): a valid hitting set S of size at most k iff (G, k) is a YES-instance; otherwise, the procedure returns NO.

```
if $k > 0$ AND $G$ has some edges then
  choose some edge $e$; {to be refined}
  $S' := \emptyset$; {solution to be constructed}
  for all $x \in e$ do
    $G' := (V \setminus \{x\}, \{e \in E \mid x \notin e\})$;
    $S' :=$ HS-simple$(G', k-1, S \cup \{x\})$
    if $S' \neq$ NO then
      return $S'$
    end if
  end for
  return NO
else if $G$ has no edges and $k \geq 0$ then
  return $S$
else
  return NO
end if
```

```
HS-simple-heuristic$(G, k, S)$:
if $k > 0$ AND $G$ has some edges then
  choose some edge $e$ of smallest degree;
  ...{as before}
end if
```

In fact, as a further technical detail, we shall use the following binary version of the latter algorithm; details of what is meant by reduction rules are explained in Chap. 2; all heuristic priorities are later exhibited, one of them—preferring small edges—has been already shown.

Lemma 5.25 *If the reduction rules and the procedure solving simple instances are correct, then HS-binary($G, k, \emptyset$) either returns a correct solution to the instance (G, k) or it returns* NO*, i.e., (G, k) has no solution.*

Proof. Formally, the proof is done by induction on the number of vertices of the graph. The base is solved by rules for simple instances. Assume that the correctness of the algorithm has been shown for all graphs up to n

Algorithm 36 A refined binary search tree algorithm for d-HS, called HS-binary

```
Input(s): $G, k$: input instance,
  $S$: partial solution constructed so far along the branch
Output(s): a valid hitting set $S$ of size at most $k$ iff $(G, k)$ is a YES-instance;
  otherwise, the procedure returns NO.

  Exhaustively apply the reduction rules.
  {call the resulting instance $(G, k)$ and the intermediate solution $S$}
  if $(G, k)$ is a simple instance then
    solve in polynomial time and return solution accordingly
  else
    choose some edge $e$ and some $x \in e$ according to heuristic priorities
    $S' := \emptyset$; {solution to be constructed}
    $G' := (V \setminus \{x\}, \{e \in E \mid x \notin e\})$;
    $S' :=$ HS-binary$(G', k-1, S \cup \{x\})$;
    if $S' =$ NO then
      {try $x$ not in solution}
      $G' = (V \setminus \{x\}, \{e \setminus x \mid e \in E\})$;
      $S' :=$ HS-binary$(G', k, S)$
    end if
    return $S'$
  end if
```

vertices. Consider now an instance (G, k) where G has $n+1$ vertices. If after applying the reduction rules, G has n vertices, the correctness follows by the induction hypothesis. Otherwise, let us call the resulting instance (G, k), as well. Possibly, (G, k) can be correctly solved by rules for simple instances. If not, we encounter a binary branching, covering two mutually exclusive cases for some chosen vertex x:

1. If x is taken into the hitting set, then in the recursive call the following instance is created:

   - x is removed from the vertex set; hence, the induction hypothesis is applicable to the created instance.
   - The parameter k is accordingly decremented.
   - x is put into the hitting set S which is going to be recursively constructed.
   - All hyperedges to which x belongs are covered and hence deleted.

2. If x is not put into the hitting set, then in the recursive call the following instance is created:
   - x is removed from the vertex set; hence, the induction hypothesis is applicable to the created instance.
   - The hitting set S which is going to be recursively constructed is not changed.
   - The parameter k is not changed.
   - From all hyperedges to which x belongs, x is removed.

The second branch is only executed when the first branch returns the empty set. This is correct, since x will be in each correct cover that is returned in the first case, and hence if the empty set is returned, this clearly signals that the corresponding branch shows no solution.

Since the described actions are obviously correct, the correctness of the algorithm follows by induction. ∎

Remark 5.26 *We haven't been specific about what we mean by "simple instances" in our search tree algorithm. Notice that although our reduction rules (as listed in Chap. 2) already guarantee that all vertices in a reduced instance have minimum degree of two (see Lemma 2.20), we could even suppose more:* HITTING SET *in hypergraphs of degree upperbounded by two is obviously equivalent to* EDGE COVER *(which is the natural edge-analog to* VERTEX COVER *in graphs).* EDGE COVER *was probably first observed to be computable in polynomial time by Norman and Rabin [311]. Already there, the connection to matching techniques was drawn. Hence,* HITTING SET, *restricted to hypergraphs of vertex degree of at most two, can be efficiently solved. Therefore, we can even assume that all instance that we ever branch on will have minimum degree of three. We will only exploit this when actually analyzing* 3-HITTING SET *in more details below.*

Heuristic priorities

We will base the analysis on several *heuristic priorities*. Let $G = (V, E)$ be a reduced instance of d-HITTING SET.

$P1$ *Prefer small edges.* More formally, let $\delta_0 = \min\{|e| \mid e \in E\}$ and

$$V_0 = \{v \in V \mid \exists e \in E : (v \in e) \wedge (|e| = \delta_0)\}.$$

$P2$ Choose some $x \in V_0$ of *maximum degree in* $G[V_0]$.

$P3$ Resolve possible nondeterminism in $P2$ by choosing among the vertices of maximum degree in $G[V_0]$ a vertex x that has *minimum degree in* $G[V_1]$, where

$$V_1 = \{v \in V \mid \exists e \in E : (v \in e) \wedge (|e| < d)\}.$$

How to analyze the our d-HITTING SET algorithm

The idea of making favorable branches first has also another bearing, this time on the way we are going to analyze our algorithm.

Let $T_d^\ell(k)$, $\ell \geq 0$ denote the *size* (more precisely, the number of leaves) of the search tree when assuming that exactly ℓ edges in the given instance (with parameter k) have a degree of (at most) $d-1$. So, ℓ is the *auxiliary parameter* in our analysis. Sometimes, we will shortcut our discussions by using $T_d^{\geq \ell}(k)$, denoting the situation of a search tree assuming that at least ℓ edges in the given instance (with parameter k) have a degree of (at most) $(d-1)$. The intuition is that, e.g., in the case of 3-HITTING SET, $T_3^4(k)$ would describe a situation which is "more like" 2-HITTING SET than $T_3^3(k)$. Therefore, we can conclude: $T_d^{\geq \ell}(k) \leq T_d^\ell(k)$. Regarding an upperbound on the size $T_d(k)$ of the search tree of the whole problem, we can estimate $T_d(k) \leq T_d^0(k)$, since the worst case is that we have no edges of low degree. In the following, we are exhibiting mutually recursive relationships between different $T_d^\ell(k)$; solving these recursions will yield the bounds on the size of the search tree and hence on the running time.

Lemma 5.27 $T_d^0(k) \leq T_d^0(k-1) + T_d^2(k)$.

Proof. Due to Lemma 2.20, the instance G contains (immediately before the branching) a vertex x of degree 2 (or larger). One branch is that x is put into the cover. Otherwise, two or more edges of degree $(d-1)$ are created. ∎

Lemma 5.28 $T_d^1(k) \leq T_d^0(k-1) + T_d^1(k-1) + T_d^2(k-1) + \cdots + T_d^{d-2}(k-1)$.

Proof. Due to Lemma 2.20, all vertices have degree at least two. If the chosen $x \in e$ is not put into the cover, then at least one new edge of degree $(d-1)$ is created besides the edge $e' = e \setminus \{x\}$ of degree $(d-2)$. According to the heuristic priorities, we would continue branching at some vertex from e'. The argument then repeats, yielding the formula as claimed, since every time a new edge of degree $(d-1)$ is created: otherwise, the reduction rules would trigger and reduce the number of branches, enabling an even better analysis in that case. ∎

When plugging in $T_d^\ell \leq T_d^1$ for $\ell \geq 1$, we obtain the following recurrences:

$$\begin{array}{rcl} T_d^0(k) & \leq & T_d^0(k-1) + (d-1)T_d^1(k-1) \\ T_d^1(k) & \leq & T_d^0(k-1) + (d-2)T_d^1(k-1) \end{array}$$

These are exactly the recurrences derived in [308], which immediately entails that we can only beat their results with our analysis. Observe that up to this point the degree heuristics $P2$ and $P3$ did not come into play.

An analysis of T_d^2

In the following discussion, let e_1 and e_2 be two edges of degree $d-1$. Moreover, since we are only analyzing situations up to T_d^2, Lemma 5.28 implies the following relation:

$$T_d^1(k) \leq T_d^0(k-1) + T_d^1(k-1) + (d-3)T_d^2(k-1) \tag{5.10}$$

Case $e_1 \cap e_2 = \emptyset$:

Lemma 5.29 *If $e_1 \cap e_2 = \emptyset$, then we can estimate*

$$T_d^2(k) \leq T_d^1(k-1) + (d-2)T_d^2(k-1).$$

Proof. By edge domination, each small edge that is created along the way when branching on say e_1 is in fact new. ∎

In the following analysis, as justified in [175], we will treat all derived inequalities as if they were equations. Lemma 5.29 then provides an expression for T_d^1 in terms of T_d^0 and T_d^2, and Lemma 5.27 allows to express T_d^2 in terms of T_d^0, so that Eq. (5.10) gives, after some algebra, the following equation that T_d^0 should satisfy:

$$0 = T_d^0(k+1) - dT_d^0(k) + (d-1)T_d^0(k-1) - T_d^0(k-2).$$

Exact solutions to such recurrence relations would entail sums of exponential functions. Since we are only interested in the growth rate, we can solve such equations by the ansatz $T_d^0(k) \approx c_d^k$. Some computations give the following table:

d	3	4	5	6	10	100
$T_d^0(k) \approx$	2.3248^k	3.1479^k	4.0780^k	5.0490^k	9.0139^k	99.0002^k

(5.11)

Comparing with the figures listed above, we already see some considerable improvements for this case, so that a further analysis seems worthwhile doing. However, in the case $d = 3$, this approach is not matching the values obtained by Niedermeier and Rossmanith with their (intricate) algorithm; how to obtain a significantly better algorithm is detailed in [175] and some ideas will be given below.

Case $e_1 \cap e_2 = \{x\}$:

Hence, $e_i = \{x, x_i^1, \dots, x_i^{d-2}\}$.

Lemma 5.30 *If $e_1 \cap e_2 = \{x\}$ and if $d \geq 4$, then we can estimate*

$$T_d^2(k) \leq T_d^0(k-1) + T_d^0(k-2) + (d-1)T_d^1(k-2) + (d^2 - 5d + 4)T_d^2(k-2).$$

Proof. Due to $P2$, we will branch at x. Taking x into the cover is a $T_d^0(k-1)$-branch. Now, notice that at the start of every branch, we are dealing with a reduced instance. Hence, the following properties are always satisfied for all $x_i^j \in e_i$. There is a further edge $e(x_i^j)$ incident to x_i^j that verifies:[3]

- $e(x_i^j)$ is different from e_i (vertex domination);
- $e(x_i^j)$ is different from $(e_{3-i} \setminus \{x\}) \cup x_i^j$ (edge domination);
- $e(x_i^j)$ is different from all other $e(x_i^{j'})$ we chose (vertex domination).

Otherwise, some reduction rules will trigger and simplify the instance, so that the corresponding branching would become only better and hence negligible in the following analysis.

Let us assume that in the case that x is not taken into the cover, the heuristic priorities yield a branch at x_1^1. If x_1^1 is going into the cover, another smallish edge, $e_2 \setminus \{x\}$ remains. A rather trivial branch (due to edge domination) on that edge overall gives one $T_d^0(k-2)$-branch, one $T_d^1(k-2)$ branch and (for simplicity) $(d-4)$ $T_d^2(k-2)$-branches.

If x_1^1 is not taken into the cover, $P1$ would force us to continue branching on $e_1 \setminus \{x, x_1^1\}$, choosing say x_1^2. Moreover, we have "gained" a new small edge $e(x_1^1)$. According to the second observations listed above, there is a vertex $y \in e_2 \setminus \{x\}$ that is not contained in $e(x_1^1)$. We continue branching at such a y according to $P3$. However, in the case that we don't take y into the cover, we might fail to find a $y' \in e_2 \setminus \{x, y\}$ that is not contained in

[3]or, there are several edges each of which caters for one of the listed conditions; this more general case offers a better branching, since vertices of higher degree are involved

$e(x_1^1)$. Therefore, we get two $T_d^1(k-2)$ branches and (for simplicity) $(d-4)$ $T_d^2(k-2)$-branches.

If neither x_1^1 nor x_1^2 go into the cover, we have gained (at least) two small edges $e(x_1^1)$ and $e(x_1^2)$. According to the second observations listed above, there is a vertex $y \in e_2 \setminus \{x\}$ that is not contained in $e(x_1^1)$ (but possibly in $e(x_1^2)$). Altogether, we get one $T_d^1(k-2)$ branch and $(d-3)$ $T_d^2(k-2)$-branches, since when y is not put into the cover, we will find a y' in $e_2 \setminus \{x, y\}$ that is neither contained in $e(x_1^2)$ nor in $e(y)$.

This argument continues, yielding the formula as listed. ∎

Some algebraic and numerical computations then allows us to infer the following table:

d	4	5	6	10	100
$T_d^0(k) \approx$	3.0974^k	3.8993^k	4.7899^k	8.6304^k	98.5091^k

(5.12)

Case $e_1 \cap e_2 = \{x, y\}$:

Hence, $e_i = \{x, y, x_i^1, \dots, x_i^{d-3}\}$. A reasoning along the lines of Lemma 5.30 shows:

Lemma 5.31 *If* $e_1 \cap e_2 = \{x, y\}$ *and if* $d \geq 4$*, then we can estimate*

$$T_d^2(k) \leq T_d^0(k-1) + T_d^1(k-1) + (d-3)T_d^1(k-2) + (d^2 - 7d + 12)T_d^2(k-2).$$

To simplify the algebra, we used a slightly worse bound, namely

$$T_d^2(k) \leq T_d^0(k-1) + T_d^1(k-1) + (d-3)T_d^0(k-2) + (d^2 - 7d + 12)T_d^2(k-2),$$

to derive the following table:

d	4	5	6	10	100
$T_d^0(k) \approx$	3.1150^k	3.7984^k	4.6020^k	8.2848^k	98.0207^k

(5.13)

Now, we covered all cases for $d = 4$, deriving an $\mathcal{O}^*(3.1479^k)$ bound.

Case $|e_1 \cap e_2| = j \geq 3$:

Now, observe that vertex domination entails that each vertex in $e_1 \cap e_2$ has degree at least three. Therefore, without any further scrutiny, the edge domination rule (again only applied to the vertices from $e_1 \cap e_2$) yields:

Lemma 5.32 *If $|e_1 \cap e_2| = j$, $j \geq 3$ and if $d \geq j+2$, then we can estimate*

$$T_d^2(k) \leq T_d^0(k-1) + T_d^1(k-1) + (j-2)T_d^2(k-1) + (d-1-j)T_d^0(k-2).$$

After some algebra, we get: $0 = T_d^0(k+1) - (j+1)T_d^0(k) - (d^2 - (2j+1)d + j^2 + 1)T_d^0(k-1) + (d^2 - (2j+1)d + j^2 + j)T_d^0(k-2)$.

Some numerical computations entail:

d	5	6	7	10	100
$j = 3, T_d^0(k) \approx$	3.8662^k	4.5869^k	5.4339^k	8.2371^k	98.0156^k
$j = 4, T_d^0(k) \approx$	$---$	4.6964^k	5.3511^k	7.9283^k	97.5250^k
$j = 5, T_d^0(k) \approx$	$---$	$---$	5.5771^k	7.7344^k	97.0371^k
$j = 8, T_d^0(k) \approx$	$----$	$----$	$---$	8.3745^k	95.5916^k
$j = 98, T_d^0(k) \approx$	$----$	$----$	$---$	$---$	98.0308^k

(5.14)

We finally observe that the actual worst case situations are always found within the seemingly simplest situation, when $e_1 \cap e_2 = \emptyset$.

More tweaking for 4-HITTING SET

In order to improve on the constants of 4-HITTING SET, we are going to derive a special analysis for the T_4^1-estimate, since this immediately comes into play together with the T_4^2-estimate in the worst case situation $e_1 \cap e_2 = \emptyset$.

Lemma 5.33

$$T_4^1(k) \leq \max\{T_4^0(k-1)+2T_4^2(k-1), T_4^0(k-1)+2T_4^0(k-2)+T_4^1(k-1)\}. \quad (5.15)$$

Proof. Let $e = \{x, y, z\}$ be an edge of degree three we chose to branch on according to priority $P1$. According to Lemma 2.20, in particular all vertices in e have degree two or larger.

Let x be the vertex of maximum degree in e to start branching at according to $P2$. Taking x into the cover means a $T_4^0(k-1)$-branch. If we don't take x into the cover, an edge e_x of degree three is created.

We now consider two main cases.

- $\deg(x) = 2$. Hence, $\deg(y) = \deg(z) = 2$. $y \in e_x$ or $z \in e_x$ are excluded due to vertex domination.

  Let e_y and e_z be other edges (besides e) such that y is contained in e_y and z is contained in e_z. Due to vertex domination, $e_y \neq e_z$. If we take y into the cover, e_x would remain a small edge, and z would become dominated by any other vertex in e_z, so that $e_z' = e_z \setminus \{z\}$ is produced as another small edge. Hence, this is a $T_4^2(k-1)$-branch. If y does not go into the cover, then $e_y \setminus \{y\}$ would become another small edge. Moreover, z must go into the cover to hit e. Therefore, we have another $T_4^2(k-1)$-branch.

Summarizing, we arrived at the following inequality:

$$T_4^1(k) \leq T_4^0(k-1) + 2T_4^2(k-1). \tag{5.16}$$

- $\deg(x) \geq 3$. Hence, in the case that x does not go into the cover, we gain (at least) two new small edges e_x^1 and e_x^2.

  If $|\{y,z\} \cap (e_x^1 \cup e_x^2)| \leq 1$, we arrive at Eq. (5.16): first, $P1$ would let us branch at x, and then (if x does not go into the cover), $P3$ would cause a branch at say y with $y \notin (e_x^1 \cup e_x^2)$. If y is not put into the cover, then z goes; possibly, e_x^1 or e_x^2 gets destroyed, but there must be a new small edge e_y that is created in this case, so that this is also a $T_4^2(k-1)$-branch.

  Now, assume that $z \in e_x^1$. Then, we can assume that $z \notin e_x^2$ due to vertex domination. By edge domination with respect to e and to e_x^1, $y \notin e_x^1$. Due to the case analysis of the previous paragraph, we could however assume that $y \in e_x^2$. W.l.o.g., let us assume that we branch at y according to $P3$. Since $y \in e_x^2$, taking y into the cover will destroy e_x^2. Hence, this is a $T_4^1(k-1)$-branch. Not putting y into the cover would cause the edge e_x^2 shrinking to size two. Hence, in the case that z goes into the cover, we know by $P1$ that we can continue branching on $e_x^2 \setminus \{y\}$. Therefore, we get two $T_4^0(k-2)$-branches.

  Altogether, we derived for this case:

$$T_4^1(k) \leq T_4^0(k-1) + 2T_4^0(k-2) + T_4^1(k-1).$$

∎

For the case $e_1 \cap e_2 = \emptyset$, we therefore get the following estimate:

$$T_4^2(k) \leq \max\{T_4^1(k-1) + 2T_4^2(k-1), T_4^1(k-1) + 2T_4^1(k-2) + T_4^2(k-1)\}.$$

Some algebra then shows that our algorithm solves 4-HITTING SET in time $\mathcal{O}^*(3.1150^k)$, one of the worst cases being met via Lemma 5.33.

More tweaking for 3-HITTING SET

We will now explain some more details about how to further improve on the running time analysis in the case of 3-HS. The first lemma in fact is more generally applicable.

In what follows, we use $\#^d E$ to count the number of edges (or vertices) of the edge set E which have size d. Moreover, for a vertex x, $\deg^d(x)$ is the number of hyperedges of degree d that contain x.

Lemma 5.34 *If e is an edge in a hypergraph $G = (V, E)$ (as an instance of d-HITTING SET) to which the edge domination rule is not applicable, then in the branch where $x \in e$ is* not *taken into the cover, the instance $G' = (V', E')$ with $V' = V \setminus \{x\}$ and $E' = \{e \setminus \{x\} \mid e \in E\}$ will have*

$$\#^{d-1}E + \deg^d(x) - \deg^{d-1}(x) \tag{5.17}$$

many edges of degree $(d-1)$.

Proof. Not taking x into the cover means that, in order to cover edges $e \in E$ with $x \in e$, some $z \in e \setminus \{x\}$ must go into the cover. Therefore, the "next instance" is G'. The question is if $e \setminus \{x\} \in E$ for any e with $x \in e$, because then less than $\deg^d(x)$ edges of degree $(d-1)$ would be created. But this is ruled out by the edge domination rule. Since $\deg^{d-1}(x)$ edges of degree $(d-1)$ are "destroyed" in G', the formula is valid. ∎

The point is that we will have to extent our analysis to search trees of type T_d^3, i.e, situation where we have at least three small edges. Rem. 5.26 shows:

Lemma 5.35 *$T_d^0(k) \leq T_d^0(k-1) + T_d^3(k)$.*

Heuristics for 3-HITTING SET

We us the following *heuristic priorities.* $P2'$ is motivated by Lemma 5.34.

$P1$ *Prefer small edges.* More formally, let $E_0 = \{e \in E \mid |e| = 2\}$. If $E_0 = \emptyset$, set $E_0 = E$.

$P2'$ *Maximize Eq. (5.17)*, i.e., let $V_1 = \{x \in \bigcup_{e \in E_0} \mid \deg^3(x) - \deg^2(x)$ is maximum$\}$.

$P3$ Choose some $x \in V_1$ of *maximum degree.*

In the next lemma, we show a first step into a strategy which will finally give us better branching behaviors. Namely, we try to exploit the effect of reduction rules triggered in different sub-cases.

Lemma 5.36 $T_3^1(k) \leq \max\{2^k, T_3^0(k-1) + T_3^2(k-1)\}$.

Proof. The instance G has an edge $e = \{x, y\}$ of degree two. Assume that $\deg(x) \geq \deg(y)$, so that we branch at x. By Lemma 2.20, $\deg(x) \geq \deg(y) \geq 2$. We distinguish now two cases:

1. $\deg(y) = 2$. If we take x into the cover, then y will become of degree one and hence will get deleted by the reduction rules (see Lemma 2.20), producing one new edge e' of degree at most two with $e' \neq e$ due to edge domination. So this gives a $T_3^1(k-1)$-branch. Not taking x into the cover means to take y into the cover by the tiny edge rule, and at least one other edge of degree two (from the ones which have been incident to x) is created. This gives another $T_3^1(k-1)$-branch. The corresponding recurrence can be solved by 2^k as claimed.

2. $\deg(y) > 2$. The worst case is that $\deg(y) = \deg(x) = 3$. If x goes into the cover, we only get a $T_3^0(k-1)$-branch. If x is not put into the cover, y will be in the cover. Moreover, at least two new edges of degree two are created, yielding a $T_3^2(k-1)$-branch in this subcase.

∎

Lemma 5.37 *Assuming two edges of degree two, we get the following bound:*

$$T_3^2(k) \leq \max\{T_3^1(k-1)+T_3^2(k-1), T_3^0(k-2)+T_3^1(k-1), T_3^0(k-1)+T_3^2(k-2)\}.$$

Proof. We consider first the situation that the two edges e_1 and e_2 of degree two are disjoint. Then, basically the analysis of the previous lemma applies, showing that

$$T_3^2(k) \leq \max\{2^k, T_3^1(k-1) + T_3^3(k-1)\} \leq T_3^1(k-1) + T_3^2(k-1). \quad (5.18)$$

Otherwise, let $e_1 = \{x, y\}$ and $e_2 = \{x, z\}$. Then, $\deg(y) \geq 2$ and $\deg(z) \geq 2$ according to Lemma 2.20, because when branching, we always have reduced instances. We again make some case distinctions:

1. If $\deg(x) = 2$, then (since $\deg^3(x) = 0$ and $\deg^3(y) > 0$, $\deg^3(z) > 0$) we can assume we branch (w.l.o.g.) at y. If y is put into the cover, then x will be removed from the vertex set by the reduction rules (since it then has degree one) and z will be put into the cover, as well, giving a $T_3^0(k-2)$-branch. If y is not put into the cover, x will be by the tiny edge rule, covering both e_1 and e_2. Moreover, since $\deg^3(y) > 0$, at least one new edge e_y of degree two is created. Since the edge domination

rule was not applicable to the instance under consideration, $x \notin e_y$, because otherwise $e_1 \subset e_y \cup \{y\}$. This branch is of type $T_3^1(k-1)$. This yields:

$$T_3^2(k) \leq T_3^0(k-2) + T_3^1(k-1). \tag{5.19}$$

2. If $\deg(x) = 3$ and say $\deg(y) = 2$, assume we branch at x (due to heuristic priority $P3$). If x goes into the cover, then the vertex domination rule eliminates y, producing a new edge of degree two. This gives a $T_3^1(k-1)$-branch. Otherwise, both y and z must go into the cover, and $\deg^3(x) = 1$ gives us a new edge of degree two (due to edge domination). This yields a $T_3^1(k-2)$-branch. In conclusion, we have:

   $$T_3^2(k) \leq T_3^1(k-1) + T_3^1(k-2).$$

   This is always better than Eq. (5.19) and won't be considered any further.

3. If $\deg(x) \geq 3$ and $\deg(y) \geq 3$, assume we branch at y (due to heuristic priority $P2'$). A simple analysis yields (cf. Eq. (5.18))

   $$T_3^2(k) \leq T_3^1(k-1) + T_3^2(k-1).$$

4. If $\deg(x) \geq 4$, then assume that we branch at x. When x is not put into the cover, then we gain two new small edges. Therefore,

   $$T_3^2(k) \leq T_3^0(k-1) + T_3^2(k-2). \quad ∎$$

Estimating running times: As has been successfully done in other search-tree analyses, we will focus in the the following on analyzing "pure" cases, where we assume that we fix the possible branching scenarios for T_3^1 and for T_3^2, assuming that the worst case will show up in these pure cases. Of course, in practice, "mixed cases" will happen, where in the same search-tree different branching cases occur. This restriction to the worst-case analysis of pure branching scenarios brings along another benefit: looking closely at the analysis given in Lemma 5.37, one can notice that the worst case is always happening if the inequalities are satisfied with equality. Hence, we have to deal with the following set of recursions.

$$\begin{array}{rrcl}
 & T_3^0(k) & = & T_3^0(k-1) + T_3^2(k) \\
 & T_3^1(k) & = & T_3^0(k-1) + T_3^2(k-1) \\
(1) & T_3^2(k) & = & T_3^1(k-1) + T_3^2(k-1) \\
(2) & T_3^2(k) & = & T_3^0(k-2) + T_3^1(k-1) \\
(3) & T_3^2(k) & = & T_3^0(k-1) + T_3^2(k-2)
\end{array}$$

We are looking for a solution c_i such that $T_3^0(k) = c_i^k$ for $i = 1, 2, 3$, corresponding to case (i) in the list of equations above. Note that we left out the cases where obviously $T_3^j(k) \le 2^k$ can be shown; this is justified by the following analysis which always gives worse branchings.

1. In this case, we have to tackle the following equations:

$$\begin{aligned} T_3^0(k) &= T_3^0(k-1) + T_3^2(k) \\ T_3^1(k) &= T_3^0(k-1) + T_3^2(k-1) \\ T_3^2(k) &= T_3^1(k-1) + T_3^2(k-1) = T_3^0(k-2) + T_3^2(k-1) + T_3^2(k-2) \end{aligned}$$

The first equation gives $T_3^2(k) = T_3^0(k) - T_3^0(k-1)$; plugged in the last relation, we get:

$$(T_3^0(k) - T_3^0(k-1)) = (T_3^0(k-1) - T_3^0(k-2)) + T_3^0(k-2) + (T_3^0(k-2) - T_3^0(k-3))$$

This simplifies as follows:

$$T_3^0(k) = 2T_3^0(k-1) + T_3^0(k-2) - T_3^0(k-3)$$

which implies that $T_3^0(k) = c_1^k$ with $c_1 \le \boxed{2.2470}$.

2.

$$\begin{aligned} T_3^0(k) &= T_3^0(k-1) + T_3^2(k) \\ T_3^1(k) &= T_3^0(k-1) + T_3^2(k-1) \\ T_3^2(k) &= T_3^0(k-2) + T_3^1(k-1) \end{aligned}$$

Hence, $T_3^2(k) = 2T_3^0(k-2) + T_3^2(k-2)$ which implies

$$T_3^0(k) - T_3^0(k-1) = 3T_3^0(k-2) - T_3^0(k-3),$$

meaning that $T_3^0(k) \le 2.1701^k$.

3.

$$\begin{aligned} T_3^0(k) &= T_3^0(k-1) + T_3^2(k) \\ T_3^2(k) &= T_3^0(k-1) + T_3^2(k-2) \end{aligned}$$

By using the first equation, we get:

$$(T_3^0(k) - T_3^0(k-1)) = T_3^0(k-1) + (T_3^0(k-2) - T_3^0(k-3))$$

which yields

$$T_3^0(k) = 2T_3^0(k-1) + T_3^0(k-2) - T_3^0(k-3)$$

Therefore $T_3^0(k) = c_3^k \le \boxed{2.2470}^k$.

In other words, even with the two worst-case cases (highlighted by putting frames around them) we arrive at a better branching behavior than Niedermeier and Rossmanith did with their more intricate "bottom-up" approach.

By a more involved but similar analysis (based on slightly different heuristic priorities and two more reduction rules), together with the kernelization as explained by Niedermeier and Rossmanith, we can show:

Theorem 5.38 3-HITTING SET *can be solved in* $\mathcal{O}(2.1788^k + n)$ *time.*

Instead of giving the complete analysis, we will only mention the algorithmically relevant details in what follows.

In order to be able to better handle the situation that some of the small edges don't intersect, we refine our heuristic priorities as follows. Note that the additional heuristic rule we introduce does not affect the analysis of T_3^ℓ for $\ell < 3$.

$P1$ *Prefer small edges.* More formally, let $\deg^E_{\min} = \min\{|e| \mid e \in E\}$ and let $E_0 = \{e \in E \mid |e| = \deg^E_{\min}\}$.

$P1'$ *Prefer small lonely edges* If $\deg^E_{\min} = 2$, then if there is an edge $e' \in E_0$ such that $\forall e \in E_0 \setminus \{e'\}(e \cap e' = \emptyset)$ then set $E_0 = \{e'\}$.

$P2'$ *Maximize Eq. (5.17)*, i.e., let $V_1 = \{x \in \bigcup_{e\in E_0} \mid \deg^3(x) - \deg^2(x)$ is maximum$\}$.

$P3$ Choose some $x \in V_1$ of *maximum degree.*

Moreover, we employ the following two *special case reduction rules*, specific to the case of 3-HITTING SET:

Reduction rule 45 <u>path reduction</u> *Let* $G = (V, E)$ *contain the small edges* $e_1 = \{x, y\}$, $e_2 = \{y, z\}$ *and* $e_2 = \{u, x\}$, $u \neq z$. *Assume that* $\deg(x) = \deg(y) = 2$. *If* z *is dominated by* u *in* $G - e_2 = (V, E \setminus \{e_2\})$, *then delete* z, *i.e., reduce* G *to* $G' = (V \setminus \{z\}, \{e \setminus \{z\} \mid e \in E\})$ *without changing the value of the parameter.*

Reduction rule 46 <u>triangle reduction</u> *Consider the instance* (G, k), *where* $G = (V, E)$ *contains the small edges* $e_1 = \{x, y\}$, $e_2 = \{y, z\}$ *and* $e_3 = \{x, z\}$ *with* $\deg(x) = 2$. *Then, put* y *into the cover, i.e., reduce to* $(G', k - 1)$ *with* $G' = (V', E')$, $E' = \{e \in E \mid y \notin e\}$ *and* $V' = \{v \in V \mid \exists e \in E'(v \in e)\}$.

These rules can be seen as a variant of the vertex domination rule; small pictures can be found in Fig. 5.1(a) and Fig. 5.1(c), respectively.

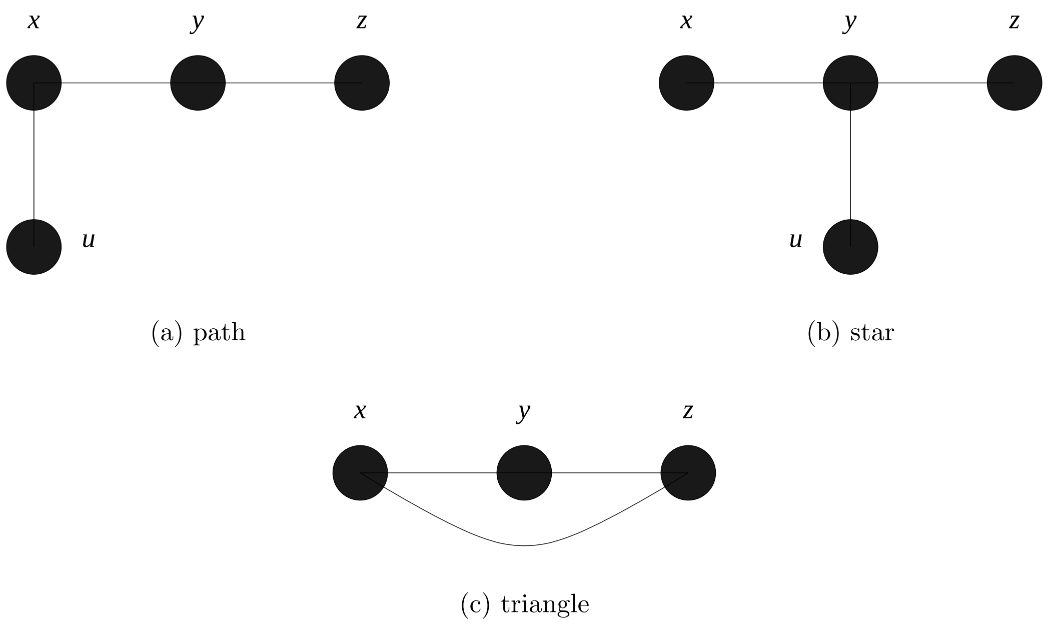


Figure 5.1: Three situations for arranging three edges.

5.3.4 FACE COVER

In this section, we consider the following problem on plane graphs:

Problem name: FACE COVER (FC)
Given: A plane graph $G = (V, E)$ with face set F
Parameter: a positive integer k
Output: Is there a *face cover set* $C \subseteq F$ with $|C| \leq k$?

Here, a *face cover set* is a set of faces whose boundaries contain all vertices of the given plane graph.

The material of this section is based on [2] and yet unpublished common work with F. Abu-Khzam and M. Langston.

Consider a plane graph $G = (V, E)$ with face set F. If we consider a vertex v to be given as a set $F(v)$ of those faces on whose boundary v lies, then a face cover set $C \subseteq F$ corresponds to a hitting set of a hypergraph $H = (F, E_H)$, where the vertex set of F is the face set of G, and

$$E_H = \{F(v) \mid v \in V\}.$$

Basically, Abu-Khzam and Langston have shown that a traditional HITTING SET algorithm can then be translated into a FACE COVER algorithm, where rather than deleting vertices or faces, they are *marked.* Vertices that are not yet marked are called *active.* Initially, all vertices and all faces are active. So, more formally, we are dealing with an annotated version of FACE COVER in the course of the algorithm, i.e.:

Problem name: ANNOTATED FACE COVER (FCANN)
Given: A plane graph $G = (V, E)$ with face set F, a function $\mu_V : V \to \{\text{active,marked}\}$ and a function $\mu_F : F \to \{\text{active,marked}\}$
Parameter: a positive integer k
Output: Is there a set $C \subseteq \{f \in F \mid \mu_F(f) = \text{active}\}$ with $|C| \leq k$ and $\forall v : \mu_V(v) = \text{active} \Rightarrow F_a(v) \cap C \neq \emptyset$?

In addition, marked vertices are shortcut by a sort of triangulation operation. This geometrical surgery allows to finally use the fact that each planar graph possesses a vertex of degree at most five to branch at (also cf. Site [Chapter 11, Site 6]). Let us explain this idea in more details. In order to do this, let us first translate the reduction rules we derived for HITTING SET in Chap. 2. In extension of the notation introduced above, let

- $F_a(v)$ collect all active faces incident to vertex v; $\deg_a(v) = |F_a(v)|$ be the *face degree* of v;
- $V_a(f)$ collect all active vertices on the boundary of face f; $\deg_a(f) = |V_a(f)|$ be the *face size* of f.

Rule 4 translates to:

Reduction rule 47 *Let $F_a(u) \subseteq F_a(v)$ for some active vertices u, v. Then, mark v.*

Rule 5 becomes:

Reduction rule 48 *If $\deg_a(v) = 1$ and v is active, then put the unique active incident face f (i.e., $F_a(v) = \{f\}$) into the face cover and mark both v and f.*

Rule 6 now reads as follows:

Reduction rule 49 *If $V_a(f) \subseteq V_a(f')$ for some active faces f, f', then mark f.*

As shown in [2], we have to be cautious with simply deleting vertices and faces, so that we only mark them with the above rules. However, it is indeed possible to simplify the obtained graph with a couple of *surgery rules.*

Reduction rule 50 *If u and v are two marked vertices with $u \in N(v)$, then merge u and v. (This way, our graph may get multiple edges or loops.)*

Reduction rule 51 *If u is a marked vertex with two active neighbors v, w such that u, v, w are all incident to an active face f, then partition f into two faces by introducing a new edge between v and w, and that edge has to be drawn inside of f. (This way, our graph may get multiple edges.) The new triangular face bordered by u, v, w is marked, while the other part of what was formerly the face f will be active.*

Reduction rule 52 *If $\deg(v) = 1$ and if v is marked, then delete v.*

Reduction rule 53 *If $\deg_a(v) = 0$ and if v is marked, then delete v. The new face that will replace all the marked faces that formerly surrounded v will be marked, as well.*

Reduction rule 54 *If f is a marked face with only one vertex or with two vertices on its boundary, then choose one edge e on the boundary of f and delete e. This will destroy f.*

Reduction rule 55 *If e is an edge with two incident marked faces f, f', then delete e, i.e., merge f and f'; the resulting new face will be also marked.*

Without proof, we state:

Lemma 5.39 *The reduction rules for* FACE COVER *are sound.*

We can now prove the following fact:

Lemma 5.40 *If $G = (V, E)$ is an annotated plane graph with face set F (seen as an instance of* ANNOTATED FACE COVER *with parameter k) that is reduced according to the* FACE COVER *reduction rules listed above, then no marked vertex will exist in G.*

Proof. Assume there is a marked vertex v in G. $\deg(v) > 1$, since otherwise Rules 53 or 52 would have been applicable. Hence, v has two neighbors u and w. If one of them would be marked, Rule 50 would have applied. Therefore, all neighbors of v must be active. To avoid application of Rule 53, we can assume that $\deg_a(v) > 0$. Then, Rule 51 applies $\deg_a(v)$ many times. The new triangular-shaped faces that would be introduced by that rule plus the already previously marked faces incident to v would be in fact all faces that are incident to v, and all of them are marked. Hence, Rule 53 (in possible cooperation with Rule 55) applies and deletes v. ∎

As mentioned within the reduction rules, it might occur that we create *degenerated faces* (i.e., faces with only one or two incident vertices) in the course of the application of the reduction rules.

Lemma 5.41 *If $G = (V, E)$ is an annotated plane graph with face set F (seen as instance of* ANNOTATED FACE COVER *with parameter k) that is reduced according to the* FACE COVER *reduction rules listed above, then the only degenerated faces that might exist are active faces f with two incident vertices. Moreover, the two faces that are neighbored with such a degenerated face f via common edges are both marked.*

Proof. Let f be a degenerated face. If f has only one vertex v on its boundary, the following will happen:

- If f and v are active and $\deg_a(v) = 1$, then Rule 48 applies and puts f into the face cover. Moreover, f and v will become marked. Hence, Rule 52 applies and deletes v, so that also f will be replaced by a probably larger face.
- If f and v are active and $\deg_a(v) > 1$, then Rule 49 applies and renders f marked.
- If f is active but v is marked, then $V_a(f) = \emptyset$, so that Rule 49 vacuously applies and renders f marked.
- If f is marked, then Rule 54 applies and removes f.

Hence, in the end a degenerated face with only one vertex on its boundary cannot exist in a reduced instance.

Consider now a degenerated face f with two vertices u and v on its boundary. If f is marked, then Rule 54 applies and removes f. Otherwise, f is active. Let f' be one of the faces with which f shares one edge. If f' were active, then Rule 49 would render f marked (see previous case). Hence, all faces that share edges with f are marked in a reduced instance. ∎

We are now going to analyze Alg. 37.

Theorem 5.42 (Abu-Khzam, Langston) *Alg. 37 solves* ANNOTATED FACE COVER *in time $\mathcal{O}^*(5^k)$.*

Proof. We have to show that, in an annotated (multi-)graph G, there is always a vertex with face degree at most five. Having found such a vertex v, the heuristic priority of choosing a face incident to the vertex of lowest face degree (as formulated in Alg. 37, line 8) would let the subsequent branches be made at faces also neighboring v, so that the claim then follows.

Algorithm 37 A simple search tree algorithm for ANNOTATED FACE COVER, called FC-ST

Input(s): an annotated plane graph $G = (V, E)$ with face set F and marking functions μ_V and μ_F, a positive integer k

Output(s): YES if G has an annotated face cover set $C \subseteq F$ with $|C| \leq k$; NO otherwise

```
    Exhaustively apply the reduction rules.
    {The resulting instance will be also called $G$ (etc.) as before.}
    if $k < 0$ then
      return NO
5:  else if $V_a = \emptyset$ then
      return YES
    else
      Let $v$ be a vertex of lowest face degree in $G$.
      {One incident face of $v$ must be used to cover $v$.}
10:   Choose $f \in F_a$ such that $f$ is incident to $v$.
      Mark $f$ and all vertices that are on the boundary of $f$.
      Call the resulting marking functions $\mu'_V$ and $\mu'_F$.
      if FC-ST$(G, F, \mu'_V, \mu'_F, k-1)$ then
        return YES
15:   else
        Mark $f$, i.e., return FC-ST$(G, F, \mu_V, \mu'_F, k)$
      end if
    end if
```

The (non-annotated) simple graph $G' = (V, E')$ obtained from the annotated (multi-)graph $G = (V, E)$ by putting *one* edge between u and v whenever there is *some* edge between u and v in G is planar; therefore, we can find a vertex v of degree at most five in G'.

However, back in G, an edge uv in G' might correspond to two edges connecting u and v, i.e., there might be up to ten faces neighboring v in G.[4] Lemma 5.41 shows that at most five of these faces can be active. ∎

Can we improve on the running time of the algorithm? Yes, we can again use Theorem 5.12. Of course, as long as in the auxiliary graph G' constructed according to the previous proof we find a vertex of degree four, we find a 4^k branching behavior and are happy with it. So, the only situation that needs to be analyzed is the following one (in G'): there are two triangular faces

[4]This was the basic concern when stating the $\mathcal{O}^*(10^k)$ (actually, even worse) algorithm for FACE COVER in [134].

neighbored via an edge $\{u, v\}$ where the sum of the degrees of u and v is at most 11. Since we want to analyze the worst case in the sequel, we can assume that $\deg(u) = 5$ and $\deg(v) = 6$.

We now discuss some possibilities for vertex u (which we would choose to branch at according to our heuristic priority, see line 8):

1. If some of the edges incident to u in G' represent degenerated faces in G and some correspond to (simple) edges in G, then Lemma 5.41 implies that the face degree of u in G is less than five, so that we automatically get a favorable branching.

2. If all edges incident to u in G' represent degenerated faces in G, then this is true in particular for the edge uv in G'. In order to achieve $\deg_a(v) = \deg(v)(= 6)$, all edges incident to v must also represent degenerated faces in G (otherwise, the branching would be only better, see Lemma 5.41).[5]

   We are dealing with the case that the edge $\{u, v\}$ is neighboring the marked triangular faces uwv and uzv (in G'). So, we have the following alternatives for covering u and v; as usual, we consider first all cases of covering the small-degree vertex u.

   - Take the degenerated face $\{u, w\}$ into the cover. Then, both faces $\{v, w\}$ and $\{u, v\}$ will get marked by the dominated face rule, so that for branching at v, only four further cases need to be considered. This is hence yielding four $T(k-2)$-branches.
   - Quite analogously, the case when we take $\{u, z\}$ into the cover can be treated, leading to another four $T(k-2)$-branches.
   - Otherwise, three possibilities remain to cover u, leading us to three $T(k-1)$-branches.

   Analyzing this branching scenario gives us: $T^k \leq 4.7016^k$.

3. If all edges incident to u (in G') correspond to simple edges in G, then we can assume (according to our previous reasonings) that also all edges incident to v (in G') correspond to simple edges in G.

[5]More precisely, if $\deg_a(v) = 5$ and $\deg(v) = 6$, this can only be if only four out of the six faces incident to v are non-degenerated. Branching first on the degenerated face uv and then (in the case that uv is not taken into the face cover) on all remaining four possibilities to cover u times four possibilities to cover v gives the recursion

$$T(k) \leq T(k-1) + 16T(k-2) \leq 4.5312^k.$$

Then, we would propose the following branching:

- Start branching at the active triangular faces uwv and uzv (in G'). This gives two $T(k-1)$-branches.
- Then, branch at the remaining three active faces surrounding u, followed (each time) by branches according to the remaining four active faces surrounding v; overall, this gives 12 $T(k-2)$-branches.

Analyzing this branching scenario gives us: $T^k \leq 4.6056^k$.

To further improve on our analysis, let us look a bit closer at the case that all edges incident with u and v represent degenerated faces. Let $N(u) = \{u_1, u_2, v, w, z\}$ be the neighbors of u and similarly $N(v) = \{v_1, v_2, v_3, u, w, z\}$.

- If the the degenerated face $\{u, w\}$ is put into the cover, then both faces $\{v, w\}$ and $\{u, v\}$ will get marked by the dominated face rule, so that for branching at v, four further cases need to be considered. Moreover, notice that our reduction rules will produce the following situation thereafter:[6]
  - u and v will be deleted.
  - All faces formerly neighboring u or v will be merged into one large marked face f.
  - On the boundary of f, w will be marked together with one vertex out of $\{v_1, v_2, v_3, z\}$.
- If the the degenerated face $\{u, z\}$ is put into the cover, then a similar analysis applies. This means that (according to the previous reasoning) we are left with the following situations:
  - u and v will be deleted.
  - All faces formerly neighboring u or v will be merged into one large marked face f.
  - On the boundary of f, z will be marked together with one vertex out of $\{v_1, v_2, v_3, w\}$.

  Now, observe that the situation that marks z together with w (on the boundary of f) has already been found before. Hence, we only need to consider three $T(k-2)$-branches here.

[6]In fact, the marked vertices will again disappear by further application of reduction rules, but this will be neglected in the following argument.

- The other cases are treated as before, giving three more $T(k-1)$-branches.

Altogether, we get as a recurrence for the search tree size:

$$T(k) \leq 3T(k-1) + 7T(k-2)$$

which gives the estimate $T(k) \leq 4.5414^k$.

This lets us conclude:

Algorithm 38 The code of FC-ST-case-1

```
    Let $G' = G \setminus \{u, v\}$ and mark the face to which (formally) $u, v$ belonged;
    modify $F, \mu_V, \mu_F$ accordingly, yielding $F'$, $\mu'_V$ and $\mu'_F$.
    **if** FC-ST-advanced($G', F', \mu'_V, \mu'_F, k-1$) **then**
      return YES
    **else**
5:    **for all** unordered vertex pairs $\{x, y\}$ such that $x \in N(u) \setminus \{v\}$ and
      $y \in N(v) \setminus \{x, u\}$ **do**
        Modify $\mu'_V$ so that $x$ and $y$ are the only vertices of $(N(u) \cup N(v)) \setminus$
        $\{u, v\}$ that are marked.
        **if** FC-ST-advanced($G', F', \mu'_V, \mu'_F, k-2$) **then**
          return YES
        **end if**
10:   **end for**
      return NO
    **end if**
```

Theorem 5.43 FACE COVER *can be solved in time* $\mathcal{O}^*(4.6056^k)$.

The pseudo-code of such an efficient algorithm for FACE COVER is shown in Alg. 41. For better readability, we grouped out three code fragments as macros that deal with the three special cases we had analyzed.

Using more sophisticated algorithms (based on structural corollaries of Theorem 5.12), Abu-Khzam, Fernau and Langston were able to further lower the base of the exponential algorithm to about 4.5.[7]

We conclude with two final remarks:

1. Bienstock and Monma [47] considered a variant of FACE COVER where some preselected vertices need not be covered. This variant can be solved by our algorithm, as well, since it evidently gives a restriction of ANNOTATED FACE COVER.

[7]unpublished note

Algorithm 39 The code of FC-ST-case-2

```
Let $G' = G \setminus \{u, v\}$ and mark the face to which (formally) $u, v$ belonged;
modify $F, \mu_V, \mu_F$ accordingly, yielding $F'$, $\mu'_V$ and $\mu'_F$.
for all vertices $x \in \{w, z\}$ do
  Modify $\mu'_V$ so that $x$ is the only vertex of $(N(u) \cup N(v)) \setminus \{u, v\}$ that
  is marked.
  if FC-ST-advanced$(G', F', \mu'_V, \mu'_F, k-1)$ then
    return YES
  end if
end for
for all vertices $x \in \{u_1, u_2\}$ and $y \in \{v_1, v_2, v_3\}$ do
  Modify $\mu'_V$ so that $x$ and $y$ are the only vertices of $(N(u) \cup N(v)) \setminus \{u, v\}$
  that are marked.
  if FC-ST-advanced$(G', F', \mu'_V, \mu'_F, k-2)$ then
    return YES
  end if
end for
return NO
```

Algorithm 40 The code of FC-ST-case-3

```
for all faces $f \in F_a(u)$, $g \in F_a(v)$ do
  Mark $f$ and $g$ and all vertices in $V_a(f) \cup V_a(g)$; call the modified marking
  functions $\mu'_V$ and $\mu'_F$.
  if FC-ST-advanced$(G', F, \mu'_V, \mu'_F, k - |\{f, g\}|)$ then
    return YES
  end if
end for
return NO
```

2. RED-BLUE DOMINATING SET (restricted to planar instances) can be also solved with our algorithm. Formally, we only must (after arbitrarily embedding the given red-blue graph into the plane)

   - mark all faces of the graph
   - attach to each red vertex an active loop
   - inflate a loop attached to the red vertex v along the edges that connect v with its blue neighbors until the loop (seen as a region in the plane) touches all these neighbors; then, we can finally remove v.

Algorithm 41 An advanced search tree algorithm for ANNOTATED FACE COVER, called FC-ST-advanced

Input(s): an annotated plane graph $G = (V, E)$ with face set F and marking functions μ_V and μ_F, a positive integer k

Output(s): YES if G has an annotated face cover set $C \subseteq F$ with $|C| \leq k$; NO otherwise

```
    Exhaustively apply the reduction rules.
    {The resulting instance will be also called $G$ (etc.) as before.}
    if $k < 0$ then
      return NO
 5: else if $V_a = \emptyset$ then
      return YES
    else
      Let $u$ be a vertex of lowest face degree in $G$.
      {One incident face of $u$ must be used to cover $u$.}
10:   if $\deg_a(u) \leq 4$ then
        Choose $f \in F_a$ such that $f$ is incident to $u$.
        Mark $f$ and all vertices that are on the boundary of $f$.
        Call the resulting marking functions $\mu'_V$ and $\mu'_F$.
        if FC-ST-advanced$(G, F, \mu'_V, \mu'_F, k-1)$ then
15:       return YES
        else
          Mark $f$, i.e., return FC-ST-advanced$(G, F, \mu_V, \mu'_F, k)$
        end if
      else
20:     {Let $N(u) = \{u_1, u_2, v, w, z\}$ be the neighbors of $u$ and similarly
        $N(v) = \{v_1, v_2, v_3, u, w, z\}$.}
        if all active faces incident with $v$ are degenerated then
          execute FC-ST-case-1
        else if no active faces incident with $v$ are degenerated then
          execute FC-ST-case-2
25:     else
          {all active faces incident with $u$ are degenerated; only one active
          face incident with $v$ is degenerated}
          execute FC-ST-case-3
        end if
      end if
30: end if
```

Chapter 6

Case studies

We are going to explore the applicability of the parameterized approach to algorithmics by working out examples from different areas. This shows the applicability of the various techniques we have seen so far to obtain parameterized algorithms.

In Sec. 6.1, we develop an efficient search tree algorithm for a problem that arises in the area of statistical databases (we will describe the background of the problem, too). Sec. 6.2 is devoted to the study of various problems that are intimately related to HITTING SET; in fact, many other problems that we meet in this chapter can be also viewed from this perspective. The application areas we draw our examples from are quite diverse: computational biology and communication networks; in further sections, we will also learn examples from graph drawing and Artificial Intelligence. Section 6.3 is devoted to the study of various graph modification problems, which again are often related to (or can be formulated as) HITTING SET problems. In Sec. 6.4, we will encounter different graph drawing problems, studied from a parameterized point of view. In particular, we will work again on LINEAR ARRANGEMENT and its variants. Finally, Section 6.5 reviews again the techniques we have seen so far for developing efficient parameterized algorithms.

6.1 MATRIX ROW COLUMN MERGING

The material of this section is taken from [60, 172].

6.1.1 Problem definition

L. Branković communicated to us the following abstract problem arising in the theory of statistical databases, stated below in an abstract and simplified version:

Problem name: MATRIX ROW COLUMN MERGING (MRCM)
Given: a $n \times m$ $\{0,1\}$-matrix M
Parameter: a positive integer k
Output: Is it possible to get the all-zeros-matrix by merging at most k neighboring rows or columns?

Here, *merging* means to perform a component-wise logical AND. For short, we will also call this a *k-mergible matrix*. In order to simplify our formulations, a *line* of a matrix is either a row or a column of that matrix. Hence, a matrix is k-mergible iff the all-zeros-matrix can be obtained by merging at most k neighboring lines. A line can be of *type* row or column. We also say that rows and columns are *opposite types*.

6.1.2 The database background

Let us explain a bit more the background from statistical databases.

Statistical security is an area of data security that focuses on the protection of individual values used for statistical purposes. This problem became apparent in the seventies [125] and was revived in recent years due to the massive data collection and growing social awareness of individual privacy. This problem is different from most other security problems which typically occur when an unauthorized user obtains an access to data, or when an authorized user obtains an access to data for which he/she is not authorized. The latter is referred to as unauthorized access by an authorized user. Note that both these problems can be solved by cryptographic and access control techniques. However, these techniques cannot solve statistical security problem, as the access to confidential individual values is not obtained directly but rather indirectly through a combination of legitimate statistical queries. Thus we have a case of a security breach by an authorized user obtaining authorized access to data.

The techniques used for preventing statistical disclosure fall into two categories: noise addition, where all users' queries are answered but the answers are only approximate rather than exact, and query restriction, where the system only accepts those queries that are considered safe. In either case, a technique is evaluated by measuring both the information loss and the achieved level of privacy. In the case of a noise addition technique, the information loss is measured by deterioration in data quality, in terms of *bias* (the difference between unperturbed statistics and the expected value of its perturbed estimate), *precision* (variance of an estimator) and *consistency* (absence of contradictions and paradoxes) [4]. For query restriction techniques, information loss is measured by the decrease in the usability, that is, the percentage of queries that can be answered without database compromise. Obviously, the goal of statistical data protection is to maximize the security while minimizing the information loss. In order to evaluate a particular technique it is important to find the theoretical lower bound on the information loss necessary to achieve a given level of privacy.

Here, we focus on query restriction in a database that contains a single confidential attribute and only allows so-called *range sum queries* on that attribute, such that it is impossible for an intruder to disclose any confidential individual value. Range queries are those that are based on records that have all attribute values within the prescribed ranges. For example, in a database with attributes "age", "education" and "years of experience" with the domains [18, 70], [HighSchool, TradeCertificate, Bachelor, Master, PhD] and [0,55], respectively, and a confidential attribute "salary", the following are range sum queries:

$$SUM(age > 50, education > bachelor; salary)$$
$$SUM(age < 35, experience < 10; salary)$$

Note that all the attributes are numerical or enumeration types, that is, all of them have a natural ordering of their values.

Databases that only allow range queries can be represented as multidimensional matrices where range queries are contiguous sub-matrices [237]. Each entry in the matrix denotes how many records with that particular combination of attribute values exist in the database. This representation is essentially the same as the *data cube* in OLAP (On-line Analytical Processing) [157], where the range queries are typically referred to as *multi-dimensional range (MDR)* queries and are widely used for exploring patterns in data warehouses. As an illustration, consider a database given in Table 6.1 and its data cube representation in Table 6.2.

Another representation often used for release of statistical data is the so-called *tabular model*, in which the entries represent the number of records with attribute values falling within specified ranges. Note that the tabular model contains less information than the data cube and thus these two models are not equivalent. Table 6.3 shows a possible tabular representation of our Sample Database from Table 6.1.

We now consider m-dimensional matrices in which all entries are greater than 0, that is, in which there exists at least one record for each combination of attribute values. It was shown in [237] that for such m-dimensional

Age	*Salary*
21	25000
21	27000
21	28000
22	26000
22	27000
22	28000
24	26000
24	27000
24	28000
25	26000

Table 6.1: Sample Database

	25000	26000	27000	28000
21	1	0	1	1
22	0	1	1	1
23	0	0	0	0
24	0	1	1	1
25	0	1	0	0

Table 6.2: Sample Database—data cube representation

	25000–26000	27000–28000
21–22	2	4
23–24	1	2
25	1	0

Table 6.3: Sample Database—tabular representation

matrices the fraction of queries that can be answered without exact disclosure of any individual value is bounded from below by $(2^m - 1)/2^m$. The minimum is achieved in the case where there is precisely one record for each combination of attribute values, that is, when a database matrix contains only ones. A precise formula for the usability of one-dimensional databases with arbitrary elements in the database matrix is given in [61], and databases with 0 entries (holes) were considered in [374].

Here, we restrict the problem to two dimensions and tackle it from a different angle. The main idea is simple: if the database is dense, that is, if the database matrix does not contain too many zeros, and if zeros are spread across the matrix, then it is worth partitioning the domains of the attributes into intervals, and disallowing all the queries whose attribute ranges cannot be expressed as unions of these intervals. This effectively corresponds to a transition from a data cube to a tabular model where the ranges are not necessarily equal but all the entries are greater than 0. In other words, this corresponds to merging neighboring rows and columns in a database matrix in order to eliminate all 0's in the matrix. Once this is achieved, we know from [237] that we can keep the data secure while achieving very high usability by allowing only "even" range queries, that is, those queries that are represented by contiguous sub-matrices with even number of elements. Clearly, our goal is to minimize the number of row or column mergings necessary for eliminating all 0's in the matrix. If we apply this to our Sample Database in Table 6.2, we arrive at the tabular model presented in Table 6.4. The advantages of this representation over the representation in Table 6.3 are obvious: there are more entries than in Table 6.3 and the set of all contiguous sub-matrices containing even number of cells is compromise free.

	25000-26000	27000	28000
21	1	1	1
22-23	1	1	1
24-25	2	1	1

Table 6.4: Sample Database—tabular representation that minimizes the number of mergings, while ensuring that all entries are greater than 0

Note that in the mathematical formulation of the preceding subsection, zeros in the database matrix are represented by 1 while positive integers are represented by 0.

6.1.3 Some examples

Let us first discuss a simple example. In this way, we will introduce a bit more notation.

Example 6.1 Let us consider the following matrix instance M (where only the 1-entries are shown, all other entries are implicitly 0-entries):

$$\begin{array}{c|ccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
1 & 1 & & & 1 & & & & & 1 \\
2 & & & & & & & & & \\
3 & 1 & & & & & & & & \\
4 & & & 1 & 1 & & & 1 & & \\
5 & & & & & & & & & \\
6 & & & & & & & & & \\
7 & & & & 1 & & & & & 1
\end{array}$$

Can we eliminate the ones by merging at most $k \leq 3$ neighboring rows or columns? For example, merging rows 1 and 2, 4 and 3, as well as 6 and 7 leaves us with:

$$\begin{array}{c|ccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
1/2 & & & & & & & & & \\
3/4 & & & & & & & & & \\
5 & & & & & & & & & \\
6/7 & & & & & & & & &
\end{array}$$

Observe that the merged rows are written as a list of row numbers. This notation is convenient, since it allows us to talk about, e.g., merging rows 4 and 5 later on (of course, this is not necessary in our example), referring to the original row numbers. Alternatively, we would have to re-number the lines after each merge operation, but this appears to be too awkward to explain examples, although it is quite natural when thinking about implementing solving algorithms. That is why we will later deviate from this convention. A second point we gain by our convention is that the order in which we apply merge operations does not matter at all. In our example, we could describe our solution as $\{\mathrm{RM}(1,2), \mathrm{RM}(3,4), \mathrm{RM}(6,7)\}$. Similarly, we can indicate column mergers by writing $\mathrm{CM}(i, j)$. To simplify our notation, we will also say that the operation $\mathrm{RM}(\cdot, \cdot)$ is of the *type row*, and the operation $\mathrm{CM}(\cdot, \cdot)$ is of the *type column*.

In other words, the given instance is 3-mergible. Is it possible to fix the instance by using only two merging operations? Let us look at the first row. It contains three ones. If all the ones were “fixed” by column mergers, this would need at least three merging operations. Hence, we must fix these ones by merging row 1 with row 2. A similar argument applies to row 4. But now we have used two (enforced) mergers but did not fix the matrix successfully. Hence, the instance is not 2-mergible.

6.1.4 Complexity results

In [172], we have shown the following result that actually makes the problem interesting to be tackled by a parameterized approach:

Theorem 6.2 MATRIX ROW COLUMN MERGING *is $\mathcal{NP}$-complete.*

In fact, all results we list in the following can be found in [172] if not noted otherwise.

Let us first try to find a simple way of proving membership of MRCM in $\mathcal{FPT}$: observe that a one at location (i, j) in the instance matrix M can only be fixed by merging it row-wise or column-wise with a neighboring line. In total, this gives a four-element *candidate set* for fixing $M(i, j)$, see Alg. 42.

It is clear that the running time of Alg. 42 is $\mathcal{O}^*(4^k)$, since each of the four branches reduces the parameter by one, i.e., the (exponential part of the) running time is bounded by the recursion $T(k) \leq 4T(k-1)$.

Can we improve on this running time? Firstly, observe that the worst case can only happen if we are not able to find a one in some row or column in M that is a *borderline* of M. Otherwise, one of the branches cannot happen, so that we immediately get a $\mathcal{O}^*(3^k)$ behavior. What does it mean if say the first row contains only zeros? Obviously, we don't need any column mergers to repair that row. Consider the row merging $\mathrm{RM}(1, 2)$. The resulting matrix M' coincides with the given one M, assuming that the second row got deleted. However, if we did, instead of $\mathrm{RM}(1, 2)$, merge rows 2 and 3, then the resulting matrix M'' would contain, as would M', a first row containing only zeros and a second row that has, compared to the second row of M', at most ones in places in which M' has ones, as well. In all other rows, M' and M'' coincide. Hence, if we compare the matrices M' and M'' componentwisely, $M'' \leq M'$.

Proposition 6.3 *If (M_1, k) is a* YES*-instance of* MATRIX ROW COLUMN MERGING *and if M_2 is a $\{0, 1\}$-matrix with the same dimensions as M_1, such that $M_2 \leq M_1$ by componentwise comparison, then (M_2, k) is a* YES*-instance of* MRCM*, as well.*

Due to Prop. 6.3, the preceding argument shows that we do not have to consider the operation $\mathrm{RM}(1, 2)$ at all in our branchings, since $\mathrm{RM}(2, 3)$ would do no worse, and $\mathrm{RM}(2, 3)$ will be considered in one of the later branches anyhow. The only exception to this observation is when there is no third row at all, so that $\mathrm{RM}(1, 2)$ is the only possible row merging operation.

This is the basic idea to show the soundness of the following reduction rule.

Algorithm 42 A simple search tree algorithm for MATRIX ROW COLUMN MERGING, called MRCM-simple

Input(s): a binary $n \times m$ matrix M, a positive integer k

Output(s): YES if there is sequence of line merges of length at most k that turns M into a matrix that only contains zeros or NO, otherwise.

```
   if $M$ contains no ones then
     return YES
   else if $k > 0$ then
     pick some index $(i, j)$ such that $M[i, j] = 1$;
5:   {branch according to the four fixing possibilities}
     if $i > 1$ then
       if MRCM-simple(RM$(i-1, i)(M), k-1$)=YES then
         return YES
       end if
10:  end if
     if $i < n$ then
       if MRCM-simple(RM$(i, i+1)(M), k-1$)=YES then
         return YES
       end if
15:  end if
     if $j > 1$ then
       if MRCM-simple(CM$(j-1, j)(M), k-1$)=YES then
         return YES
       end if
20:  end if
     if $j < m$ then
       return MRCM-simple(CM$(j, j+1)(M), k-1$)
     else
       return NO
25:  end if
   else
     return NO
   end if
```

Reduction rule 56 <u>*0-borderline-rule:*</u> *If (M, k) is an instance of* MATRIX ROW COLUMN MERGING *whose first and last row are completely consisting of zeros, then (M, k) can be reduced to (M', k), where M is obtained from M by deleting the first row.*

A similar rule can be stated for columns.

In actual fact, this rule is true in a slightly more general fashion; this also explains some modifications that have to be done in [172] to cope with the (additional) if-condition.

Reduction rule 57 *0-borderline-rule (general): If (M, k) is an instance of* MATRIX ROW COLUMN MERGING *whose first row is completely consisting of zeros, then (M, k) can be reduced to (M', k), where M is obtained from M by deleting the first row, if M' does not contain a column that is completely filled with ones.*

A similar rule can be stated for columns.

Due to the 0-borderline-rule, we can be sure to always find a one in the first or in the last column of a reduced matrix (if that matrix contains at least one element). This shows the correctness of Alg. 43, and also the running time of $\mathcal{O}^*(3^k)$ should be clear.

Can we further improve on this running time? In fact, in [172] we described a method of how to do this, basically by a *triviality last* methodology. To be able to do so, we must first find trivial instances, i.e., instances that can be solved in polynomial time.

To this end, we considered the following problem variant:

Problem name: MATRIX ROW MERGING (MRM)
Given: a $n \times m$ $\{0, 1\}$-matrix M
Parameter: a positive integer k
Output: Is it possible to get the all-zeros-matrix by merging at most k neighboring rows?

How can we solve this problem in polynomial time? Quite easily, with the help of reduction rules. Namely, first of all, observe that, since MRM is a restriction of MRCM, the reduction rule 56 is also valid in this case. But, if (M, k) is an instance that is reduced with respect to that rule, in particular the first row contains a one. Such a one can only be fixed by using the operation RM(1,2), since there are no column mergers at our disposal. This justifies the following specific reduction rule:

Reduction rule 58 *Let (M, k) be an instance of* MATRIX ROW MERGING. *If the first row of M contains a one, merge that row with the second row to obtain an instance $(M', k-1)$. If the last row of M contains a one, merge that row with the penultimate row to get an instance $(M', k-1)$.*

Observe that always either Rule 56 or Rule 58 is applicable to an instance M of MATRIX ROW MERGING, since alway either the first row or the last row

Algorithm 43 A revised search tree algorithm for MATRIX ROW COLUMN MERGING, called MRCM-revised

Input(s): a binary $n \times m$ matrix M, a positive integer k
Output(s): YES if there is sequence of line merges of length at most k that turns M into a matrix that only contains zeros or
NO, otherwise.

```
Exhaustively apply the 0-borderline-rule.
{The resulting matrix is also called $M$.}
if $M$ contains only zeros then
  return YES
else if $k > 0$ then
  if the first column contains only zeros then
    if $M$ contains only two columns then
      return YES
      {Here, CM(1,2) will do.}
    else
      Replace $M$ by a matrix that is obtained from $M$ by swapping the
      first and the last column, the second and the third column etc.
      {This covers the case that the last column, not the first column,
      of $M$ contains a one.}
    end if
  end if
  pick some index $(i, 1)$ such that $M[i, 1] = 1$;
  {branch according to the at most three fixing possibilities}
  if $i > 1$ then
    if MRCM-simple(RM$(i-1, i)(M), k-1$)=YES then
      return YES
    end if
  end if
  if $i < n$ then
    if MRCM-simple(RM$(i, i+1)(M), k-1$)=YES then
      return YES
    end if
  end if
  if $1 < m$ then
    return MRCM-simple(CM$(1, 2)(M), k-1$)
  else
    return NO
  end if
else
  return NO
end if
```

of M contains a one or both the first and the last row of M contain only zeros.

It is obvious that an instance (M, k) that is reduced both according to both Rules 56 and 58 is either a matrix with zero or with one row; in the first case, the instance is solvable, while in the second case, it is not. Observe that, according to the last remarks in Sec. 2.1.1, it is well possible that an instance is detected as non-solvable when trying to apply the reduction rules, namely if the budget k that was originally provided was too small.

As detailed in [172], the sketched polynomial time solving algorithm for MATRIX ROW MERGING is interesting on its own right since it also yields a simple solving algorithm for VERTEX COVER on so-called interval graphs.

Having found this simple solution for row (or column) mergers only, we can now formulate an algorithm for which we can show a further improved running time, called MRCM-final, see Alg. 44. This procedure is used as described in the following:

```
Let $(M, k)$ be the given MRCM instance.
$A$ is an initially all-zero auxiliary matrix with the same dimensions as $M$.
Call $S$ :=MRCM-final$(M, k, \emptyset, 0, k, A, 1, 0)$.
**if** $M$ contains a one and $S = \emptyset$ **then**
  $(M, k)$ is a NO-instance.
**else**
  $(M, k)$ is a YES-instance, solved by $S$.
**end if**
```

The idea of this algorithm is pretty simple and similar to the previous case: a one in the first column can be either solved by a column merger or by some row mergers. The trick is that we defer solving by row mergers till the very end, which is possible since the merging operations are commutative (in a certain sense: namely, when considered on the original matrix line indexing). Observe that $\lambda \geq 1$ due to the fact that M contains at least one one in the first column.

Observe that the line marked with (*) in Alg. 44 is needs some modifications along the lines of Alg. 43 to cover the cases in which the 0-borderline-rule is not applicable; since the corresponding swapping of indices introduces some technicalities that are tedious but don't add to the understanding of the overall procedure. The necessary modifications should be doable by the reader.

We allow ourselves to override some ones in M by twos, namely, if we decide that the corresponding row containing that one should be resolved by a row merger. Then, we should not consider the possibility of repairing a column containing only zeros and twos by column mergings (at all). Rather,

Algorithm 44 Our final search tree algorithm for MATRIX ROW COLUMN MERGING, called MRCM-final

Input(s): a binary $n \times m$ matrix M, a positive integer k, a solution set S, an auxiliary parameter ℓ, an auxiliary matrix A, a pointer a pointing to the next free row in A, a pointer '`offset`' that keeps track of the "original" index minus one of what is now the first column of M (M gets modified throughout recursion)

Output(s): YES if there is sequence of line merges of length at most k that turns M into a matrix that only contains zeros or
NO, otherwise.

```
Let $c$ be the number of columns of $M$.
if the first column of $M$ contains no ones AND $c > 2$ then
   copy the first column of $M$ into the $a$s column of $A$, thereby turning 2s into 1s;
   op:=delete first column; {(*); see text body}
   return MRCM-final(op($M$),$k$,$S$,$\ell$,$A$,$a+1$,offset+1).
end if
{Now, the first column of $M$ contains a 1 or $c \leq 2$. The remaining simple cases are solved with the MRM-solver, as described in the text.}
if $(c \leq 1) \vee (\ell < 1)$ then
   copy all columns from $M$ into $A$, starting at position $a$, thereby turning 2s into 1s;
   return solve-MRM($A, k, S, 0$).
else
   {Branch according to two subcases}
   op:=CM(1,2);
   $S_{update} := S \cup \{\text{CM}(1 + \texttt{offset}, 2 + \texttt{offset})\}$;
   $S'$ :=MRCM-final(op($M$),$k-1$,$S_{update}$,$\ell-1$,$A$,$a$,offset+1).
   if $S' \neq \emptyset$ then
      return $S'$
   else
      {Try the row mergers}
      if $c = 2$ then
         {the only remaining simple case}
         copy all columns from $M$ into $A$, starting at position $a$, thereby turning 2s into 1s;
         return solve-MRM($A, k, S, 0$).
      else
         {$c > 2$. Hence, the first column of $M$ contains a one.}
         MODIFY $M$ and $A$ (see Alg. 45)
      end if
   end if
end if
```

Algorithm 45 The macro MODIFY as used in Alg. 44.

```
$\lambda := 0$;
set the $a$th column of $A$ to be the all-zero-vector;
for all rows $i$ in the first column of $M$ with $M[i,1] = 1$ do
  $\lambda := \lambda + 1$;
  for all columns $j > 1$ of $M$ with $M[i,j] = 1$ do
    $M[i,j] := 2$;
  end for
end for
for all rows $i$ in the first column of $M$ with $M[i,1] \neq 0$ do
  $A[i,a] := 1$;
end for
op:=delete first column;
return MRCM-final(op($M$),k,S,$\ell - \lambda/2$,$A$,$a+1$,offset+1).
```

in this sense a two would be treated as a zero by the 0-borderline-rule within M, deferring the solution to the polynomial-time phase with solve-MRM. This justifies the initial part of the algorithm. Still, if we decide to merge the first column with the second which may contain a two, then this two is treated as if it were a one. More precisely, merging within M is then done according to the following algebraic laws:

- 0 AND $X = 0$ for $X = 0, 1, 2$;
- X AND $X = X$ for $X = 0, 1, 2$.

Note that we will never merge a one with a two, since all ones in a row will be turned into twos (when we assume that that row is fixed by row merging) or not, see the macro described in Alg. 45 for details. Hence, the given rules cover all cases that may occur.

The idea behind the new variable ℓ is that for each row which we decide to be resolved by row mergers only, we can be sure to put aside at least one half of a parameter value, since one row merger can completely resolve at most two rows to be fixed by row mergers only. This "book-keeping trick" (which might be in fact wider applicable: use simple approximation bounds to account for cases to be resolved later in a deterministic, polynomial-time fashion) gives us the following recursion:

$$T(\ell) \leq T(\ell - 1) + T(\ell - 1/2) + O(1)$$

Assuming $T(\ell) = c^\ell$ yields $c = (3 + \sqrt{5})/2 \approx 2.6181$. Alternatively, $c = \phi^2 = \phi + 1$, where $\phi \approx 1.6181$ is the number of the golden ratio.

Hence, we may state as our final version:

Theorem 6.4 MATRIX ROW COLUMN MERGING *can be solved in time* $\mathcal{O}(2.6181^k mn)$*, given an* $n \times m$ *matrix* M *and a parameter* k*.*

6.1.5 Revisiting our example

Let us follow the work of this algorithm by having another look at the first example, solving it as instance $(M, 3)$. For clarity, we put the value of the `offset` variable into each table.

`offset` $= 0$	1	2	3	4	5	6	7	8	9
1	1			1					1
2									
3	1								
4			1	1			1		
5									
6									
7				1					1

The first branch would take CM(1,2), and one application of the 0-borderline-rule (in its general form) leaves us with a new instance $(M\{1\}, 2)$, using curly brackets to denote the different subcases:

`offset` $= 2$	1	2	3	4	5	6	7
1		1					1
2							
3							
4	1	1			1		
5							
6							
7		1					1

Alternatively, the second branch would copy the first column of M into A and color some 1s into 2s, arriving at the following matrix instance $(M\{2\}, 2)$, after one application of the 0-borderline-rule (in its general form):

`offset` $= 2$	1	2	3	4	5	6	7
1		2					2
2							
3							
4	1	1			1		
5							
6							
7		1					1

To shortcut the exposition, let us now focus on what further happens with this second branch, i.e., with $(M\{2\}, 2)$: Again, we have CM(1,2) as first alternative, yielding as next instance $(M\{2.1\}, 1)$. Notice that since the current offset is two, the operation CM(3,4) will be put into the partial solution in this branch. So, we arrive at:

`offset` = 3	1	2	3	4	5	6
1						2
2						
3						
4	1			1		
5						
6						
7						1

Obviously, this is a dead end (it cannot be repaired by only one merging operation). Alternatively, deferring the row merging yields $(M\{2.2\}, 1.5)$:

`offset` = 3	1	2	3	4	5	6
1	2					2
2						
3						
4	2			2		
5						
6						
7	1					1

The matrix A looks up to now as follows:

	1	2
1	1	
2		
3	1	
4		1
5		
6		
7		

Since column mergers alone surely would not resolve $(M\{2.2\}, 1.5)$, let us follow up the situation assuming row mergers. At an intermediate state, in $M\{2.2\}$ all ones would be turned into twos. The copy operations would then finally turn A besides one missing 0-column) into the original matrix M. Applying solve-MRM$(A, 3, \emptyset, 0)$ will then return as solution

$$S = \{\mathrm{RM}(1, 2), \mathrm{RM}(3, 4), \mathrm{RM}(6, 7)\}.$$

6.2 Problems related to HITTING SET

In this section, we will show how a whole variety of problems can be solved with the help (or at least taking ideas from) HITTING SET algorithms as developed in Chap. 5. In fact, we already started with such type of problems in the preceding section, since we sketched the connections between 4-HS and MATRIX ROW COLUMN MERGING. However, in that example, we used further techniques that are specific to MRCM to get better running times. Presumably, it would be also possible to improve on the running times of some of the algorithms described below by exploiting more problem-specific properties (if it is not clear that the problems are not exactly described by HS).

More specifically, we will encounter in this sections HITTING SET-like problems from the areas of computational biology and of network algorithms. Later in this chapter, further related problems will be discussed.

6.2.1 Problems from computational biology

Let us first mention here the following two problems arising in computational biology that can be effectively solved with the help of 3-HITTING SET, see [17, 45, 137]: MAXIMUM AGREEMENT SUBTREE and MAXIMUM COMPATIBLE TREE.

To formulate these problems, we need some more terminology: a *phylogeny* is a rooted tree T whose leaf set $L(T)$ is in bijection with a label set (with which it can be hence identified). Within MAXIMUM AGREEMENT SUBTREE, the rooted tree is always binary, i.e., all inner nodes but the root have degree three; the root has degree two.[1] To maintain this property, we assume that *label deletion* may propagate in the following sense:

- Assume that x is an inner node with children y and z, where z is a leaf that is going to be deleted.
- Upon deleting z, x would have a degree that is too small. Therefore, x is deleted as well. Two subcases arise:
  - x is the root of the tree. Then, y will be the root of the tree that is produced.
  - x is not the root of the tree. Then, x is the child of another node x', and an edge between x' and y is added in the tree that is produced to maintain it connected.

[1]We disregard trees with two or fewer nodes in this discussion to avoid messy formulations.

Let $T \setminus \{\lambda\}$ the tree obtained after deleting label λ from $L(T)$. We can extend this operation to sets of labels, arriving at the notation $T \setminus \Lambda$ for a set of labels Λ.

Problem name: MAXIMUM AGREEMENT SUBTREE (MAST)
Given: A set $\{T_1, \dots, T_n\}$ of binary rooted trees with equal label set $L = L(T_1) = \cdots = L(T_n)$
Parameter: a positive integer k
Output: Is there a set of labels $L' \subseteq L$, $|\Lambda| \leq k$, such that all trees $T_i \setminus \Lambda$ are isomorphic?

MAXIMUM AGREEMENT SUBTREE can be basically solved with the help of 3-HITTING SET by creating a hyperedge between three labels ℓ_1, ℓ_2, ℓ_3 if in one tree T_i the least common ancestor of ℓ_1 and ℓ_2 is closer to the root than the least common ancestor of ℓ_1 and ℓ_3, while in another tree T_j, the opposite is true: the least common ancestor of ℓ_1 and ℓ_3 is closer to the root than the least common ancestor of ℓ_1 and ℓ_2. The operation of deleting a label corresponds to putting the corresponding label into the hitting set.

The problem MAXIMUM COMPATIBLE TREE is more general, since it allows inner nodes of degree larger than three for a phylogeny. However, it is shown in [45] that this problem is still solvable with the help of 3-HITTING SET.

From the point of view of applications, also other, related problems seem to be of interest. For example, Berman, DasGupta and Sontag [43] considered the following problem (which we already rephrased in terms of HITTING SET) that was motivated by applications to reverse engineering of protein and gene networks:

Problem name: MULTI-HITTING SET (HSMULTI)
Given: A hypergraph $G = (V, E)$
Parameter: positive integers k, ℓ
Output: Is there a *multi-hitting set* $C \subseteq V$ with $|C| \leq k$, i.e., C satisfies $\forall e \in E \exists c \subseteq e (|c| \geq \ell \wedge c \subseteq C)$?

Again, if we restrict the size of the hyperedges to d, we can expect $\mathcal{FPT}$ results. More specifically, it is then sufficient to branch on all $\left(\begin{array}{c} d \\ \ell \end{array}\right)$ possibilities to cover a chosen hyperedge e (*candidate set* method). This immediately gives an $\mathcal{O}^*\left(\left(\begin{array}{c} d \\ \ell \end{array}\right)^{k/\ell}\right) = \mathcal{O}^*(d^k)$ algorithm for d-HSMULTI.

Better results can be expected in certain circumstances. For example, if $\ell = d - 1$, then we can simulate the fact that $d - 1$ out of the d elements of a hyperedge must be covered by the multi-hitting set by replacing that hyperedge by a clique (in the classical graph theoretic sense) on these d vertices and considering the translated graph as a VERTEX COVER instance. More specifically, we would use the program fragment listed as Alg. 46.

Algorithm 46 How to turn a special MULTI-HITTING SET instance into an equivalent VERTEX COVER instance

Input(s): $G = (V, E)$: the input hypergraph with parameter k

Output(s): $G' = (V', E')$: the output graph with parameter k' (the equivalent VC instance).

```
Let $V' := E' := \emptyset$
**for all** $e \in E$ according to increasing size **do**
  **if** $|e| < d - 1$ or $k < d - 1$ **then**
    Let $E' = \{vv'\}$ for arbitrary $v, v' \in V$.
    Let $k' = 0$.
    return $(G', k')$.
  **else if** $|e| = d - 1$ **then**
    Remove all vertices from $e$ from $V$ and from $E$.
    Remove empty hyperedges from $E$.
    Decrease $k$ by $d - 1$.
  **else**
    Add the vertices in $e$ to $V'$.
    Make these vertices a clique in $V'$ by accordingly updating $E'$.
    Remove $e$ from $E$.
  **end if**
**end for**
return $(G', k)$.
```

This shows that also reductions can be sometimes used to solve a specific problem in particular with the help of HITTING SET programs.

6.2.2 Call control problems

Th. Erlebach and his co-authors investigated variants of *call admission control problems*. In such a setting, we are given a set of connection requests in a communication network and have to determe which of the requests can be accepted without exceeding the capacity of the network. The goal is to maximize the number of accepted requests or, equivalently, to minimize the number of rejected requests. We follow the terminology as introduced in [23] in this section. In fact, most results are obtained in the quoted paper; however, we think this setting allows for even a certain number of nice examples where HITTING SET techniques can be fruitfully applied.

We restrict our discussion here to communication networks that can be described by undirected graphs. In [23], also networks that can be modeled by directed or (as a special case) bidirected graphs are treated. So, here a *communication network* is represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$. A *communication request* is a pair of vertices. If a request (u, v) is an *accepted request*, then a unit of bandwidth is reserved on all edges along a path $p((u, v))$ from u to v. Naturally, this means the remaining capacity decreases by one for each edge involved in $p((u, v))$, once this path has been chosen to satisfy request (u, v). Accordingly, a set of paths is a *feasible path set* if no edge $e \in E$ is contained in more than $c(e)$ paths. For a given set P of paths in a graph with edge capacities, an edge e is called a *violated edge* if the number of paths in P that contain e is greater than $c(e)$. In this setting, we can now formulate the basic *call admission* problem.

Problem name: CALLCONTROL
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ and a corresponding feasible path set $p(A)$ resulting from assigning to each request $r \in A$ some path $p(r)$ such that $p(A)$ is feasible and the set of *rejected requests* $R \setminus A$ contains no more than k elements?

(Recall that $P(G)$ collects all paths of G.) Unfortunately, there is no real hope to find efficient parameterized algorithms, since CALLCONTROL is $\mathcal{NP}$-hard, even for $k = 0$ and when we only consider unit capacities and when the input graphs are restricted to series-parallel graphs (a usually very "easy" class of graphs, where many graph problems that are $\mathcal{NP}$-hard in general can be efficiently solved). The basic reason is that, in the formulation of the preceding problem, we have an additional degree of freedom (in comparison to other similar parameterized problems), namely the choice of the path $p((u, v))$ to satisfy the request (u, v). This path assignment problem itself is closely related to the problem of finding disjoint paths and can therefore not efficiently solved.

Moreover, in some applications, this path is actually predetermined by a *path assigning function.* We can therefore formulate the following variant:

Problem name: CallControl-Pre
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Now, the (in principle) unbounded edge capacities actually make the problem hard in the parameterized sense. More technically speaking, in [23], showed that that problem is W[2]-hard, even when restricted to series-parallel graphs.

The picture changes if we further restrict the problem to instances with *bounded edge capacities*, i.e., assume there is a constant d such that, for all e, $c(e) \leq d$. We denote the problem as d-CallControl-Pre (to be specified formally below).

Now, there is a simple *candidate set* in each step of a simple search tree: namely, if there happen to be more request then capacity for a certain edge e, then take an arbitrary selection of $c(e)+1 \leq d+1$ paths through e. One of these paths (or better said one of the corresponding calls) must be rejected. This motivates Alg. 47 for solving d-CallControl-Pre.

Interestingly enough, the Alg. 47 (that is implicitly presented in [23]) assumes a sort of online generation of the candidate sets. Alternatively, one could think of trying to generate all possible candidate sets in the very beginning. Then, we would have all violating candidate sets at hand in the very beginning of the branching process. In fact, we would have produced a special instance of $(d+1)$-hitting set in this way. Then, we could take the best known parameterized off-line algorithm for hitting set, see Sec. 5.3.3. This would of course improve on the presented on-line algorithm which has a worst-case running time of $\mathcal{O}((d+1)^k n)$. Note that although the overall size of the produced d-hitting set instance is (alternatively) bounded by $\mathcal{O}(n^d)$ and is hence polynomial for any predetermined fixed d, this many (trial) conflict edges may be generated if we consider d really as fixed. However, as we already remarked in Sec. 3.4, it is usually the case in d-HS algorithms that they might work uniformly in the sense that both d and k can be viewed as parameters of the problem. If we like to get a similar behavior for this call control problem, we cannot afford spending so much time on the generation of the instance. So, formally we attack the following problem:

Algorithm 47 A simple search tree algorithm for d-CallControl-Pre, called online-CCP

Input(s): a graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ bounded by d, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$, a positive integer k

Output(s): YES if there is a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements or NO if no set of rejected requests of size at most k (making all the other requests feasible) exists.

```
violated:=NO
for all edges $e$ do
  if # paths through $e$ is greater than $c(e)$ then
    violated:=YES
  end if
end for
if violated then
  if $k > 0$ then
    pick violating edge $e$
    create candidate set $C$ consisting of $c(e) + 1$ paths
    found:=NO
    for all candidate paths $p \in C$ AND not yet found do
      create new instance $I'$, taking out $p$ and reducing $k$ by one
      found:=online-CCP($I'$)
    end for
    return found
  else
    return NO
  end if
else
  return YES
end if
```

Problem name: d-CallControl-Pre
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$
Parameter: a positive integer k, an edge capacity bound d
Output: Is there a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Let us therefore analyze our situation more carefully.

For example, it could well be that an edge has a *demand* $p(e)$ (measured in terms of the number of paths that want to use e) that is "much" larger than the capacity $c(e)$. This automatically means (by pigeon hole principle) that at least $p(e) - c(e)$ paths have to be removed. Therefore, the following reduction rule is safe:

Reduction rule 59 *If* $p(e) - c(e) > k$, *we can immediately reply* NO.

In terms of HS, such a situation corresponds to a sort of hyperclique. Observe that otherwise, each of the paths (that correspond to the vertices in the instance of HITTING SET) has "degree" $p(e)$.

To see if the solutions that can be found when solving CALLCONTROL-PRE or HS, let us be a bit more formal. We study how the information that certain paths are removed (i.e., vertices are put into the hitting set) is propagated in the branching process. Consider a CALLCONTROL-PRE instance I and the "corresponding" HS instance $HS(I)$, where the translation from I to $HS(I)$ can be described as follows:

- IF $G = (V, E)$ is the underlying network graph of I, then $P(G)$ is the vertex set of $HS(I)$.
- If $e \in E$ with $p(e) - c(e) > 0$, then let $P(e) \subseteq P(G)$ be the paths that contain e and put all $(c(e) + 1)$-element subsets of $P(e)$ as hyperedges into $HS(I)$. All hyperedges in $HS(I)$ are described this way.
- If k is the parameter of I, then k is also the parameter of $HS(I)$.

We have to show that if C is a path set whose removal resolves all conflicts in I, then C is a hitting set in $HS(I)$ and vice versa.

On the other hand, the instance that would have to be generated could be rather big, so that in the end the on-line variant could have an edge over the off-line variant. Namely, consider an edge e with $p(e) - c(e) = k$ many paths to be removed. This means that in $HS(I)$, e would "generate"

$$\binom{p(e)}{c(e)+1} = \binom{c(e)+k}{c(e)+1} \in \mathcal{O}((c(e)+1)^k)$$

many hyperedges, which are "too many" from a parameterized perspective.

More precisely, since we consider $c(e) \leq d$ for all edges, and since we translate a d-CALLCONTROL-PRE instance I into a $(d+1)$-HS instance $HS(I)$, this generated instance could be as large as $\mathcal{O}((d+1)^k)$ (even if we assume that we don't translate instances I that contain edges with $p(e) -$

$c(e) > k$, i.e., assume that the instance is reduced with respect to rule 59). Since any HITTING SET algorithm (even with kernelization preprocessing after arriving at the HS instance) needs time at least linear in the number of hyperedges, the possible use of sophisticated HS algorithms is not helpful in terms of lowering the running times, compared with algorithm online-CCP.

The idea to overcome this difficulty is based on the following observation:

Observation 6.5 *To resolve a conflict on edge e, at least $p(e) - c(e)$ paths have to be cancelled.*

The simple idea is to test all possibilities as indicated by the observation, i.e., all $(p(e) - c(e))$-element subsets of the set of all paths containing e, which is by definition of cardinality $p(e)$. How dear is this solution in terms of recursion? Since the parameter will be reduced by $p(e) - c(e)$, we arrive at

$$T(k) \leq \binom{p(e)}{p(e) - c(e)} T(k - (p(e) - c(e))).$$

If we simplify by considering the corresponding equation, omitting the argument e and setting $\Delta = p - c$, we arrive at

$$T(k) = \binom{p}{\Delta} T(k - \Delta).$$

With the ansatz $T(k) = \alpha(c, \Delta)^k$ we get

$$T(k) = \binom{c + \Delta}{\Delta}^{k/\Delta}.$$

Table 6.5: A table for branching on hypercliques

$c =$	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$
1	2	1.7321	1.5875	1.4954	1.4310
2	3	2.4495	2.1545	1.9680	...
3	4	3.1623	2.7145	...	
4	5	3.8730	...		
5	6	4.5926	...		

Rounded up to 4 decimal places after the decimal point, this yields the Table 6.5 for $\alpha(c, \Delta)$. This table indicates that already for relatively small values of Δ, the branching we obtain in this way is far better than any branching numbers we earlier obtained for HITTING SET. We replaced several numbers by dots because of the following monotonicity property, which also explains that "sooner or later" also in the case $c = 1$ (which corresponds to VERTEX COVER), this approach will beat any sophicated algorithm for VC. More precisely, for $c = 1$ and $\Delta = 9$, we have a branching number $\alpha(1, 9) \leq 1.2916$, for $\Delta = 10$, we get $\alpha(1, 10) \leq 1.2710$, and for $\Delta = 20$, $\alpha(1, 20) \leq 1.1645$.

Proposition 6.6 *For every fixed c, the sequence*

$$\left(\binom{c+\Delta}{\Delta}^{1/\Delta}\right)_{\Delta=1,2,3,\dots}$$

is strictly decreasing and converges to one.

Proof. Combine $\binom{c+\Delta}{\Delta} \in \mathcal{O}(\Delta^c)$ with $\lim_{\Delta\to\infty} \sqrt[\Delta]{\Delta} = 1$. ∎

Therefore, we can propose the off-line counterpart outlined in Alg. 48, whose running time (in terms of $\mathcal{O}^*$) coincides with that of the best corresponding algorithm for HITTING SET, since the size of the generated HS instance is only dependent on the capacity bound d (times k). In Alg. 48, the constant ϵ_d has to be chosen large enough to ensure that the first branchings are not worse than the ones incurred by the chosen $(d + 1)$-HITTING SET algorithms. We have therefore demonstrated the following result that strengthens the observations from [23].

Theorem 6.7 *d-CALLCONTROL-PRE can be solved in the same time bounds as the best corresponding algorithm for d-HITTING SET. In particular, this problem is in $\mathcal{FPT}$.*

Anand *et al.* describe some more $\mathcal{FPT}$ algorithms for call control problems. Let us mention two particular cases:

Problem name: CALLCONTROL IN TREES OF RINGS
Given: A communication network represented by an undirected graph $G = (V, E)$ that is a tree of rings with unit edge capacities, a set $R \subseteq V \times V$ of communication requests
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ and a path assigning function $p : R \to P(G)$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Algorithm 48 A still simple search tree algorithm for CallControl-Pre, called offline-CCP

Input(s): a graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ bounded by d, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$, a positive integer k

Output(s): YES if there is a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements or NO if no set of rejected requests of size at most k (making all the other requests feasible) exists.

```
found:=NO
for all edges $e$ and not yet found do
  if # paths $p(e)$ through $e$ is greater than $c(e) + \epsilon_d$ then
    if $p(e) - c(e) > k$ then
      return NO
    else
      $\Delta = p(e) - c(e)$
      for all path sets $P$ of size $\Delta$ that contain $e$ AND not yet found do
        create new instance $I'$, taking out $P$ and reducing $k$ by $\Delta$
        found:=offline-CCP($I'$)
      end for
    end if
  end if
end for
Transform the given instance $I$ into $HS(I)$.
{Let $(d+1)$-HS be the HITTING SET algorithm of your choice.}
return $(d+1)$-HS($HS(I)$)
```

The problem on bidirected trees of rings can be similarly treated, see [23]. We omit those details here.

It is shown in [23] that at most three rejection candidates can be found at each level of the recursion. This again "smells" like 3-HITTING SET. Let us see if we can again reduce this problem completely to 3-HS. To this end, we need to introduce some more notation (similar to [23]) to study CallControl in trees of rings in more depth.

- For any request (u, v) in a tree of rings, all undirected paths from u to v contain edges of the same rings.
- Hence, in any ring a path from u to v passes through, the vertex at which the path enters the ring (or begins) and the vertex at which the path leaves the ring (or terminates) is uniquely determined by u and v.

Hence, a path request in a tree of rings can be equivalently represented by a set of *subrequests* in all rings that the path passes through. Let us therefore study the problem in rings first.

Let R be a ring, drawn as a circle in the plane. Represent a subrequest (u, v) between two vertices on R as a straight line segment (called *chord*) joining u and v. Two chords are *parallel chords* if they intersect at most in points on R or if they are identical. A *cut* is a pair of edges of the graph on the ring. A request *cross*es a cut if each of the two possible paths connecting the endpoints of the request contains exactly one of the edges in the cut. In [23], the following characterization of routeability in rings is shown:

Lemma 6.8 *Given a ring R and a set of requests S in R, the requests in S can be routed along edge-disjoint paths iff*

- *all chords of the requests in S are pairwisely parallel and*
- *no cut is crossed by three (or more) requests.*

More precisely, Algorithm 49 can be used to find all routes if the conditions of the lemma apply. This algorithm can be read off the proof of the preceding lemma.

Algorithm 49 can be adapted to get an overall algorithm for the problem even in the presence of conflicts. More precisely, the first lines of Alg. 49 reflects the preprocessing that allows us to restrict our attention to subrequest on each ring.

According to Lemma 6.8, there are two reasons why a set of requests is not routeable in a ring:

1. There are two chords that are not parallel.

   In that case, one of the two corresponding requests must be deleted. Checking all possibilities gives hence a trivial $\mathcal{O}^*(2^k)$ running time for this part of the algorithm.

2. All chords in all rings are parallel, but there are rings that contain cuts which are crossed by three requests.

   As proposed by Anand *et al.*, there is a trivial $\mathcal{O}^*(3^k)$ algorithm for testing all possible request deletion for each violating cut. Since there cannot be more than $\mathcal{O}(n^3)$ many conflict sets in a ring that bears n subrequests, this part of the overall algorithm can be sped up by using the best available 3-HITTING SET algorithm.

Algorithm 49 A greedy algorithm for CallControl in trees of rings (assuming no conflicts), called greedy-CCTR

```
Input(s): a tree of rings $G = (V, E)$, a set $R \subseteq V \times V$ of communication
    requests
Output(s): a feasible routing of all requests
    {The overall routing can be reconstructed from the routings of the sub-
    requests.}
    for all rings $R$ do
      create list $L$ or parallel chords (corresponding to subrequests)
      {$L$ could contain multiple occurrences of the same subrequest, origi-
      nating from different requests.}
 5:   while $|L| > 1$ do
        choose two parallel chords $c$ and $c'$ from $L$ corresponding to subre-
        quests $u$ and $v$, as well as $u'$ and $v'$
        if $c = c'$ then
          route the corresponding subrequests arbitrarily
        else
10:       route $c$ and $c'$ in a unique fashion
        end if
        remove $c$ and $c'$ from $L$
      end while
      if $L$ is not empty then
15:     route $r \in L$ arbitrarily
      end if
    end for
```

In the leaves of the search tree, a postprocessing that actually finds routes in the now conflict-free case can be made (according to Alg. 49, lines 3–16). It is important to note that this postprocessing won't make any changes to the parameter value.

We can therefore state:

Theorem 6.9 CallControl in trees of rings *can be solved in time $\mathcal{O}(2.1788^k p(n))$, where p is some polynomial.*

Observe that it is important that in the first phase of the search tree, the seeming 2-hitting set instance is not solved by the best available algorithm for vertex cover, since the complexity of the subsequent 3-HS instance (to be solved in the second phase of the search tree processing) depends on the changes incidentally obtained by choosing one solution or other to the

2-HITTING SET instance in the first phase. In fact, we could see the first phase as (parameterized) enumeration of all (minimal) vertex covers of size up to k; this issue is discussed in greater detail in Chap. 8.

Is it always possible, say, to improve on algorithms that originate from candidate sets of size three (that are therefore akin to 3-HITTING SET) by actually translating such an instance into a 3-HITTING SET instance and then use one of the sophisticated algorithms for that problem? At least this is far from obvious.

Let us consider the following problem (which is in fact a reparameterization of the call control problems we treated up to now):

Problem name: CALLCONTROL IN TREES WITH CAPACITIES ONE OR TWO (CALLCONTROL IN TREES 1-2)
Given: A communication network represented by an undirected graph $G = (V, E)$ that is a tree with $E_i \subseteq E$ being the edges of capacity $i \in \{1, 2\}$, a set $R \subseteq V \times V$ of communication requests, a positive integer k
Parameter: $\ell = |E_2|$
Output: Is there a subset $A \subset R$ and a path assigning function $p : R \to P(G)$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

As mentioned in [23], the problem with $\ell = 0$ is solvable in polynomial time. Hence, ℓ can be seen as a measure of the *distance from triviality* (here: polynomiality), as proposed in [220] as a possible way of parameterizing problems, since the problem is $\mathcal{NP}$-hard if ℓ can arbitrarily vary.

Since in a tree, a route between two points is uniquely determined, a call control problem in trees with bounded edge capacities is a sort of special case of CALLCONTROL-PRE; hence, if parameterized with k, this problem has already been treated above. However, a reasoning along the lines leading to Alg. 47 shows the following:

Theorem 6.10 CALLCONTROL IN TREES OF RINGS *can be solved in time* $\mathcal{O}(3^\ell p(n))$*, where* p *is some polynomial.*

Of course, one could again try to translate this problem into 3-HITTING SET. However, we have to be careful with the postprocessing that needs to be done in the leaves: there, we still have to find an optimal solution with the mentioned "tree algorithm." It might be that (by optimizing the 3-HITTING SET algorithm) we have destroyed a possibility that needs to be regarded when looking at E_1. Therefore, we cannot avoid enumerating all possibilites in the search tree, see Chap. 8 for a thorough discussion.

6.3 Graph modification problems

Graph modification problems are abundant and turn up in many disguises. The main idea is if it is possible to change a given graph by a number k of certain allowed *graph edit operation*s like *vertex deletion*, *edge deletion*, or *edge adding* into a graph of a prescribed type.

For example, VERTEX COVER can be viewed as a graph modification problem, the task being to find a set of at most k vertices whose removal turns the given graph G into a graph without any edges.

With adding certain constraints, also maximization problems can be created that may be viewed as graph modification problems. Typically, the question is then if at least k edit operations suffice to get a graph of a prescribed type, where the constraints restrict the way in which the edit operations may be performed.

6.3.1 A maximization graph modification problem

Consider for example the following problem:[2]

Problem name: MAXIMUM MINIMAL VERTEX COVER (MMVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a minimal vertex cover set of cardinality $\geq k$?

The additional constraint is here the minimality of the considered vertex covers. This problem can therefore be equivalently formulated as follows: Given a graph $G = (V, E)$ and an integer k, is it possible to find a vertex set M, $|M| \geq k$, whose removal produces a graph $G[V \setminus M]$ with empty edge set, such that for all $x \in M$ there is a $y \in V \setminus M$ with $\{x, y\} \in E$?

In fact, it is pretty easy to design an $\mathcal{FPT}$ algorithm for MAXIMUM MINIMAL VERTEX COVER, based on the simple first search tree algorithm we presented for VERTEX COVER, since still we are looking for a vertex cover, i.e., every edge must be covered. The only difference is how we treat the leaves of the search tree. After having put k vertices in the cover along a recursion path, we have to check if possibly some vertices could be added to produce a minimal vertex cover; so the point is to check the minimality assumption rather than the cover property at this point. The corresponding algorithm is listed in Alg. 50. Initially, the algorithm is called with the parameters $(G, k, \emptyset)$, where (G, k) is the instance of MMVC under scrutiny.

[2]We are not aware of any publication of the results of this subsection elsewhere.

Algorithm 50 A simple search tree algorithm, called MMVC-simple

Input(s): a graph $G = (V, E)$, a positive integer k, a partial solution $M \subseteq V$

Output(s): YES if there is a minimal vertex cover $C \subseteq V$, $|C| \geq k$, $M \subseteq C$ or
NO if no vertex cover of size at least k exists.

```
if $k \leq 0$ then
  if $M$ can be enlarged to get a minimal vertex cover then
    return YES
  else
    return NO
  end if
else if $E = \emptyset$ then
  return NO
else
  Choose edge $e = \{x, y\} \in E$
  if MMVC-simple($G - x, k - 1, M \cup \{x\}$) then
    return YES
  else
    return MMVC-simple($G - N[x], k - \deg(x), M \cup N(x)$)
  end if
end if
```

Can we further improve on that algorithm? This might be done, indeed, by providing proper data structures to simply detect non-minimality "on the way" of the branching process itself. To this end, we may start with a set collection $\mathcal{C} = \{N[v] \mid v \in V\}$. This collection should always maintain the property that, whenever $\emptyset \in \mathcal{C}$ shows up in one branch, we can stop the branching process, since we can no longer obtain a minimal cover on this branch. Hence, if we take a vertex v into the vertex cover, we would delete v from every set $S \in \mathcal{C}$, and in the branch that don't takes v into the cover but its neighbors, we would first delete all sets $S \in \mathcal{C}$ that contain v (because all these sets have now a member that is *not* in the cover, hence ensuring minimality), and then we would delete every $x \in N(v)$ from every set $S \in \mathcal{C}$. The leaves of the corresponding search tree would never correspond to partial covers that cannot be extended to minimal vertex covers of the whole graph. However, we were not able to prove better running times for a search tree algorithm using such additional information, even at some stage the graph contains only vertices of degree one. The problem is that still vertices might depend on one another due to the collection $\mathcal{C}$.

Recall that we have derived a small kernel for MAXIMUM MINIMAL VERTEX COVER in Chap. 4, so that we can state:

Theorem 6.11 MAXIMUM MINIMAL VERTEX COVER *can be solved in time* $\mathcal{O}(2^k + p(n))$, *where* p *is some polynomial.*

6.3.2 Graph modification problems related to HS

In this section, we discuss several graph modification problems mentioned in [211]. In [211, Sec. 4.1.3], the following problem is mentioned:

Problem name: TRIANGLE EDGE DELETION (TED)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an edge set $C \subseteq E$ with $|C| \leq k$ whose removal produces a graph without triangles as vertex-induced subgraphs?

Observe that three vertices $\{x, y, z\}$ are a triangle subgraph iff the graph induced by these three vertices is a triangle, while in general terms there is a difference between subgraphs and vertex-induced subgraphs.

A TRIANGLE EDGE DELETION instance $G = (V, E)$ can be transformed into an 3-HITTING SET instance G' as follows:

- Interpret the edges E as vertices of G'.
- A hyperedge $h = \{e_1, e_2, e_3\}$ is contained in G' iff, for some $x, y, z \in V$, $e_1 = \{x, y\}$, $e_2 = \{y, z\}$ and $e_3 = \{x, z\}$, and $h \subseteq E$.

It is straightforward to see that the removal of $C \subseteq E$ produces a graph $G - C$ that does not contain any vertex-induced triangle iff C is a hitting set of G'.

Moreover, the currently best algorithms are based on 3-HITTING SET, so that the algorithm presented in Section 5.3.3 automatically improves the constants as presented in [211].

Corollary 6.12 TRIANGLE EDGE DELETION *can be solved in time* $\mathcal{O}(|E(G)|^3 + 2.1788^k)$, *given an instance* G.

Proof. According to what we earlier observed, we only have to transform G into a 3-HITTING SET instance. This transformation takes time $\mathcal{O}(|V(G)|)$, since we have to cycle through all triples of vertices that might form a triangle. Then we can run a 3-HITTING SET algorithm to get the claimed running time.
∎

There are three more vertex deletion problems mentioned in [211] that we incidentally improve on:

Problem name: TRIANGLE VERTEX DELETION (TVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a graph without triangles as vertex-induced subgraphs?

Corollary 6.13 TRIANGLE VERTEX DELETION *can be solved in time* $\mathcal{O}(|V(G)|^3 + 2.1788^k)$*, given an instance* G*.*

Problem name: CLUSTER VERTEX DELETION (CVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a graph being a union of vertex-induced cliques?

Here, it might not be immediately clear how to apply the proposed methodology. However, we have the following forbidden subgraph characterization also mentioned in [351]:

Theorem 6.14 *A graph* G *is a* cluster graph*, i.e., a forest of cliques, iff no selection of three vertices from* G *induces a path of length two in* G*.*

We can hence conclude:

Corollary 6.15 CLUSTER VERTEX DELETION *can be solved in time* $\mathcal{O}(|V(G)|^3 + 2.1788^k)$*, given an instance* G*.*

Observe, however, that we cannot (completely) use the methodoloy of reducing to HITTING SET problems for the edge deletion analogue of CLUSTER VERTEX DELETION, since the deletion of edges might also *create* induced paths of length two. Hence, we would need a problem-specific analysis to improve over the trivial $\mathcal{O}^*(2^k)$ search tree for that problem (where, for each induced path of length two, one of the two involved edges is deleted in each branch); this has been done in [211] to obtain a rather complicated $\mathcal{O}^*(1.53^k)$ search tree algorithm. A similar effect is encountered in the edge deletion problem CLIQUE COMPLEMENT COVER discussed below.

To understand the following problem from [211], we need some more terminology.

A *cograph* (or *complement-reducible graph*) is a graph defined by the three criteria:

1. K_1 (i.e., a graph having only one vertex) is a cograph,
2. If G is a cograph, then so is its graph complement G^c, and
3. If G and G' are cographs, then so is their graph union.

According to the entry `http://mathworld.wolfram.com/Cograph.html` in Mathworld (see [Chapter 11, Site 14]), there is one criterion characterizing cographs that is especially interesting to us:

Theorem 6.16 *G is a cograph iff G does not contain any set of four vertices that induce a path.*

Another interesting property of these types of graphs is that some hard problems can be efficiently solved when restricted to cographs. But let us now formulate the graph modification problem:

Problem name: COGRAPH VERTEX DELETION (CoVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a cograph ?

We can hence conclude:

Corollary 6.17 COGRAPH VERTEX DELETION *can be solved in time $\mathcal{O}(|V(G)|^3 + 3.1150^k)$, given an instance G.*

6.3.3 CLIQUE COMPLEMENT COVER: an easy problem?

The idea of relating graph modification problems with HITTING SET is often helpful, but not always leading to the applicability of the best known HITTING SET algorithms. This will be exemplified by the next two problemwe consider: CLIQUE COMPLEMENT COVER. Let us first recall CCC from Chap. 2:[3]

Problem name: CLIQUE COMPLEMENT COVER (CCC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *clique complement cover* $C \subseteq E$ with $|C| \leq k$?

[3]The results of this subsections were not published elsewhere.

Bar-Yehuda [34] found that this problem is closely linked to VC: namely, he exhibited a simple factor-2-approximation algorithm which is based on the following observations:
Observation 1: If e_1 and e_2 are two edges with endpoints $x \in e_1$ and $y \in e_2$ that are not neighbors, then e_1 and e_2 cannot participate in the same complete subgraph.

We call two edges which have at least two non-neighbor endpoints x, y (as described) *conflicting edges*.
Observation 2: A graph containing no conflicting edges is a clique.

This immediately yields the search-tree algorithm outlined in Alg. 51.

Algorithm 51 A search tree algorithm, called CCC-st
Input(s): a graph $G = (V, E)$, a positive integer k
Output(s): YES iif there is a clique complement cover $C \subseteq V$, $|C| \leq k$

```
if $k < 0$ then
  return NO
else if $G$ is a complete graph then
  return YES
else
  pick two conflicting edges $e_1$ and $e_2$
  {these edges must exist due to Observation 2}
  if CCC-st($G - e_1, k - 1$) then
    return YES
  else
    return CCC-st($G - e_2, k - 1$)
  end if
end if
```

This algorithm is very similar to the straightforward search-tree algorithm for VERTEX COVER as first observed by Mehlhorn in 1984, see Alg. 16. We can therefore infer:

Lemma 6.18 CCC *can be solved in time $\mathcal{O}(2^k|E|^2)$.*

Why can't we simply translate a given CLIQUE COMPLEMENT COVER instance into an "equivalent" VERTEX COVER instance and use the very clever branching algorithms developed for VC by using the auxiliary graph whose vertices are the edges of the original graph instance and whose edges reflect the "conflicting edges" relation as defined above? The problem is that the very branching might create new conflicting edges that did not appear before; have a look at the following example:

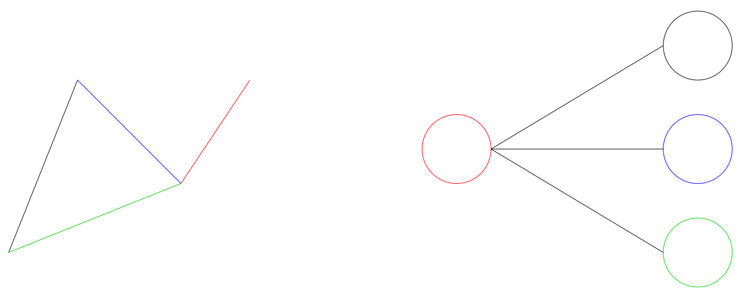


Figure 6.1: The left-hand side shows a small CLIQUE COMPLEMENT COVER instance, with the corresponding conflict graph depicted on the right-hand side.

Example 6.19 In Fig. 6.1, an instance of CLIQUE COMPLEMENT COVER is shown with four vertices and four edges. Obviously, an optimal solution is to take away the red edge. However, our search tree algorithm would alternatively test the possibility to take out the blue edge; this is also to be seen on the right-hand side of the figure where the conflict graph is drawn. However, if the blue edge is taken out, this would of course also solve the conflict graph, but this would be no valid solution of the original CLIQUE COMPLEMENT COVER instance. In fact, the branching algorithm would possibly continue on that branch if the branching actually started with the blue edge and the parameter budget would be high enough.

We have not bothered to further lower the constants in the search tree algorithm nor with providing a kernelization. However, observe that Buss' rule can be applied to the conflict graphs that can be constructed in each step of the branching process. We can also transfer the rules that get rid of vertices of degree zero and one to CCC. The main observation here to show the validity of such reduction rules is the following: destroying a not necessarily induced K_1 or K_2 in the conflict graph corresponds to possibly destroying a particular K_1 or K_2 as "target clique" in the original graph; these can be easily replaced by other tiny cliques if necessary. Observe that we cannot claim to actually have found a small kernel for the original problem. However, this can be shown with some additional effort. Actually finding a small kernel is left to the reader in this case.

As a specific rule for CLIQUE COMPLEMENT COVER, we may use the fact that vertices of degree larger than k must all belong to the clique that should remain, since it is too dear to separate them from the rest. Hence, whenever the set

$$\{v \in V \mid \deg(v) > k\}$$

does not induce a clique in the given instance G, then we can answer NO.

Hence, an $\mathcal{O}(1.6182^k + p(n))$ algorithm is attainable by the triviality first principle.

Let us finally remark that Bar-Yehuda (as well as Hochbaum) consider [34, 36, 232] also weighted versions of CLIQUE COMPLEMENT COVER (and of other similar problems that can actually be treated by similar techniques); weighted versions formed in analogy to WEIGHTED VERTEX COVER can be also solved in time $\mathcal{O}^*(2^k)$. Note that the quoted reference mostly deal with finding approximation algorithms. In the case of CLIQUE COMPLEMENT COVER, the approximation algorithm also bears some similarity with the well-known approximation algorithms for VERTEX COVER. However, it is noteworthy that the simplest 2-approximation algorithm for VERTEX COVER (that does *not* transfer to the weighted case) that simply takes both vertices incident to a "current" edge into the cover (while the search tree algorithm 16 we considered would branch on the two possibilities) does not transfer to CLIQUE COMPLEMENT COVER due to the effect described in our example: deleting both the red and blue edge would be no valid solution to the problem instance.

Let us mention finally that there are also other graph operations that qualify as graph modifications. For example, consider the following problem that is mentioned in [347]:

Problem name: BIPARTIZATION, REPLACING EDGES BY 2-PATHS VARIANT (BPEDGEVAR)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *bipartization set* $C \subseteq E$ with $|C| \leq k$ such that replacing each edge in C by a path of length two produces a bipartite graph?

In fact, it is not hard to see that BPEDGEVAR is equivalent to BIPARTIZATION, EDGE VARIANT in the following strong sense:

Lemma 6.20 *(G, k) is a* YES *instance of* BPEDGEVAR *iff (G, k) is a* YES *instance of* BIPARTIZATION, EDGE VARIANT.

How to actually solve BIPARTIZATION, EDGE VARIANT in $\mathcal{FPT}$ time will be explained in Chap. 8.

It should have been made clear by now that many graph modification problems are indeed in $\mathcal{FPT}$. This is true in quite general terms as long as a finite forbidden subgraph characterization is available. In general terms, this has been addressed by Cai [67], also see [134, Sec. 3.2.2]. More candidate problems can be found in [302], where the question of $\mathcal{FPT}$-membership of these problems is explicitly raised. The list of references given there also provides a good starting point for further studies.

6.4 Graph drawing problems

The problems that we shall study in this section can be paraphrased as follows:

> Given a graph G, try to draw G in the plane in a specific way such the drawing looks as good as possible.

The quality of a drawing is usually measured by estimating the distance between a specific drawing and the case of a "perfect drawing" (amongst all graphs, disregarding the specific input graph); at least, it is expected that a perfect drawing of some graph has a minimal measure amongst all graphs that have the same number of edges or vertices (depending on the concrete problem).

For more background to the problems under study, we can generally refer to [283, Chapter 5], a chapter written by O. Bastert and C. Matuszewski with the title "Layered Drawings of Digraphs."

6.4.1 LINEAR ARRANGEMENT again...

In this context, let us revisit two problems we already came across in earlier chapters:[4]

Problem name: LINEAR ARRANGEMENT (LA)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that
$$\sum_{\{u,v\}\in E} |\sigma(u) - \sigma(v)| \leq k\ ?$$

[4]The contents of this subsection did not yet appear elsewhere but has been accepted for presentation at CTW 2005.

Problem name: LINEAR ARRANGEMENT (LA), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \ldots, |V|\}$ such that

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| \leq k + |E|?$$

Observe that the expression

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)|$$

can be seen as measuring the distance between the linear drawability of $G = (V, E)$ and that of the graph $G = (V, \emptyset)$ (where the measure would become 0); again, the modified problem (parameterizing about guaranteed values) is a bit more meaningful, since

$$\sum_{\{u,v\} \in E} (|\sigma(u) - \sigma(v)| - 1)$$

becomes zero for all graphs that are collection of paths (and only for those), so that that expression can be viewed as measuring the distance of an input graph G from a forest of paths. As we have already seen, the path-similarity can be alternatively measured in terms of pathwidth and bandwidth. Another related problem is the following graph modification problem:

Problem name: LINEAR ARRANGEMENT BY DELETING EDGES (LADE)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an edge set E' with $|E'| \leq k$ and a one-to-one mapping $\sigma : V \to \{1, \ldots, |V|\}$ such that

$$\sum_{\{u,v\} \in E \setminus E'} |\sigma(u) - \sigma(v)| = |E \setminus E'| \ ?$$

So, the task is to find a minimum number of edges whose removal turns the given graph into a *forest of paths*.

Let us start our considerations in this section with the last problem, since it is intimately related to the graph modification problems that we considered earlier.

In order to develop a search tree algorithm for this problem, we should ask ourselves: what are possible candidate sets to branch on? In other words, what makes a graph a forest of paths?

A helpful observation is the following one:

Lemma 6.21 *A graph of maximum degree two is a collection of path and of cycle components.*

This lemma already provides us with a simple branching idea: whenever we find a vertex of degree three (or larger) in a graph, take three of the incident edges to form a candidate set: one of these edges must be removed so that the (modified) graph can satisfy the condition of the lemma.

After having done this sort of branching, we are left with path and cycle components. Obviously, it does not matter which of the edges in a cycle component we take out to turn the cycle into a path, so that the left-over graph can be dealt with in polynomial time. This justifies Alg. 52. Note that this algorithm is very similar to the one suggested by Dujmovič *et al.* in [143] for the related problem ONE-LAYER PLANARIZATION.

Algorithm 52 A search tree algorithm solving LADE, called LADE-ST

Input(s): a graph $G = (V, E)$, an integer k

Output(s): YES iff the given LADE instance has a solution

```
if $k < 0$ then
  return NO.
end if
Let $v$ be a vertex of $G$ of maximal degree.
if $\deg(v) \leq 2$ then
  return $k \geq$ #cycle components of $G$
else
  Take three edges $e_1, e_2, e_3$ with $\{v\} = e_1 \cap e_2 \cap e_3$.
  for all $i = 1, 2, 3$ do
    if LADE-ST$((V, E \setminus \{e_i\}), k - 1)$ then
      return YES
    end if
  end for
  return NO
end if
```

Theorem 6.22 LINEAR ARRANGEMENT BY DELETING EDGES *is $\mathcal{NP}$-complete but in $\mathcal{FPT}$. More specifically, it can be solved in time $\mathcal{O}(3^k|G|)$.*

Proof. Membership in $\mathcal{FPT}$ can be seen (including the claimed running time) by Alg. 52. Since the search can be viewed as a nondeterministic guessing process, membership in $\mathcal{NP}$ is also clear.

$\mathcal{NP}$-hardness can be seen via a rather standard reduction from 3-SAT [199]:

Let F be a collection of clauses, each of size three, containing literals formed from the n variables of the given 3-SAT instance.

The corresponding instance (G, k) of LINEAR ARRANGEMENT BY DELETING EDGES is formed as follows: for each variable x, we introduce four vertices x^0, x^1, x^2, and x^3, so that G has $4n$ vertices. Furthermore, G contains the following edges (and only those):

- For each variable x, introduce the edges x^1x^2, x^2x^3, x^0x^2. Hence, $G[\{x^0, x^1, x^2, x^3\}]$ is a star with center x^2.
- For each clause $c = \ell_1 \vee \ell_2 \vee \ell_3$ with $\ell_i \in \{x_i, \bar{x}_i\}$, introduce four edges according to the following prescription.

  If $\ell_i = x_i$, let $c_i^1 := x_i^1$ and $c_i^2 := x_i^2$ be the connection points of x_i, and if $\ell_i = \bar{x}_i$, let $c_i^1 := x_i^2$ and $c_i^2 := x_i^3$ be the connection points of x_i. Then, introduce the following edges, so that the connection points form a cycle: $c_1^2c_2^1$, $c_2^2c_3^1$, $c_3^2c_1^1$.

Finally, let $k := n$.

The claim is that (G, n) has a solution iff the original 3-SAT instance is solvable. Namely, if $\alpha : X \to \{0, 1\}$ is an assignment that satisfies F, then it is readily seen that taking out the following n edges will turn G into a forest of paths: take out x^1x^2 if $\alpha(x) = 1$ and take out x^2x^3 if $\alpha(x) = 0$.

Conversely, the three edges that are introduced into G for each variable x guarantee that in any solution, one of these three edges has to be removed (since x^2 has degree three). Hence, there cannot be any feasible solution to G that takes out less than n edges, and any solution that takes out n edges will take out one edge from the edges x^1x^2, x^2x^3, x^0x^2 for each variable x. It is clearly true that a feasible solution to G that takes out n edges can be turned into a satisfying assignment of F as follows: if x^1x^2 is taken out, set $\alpha(x) = 1$; otherwise, set $\alpha(x) = 0$. ∎

Can we further improve on the running time of Alg. 52? It is of course tempting to transform a LINEAR ARRANGEMENT BY DELETING EDGES instance into an instance of 3-HITTING SET, by considering candidate sets as hyperedges. The problem with this approach lies in the fact that it might be the case that a seemingly worse solution to 3-HITTING SET actually is the one to take into consideration for the original LINEAR ARRANGEMENT BY DELETING EDGES instance, since that solution might incidentally destroy more (or create less) cycle components than an optimum solution to the "corresponding" 3-HITTING SET. However, there is good hope that the ideas that led to the improvements for the search tree analysis of 3-HITTING SET can be partially transferred to this case, as well. In fact, we will discuss a similar situation with ONE-LAYER PLANARIZATION below.

Let us now turn our attention towards LINEAR ARRANGEMENT. We already know of kernelization algorithms that basically tell us that there are no more than $2k$ vertices in a reduced graph (we formulated this as having at most k edges, but then the claim readily follows). We even obtained a kernelization scheme that allows us to get a $(1+\epsilon)k$ kernel (measured in terms of the number of vertices), see Chap. 3. This allows us to conclude:

Corollary 6.23 LINEAR ARRANGEMENT *can be solved in time* $\mathcal{O}(((1+\epsilon)k)! + p_\epsilon(|G|)$*, where* p_ϵ *is a polynomial whose degree depends on the choice of* $\epsilon \leq 1$*.*

Proof. After the kernelization phase, we simply test all possible combinations of the remaining vertices. ∎

So, if we are going to design a search tree algorithm for LINEAR ARRANGEMENT, we should try to arrive at a $\mathcal{O}^*(c^k)$ algorithm to beat the naive "test-all-combinations" algorithm sketched above.

What is a good starting point for a search tree algorithm development for LA ? Obviously, each edge has to "pay" at least one unit for being drawn in whatever direction. At the leaves of such a search tree, every edge would have got an orientation, i.e., our input graph would have become a directed graph. Unfortunately, there might be various *topological ordering*s of this directed graph, and it is not clear which of them minimizes our LA criterion.

In fact, the following problems are known to be $\mathcal{NP}$-hard [199, GT 43]:

Problem name: DIRECTED LINEAR ARRANGEMENT (DLA)
Given: A directed graph $G = (V, A)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ that respects the orientation of G, i.e., $\sigma(u) < \sigma(v)$ whenever $(u, v) \in A$, such that

$$\sum_{(u,v)\in A} |\sigma(u) - \sigma(v)| \leq k \ ?$$

Problem name: DIRECTED LINEAR ARRANGEMENT (DLA), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A directed graph $G = (V, A)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ that respects the orientation of G such that

$$\sum_{(u,v)\in A} |\sigma(u) - \sigma(v)| \leq k + |A| \ ?$$

More specifically, the sketched search tree algorithm for LINEAR ARRANGEMENT has (as of yet) reduced the original instance to an instance $(G, k, \prec)$ that can be viewed as an instance of DIRECTED LINEAR ARRANGEMENT, parameterized above guaranteed value, see Alg. 53. The procedure LA-ST is called with $(G, k, \emptyset)$ at the beginning, where (G, k) is the instance of LINEAR ARRANGEMENT we like to investigate.

Algorithm 53 A search tree algorithm solving LA, called LA-ST

Input(s): a graph $G = (V, E)$, an integer k, a partial ordering $\prec$ on V
Output(s): YES iff the given LA instance has a solution

```
Determine the edges that are settled by transitivity and adjust $\prec$ and $k$
accordingly.
if $k < 0$ or $\prec$ contains both $(x, y)$ and $(y, x)$ then
  return NO.
else if $\exists\{x, y\} \in E$ : neither $x \prec y$ nor $y \prec x$ is settled then
  if LA-ST$(G, k - 1, \prec \cup \{(x, y)\})$ then
    return YES
  else
    return LA-ST$(G, k - 1, \prec \cup \{(y, x)\})$
  end if
else
  return DLAgv-ST$((V, \prec), k, \emptyset)$
end if
```

In Alg. 53, $(G, \prec)$ should denote the directed acyclic graph that corresponds to G, when the arcs of $(G, \prec)$ are oriented so that they respect the ordering $\prec$.

Lemma 6.24 *Alg. 54 correctly solves* DIRECTED LINEAR ARRANGEMENT, *parameterized above guaranteed value.*

Algorithm 54 A search tree algorithm solving DLA, parameterized above guaranteed value, called DLAgv-ST

Input(s): a directed acyclic graph $G = (V, A)$, an integer k, a partial ordering $\prec$ on V that respects the arc order, i.e., $u \prec v \implies (v, u) \notin A$.

Output(s): YES iff the given DLA instance has a solution

```
    Determine the edges that are settled by transitivity and adjust $\prec$ and $k$ accordingly.
    if $k < 0$ or $A \cup \prec$ contains both $(x, y)$ and $(y, x)$ then
       return NO.
    else if $\exists x, y \in V, x \neq y$: neither $(x, y) \in A \cup \prec$ nor $(y, x) \in A \cup \prec$, and $\exists u, v \in V \setminus \{x, y\}$ (possibly with $u = v$): either $(u, x) \in A \wedge (v, y) \in A$ or $(x, u) \in A \wedge (y, v) \in A$ then
 5:    if DLAgv-ST$(G, k - 1, \prec \cup \{(x, y)\})$ then
          return YES
       else
          return DLAgv-ST$(G, k - 1, \prec \cup \{(y, x)\})$
       end if
10: else if $\exists x, y \in V, x \neq y$ : neither $(x, y) \in A \cup \prec$ nor $(y, x) \in A \cup \prec$, but $x, y$ belong to the same (weak) component in $G$ then
       if $A \cup \prec \cup \{(x, y)\}$ is a partial ordering on $V$ then
          return DLAgv-ST$(G, k, \prec \cup \{(x, y)\})$
       else
          return DLAgv-ST$(G, k, \prec \cup \{(y, x)\})$
15:    end if
    else if $\exists x, y \in V, x \neq y$ : neither $(x, y) \in A \cup \prec$ nor $(y, x) \in A \cup \prec$ then
       return DLAgv-ST$(G, k, \prec \cup \{(x, y)\})$
    end if
```

Proof. The interesting part is here to see that unresolved orderings between vertices can be deterministically resolved "in the end" (i.e., in lines 10–17 of Alg. 54) for $x, y \in V, x \neq y$ if neither $(u, x) \in A \wedge (v, y) \in A$ nor $(x, u) \in A \wedge (y, v) \in A$ without loosing optimality. There are two principal cases:

1. x, y pertain to the same (weak) component of G.
2. x, y belong to different components of G.

The second case can be arbitrarily resolved due to Lemma 3.4; observe that this case will only show up if the first case does not apply anywhere, which means that all components have been already linearly ordered.

To prove the correctness of the first case, we settle a couple of claims.

Claim 0: There exists $u, v \in V \setminus \{x, y\}$ (possibly $u = v$) such that u is connected by an arc to x and v is connected by an arc to y.
Proof of Claim. Since there is no arc connecting x and y but both belong in the same weak component, the claim is true. $\Diamond$

In the following, let us fix such $u, v \in V$ according to Claim 0. Since the previous If-branch is not applicable, we have either $(x, u) \in A$ and $(v, y) \in A$, or $(u, x) \in A$ and $(y, v) \in A$. For reasons of symmetry, we only discuss the first of these possibilities in what follows.
Claim 1: Either $(x, v) \in A$ or $(v, x) \in A$.
Proof of Claim. Otherwise, the previous If-branch would have been applicable, since we have neither $(x, v) \in A$ nor $(v, x) \in A$ but $(x, u) \in A$ and $(v, y) \in A$. $\Diamond$

Symmetrically, one can show:
Claim 1': Either $(y, u) \in A$ or $(u, y) \in A$.
This leaves us with the following possibilities:

1. $(x, u) \in A$, $\underline{(v, y) \in A, (x, v) \in A}$, (the relation between y and u is arbitrary): the underlined relations imply $(y, x) \in \prec$, and this has been found by the transitivity check at the very beginning of the routine, so that this case cannot occur at this stage.

2. $(x, u) \in A$, $(v, y) \in A$, $(v, x) \in A$, $(y, u) \in A$:

   There are no arcs leading into x and no arcs emanating from y, since otherwise the previous If-branch would have been applicable. Hence, there is only one way of ordering x and y without (locally) incurring additional costs, and this ordering $y \prec x$ does not interfere with other connections due to the previous sentence.

3. $\underline{(x, u) \in A}$, $(v, y) \in A$, $(v, x) \in A$, $\underline{(u, y) \in A}$: the underlined relations imply $(x, y) \in \prec$, so that this case cannot occur at this stage.

∎

Observe that Alg. 54 can be also used to solve the problem when parameterized in the standard way. Namely, one could call DLAgv-ST$(G, k - |A|, \emptyset)$, when $(G = (V, A), k)$ is the given DLA instance.

Theorem 6.25 *The problems* LINEAR ARRANGEMENT, DIRECTED LINEAR ARRANGEMENT *(the latter parameterized either in the standard way or above guaranteed value) are solvable in time $\mathcal{O}(2^k |G|)$.*

Proof. The fact that each of the two branches listed in Alg. 53 reduces the parameter by one is clear, since each edge has to be settled and produces a cost of at least one.

The case where we actually branch in Alg. 54 is a bit more involved.

Consider the case that the relation between x and y is still unsettled and that $(u, x) \in A$ and $(v, y) \in A$ (the other case is completely symmetric). Then, the length of the path that corresponds to (u, x) in the linear embedding was taken into account as of yet only excluding the possible position of y, and the same holds true for the path that corresponds to the arc (v, y) with respect to the position of x. If we now settle $x \prec y$, then the path corresponding to (v, y) will necessarily be one unit greater than our previous estimate. Accordingly, settling $y \prec x$ will enlarge our estimate of the path corresponding to (u, x) by one unit. So, in both cases we can safely reduce the parameter by one. ∎

6.4.2 ONE-SIDED CROSSING MINIMIZATION

Let us now turn to another problem that can be viewed, to a certain extent, as a linear arrangement problem, as well, see [350] for a formal treatment of these relations.

Problem name: ONE-SIDED CROSSING MINIMIZATION (OSCM)
Given: A bipartite graph $G = (V_1, V_2, E)$ and a linear order $\prec_1$ on V_1.
Parameter: a positive integer k
Output: Is there a linear order $\prec$ on V_2 such that, when the vertices from V_1 are placed on a line (also called layer) L_1 in the order induced by $\prec_1$ and the vertices from V_2 are placed on a second layer L_2 (parallel to L_1) in the order induced by $\prec$, then drawing straight lines for each edge in E will introduce no more than k edge crossings?

The material of this section is joint work with V. Dujmović and M. Kaufmann, see [144]. From an algorithmic perspective, it is worthwhile remarking that OSCM can be solved in polynomial time if the input graph is a tree [350], and the quoted (non-trivial) algorithm relies on the polynomial solvability of LINEAR ARRANGEMENT on trees [352].

ONE-SIDED CROSSING MINIMIZATION is the key procedure in the well-known layout framework for layered drawings commonly known as the Sugiyama algorithm. After the first phase (the assignment of the vertices to layers), the order of the vertices within the layers has to be fixed such that the number of the corresponding crossings of the edges between two adjacent layers is minimized. Finally, the concrete position of the vertices within the layers is determined according the previously computed order. The crossing minimization step, although most essential in the *Sugiyama approach* to layered graph drawing, is an $\mathcal{NP}$-complete problem. The most commonly used method is the layer-by-layer sweep heuristics where, starting from $i = 1$, the order for L_i is fixed and the order for L_{i+1} that minimizes the number of crossings amongst the edges between layers L_i and L_{i+1} is determined. After increasing index i to the maximum layer index, we turn around and repeat this process from the back with decreasing indices. In each step, an ONE-SIDED CROSSING MINIMIZATION problem has to be solved. Unfortunately, this seemingly elementary problem is $\mathcal{NP}$-complete [149], even for sparse graphs [298].

This elementary graph drawing problem attracted several researchers from the area of fixed-parameter tractable ($\mathcal{FPT}$) algorithms. The first approaches to the more general variant of this problem have been published by Dujmović *et al.* in [142] and [143]. The last one has been greatly improved by Dujmović and Whitesides [145, 146] who achieved an $\mathcal{O}(1.6182^k n^2)$ algorithm for this problem. There has been a similar race to get better approximation algorithms. The to our knowledge best one has been reported at *Graph Drawing 2003*, see [144, 301].

A bipartite graph $G = (V_1, V_2, E)$ together with linear orderings on V_1 and on V_2 is also called a *drawing* of G. This formulation implicitly assumes a drawing of G where the vertices of V_1 and V_2 are drawn (w.l.o.g., with unit-distance) on two (virtual) horizontal lines, the line L_1 corresponding to V_1 being above the line L_2 corresponding to V_2. If $u \prec v$ for two vertices on L_1 or on L_2, we will also say that u is *to the left* of v. A linear order on V_2 that minimizes the number of crossings subject to the fixed linear order of V_1 is called an *optimal ordering* and the corresponding drawing of G is called an *optimal drawing*. Since the positions of isolated vertices in any drawing are irrelevant, we disregard isolated vertices of the input graph G in what follows.

If $|V_2| = 2$, then there are only two different drawings. This gives us the useful notion of a *crossing number*. Given a bipartite graph $G = (V_1, V_2, E)$ with $|V_2| > 1$, for any two distinct vertices $a, b \in V_2$, define c_{ab} to be the number of crossings in the drawing of $G[\{a, b\} \cup (N(a) \cup N(b))]$ when $a \prec b$ is assumed.

Consider the following as a running example:

Example 6.26 In Fig. 6.2, a concrete drawing of a bipartite graph is shown. Is this drawing optimal with respect to the number of crossings, assuming the ordering of the upper layer being fixed? Notice that at some points,

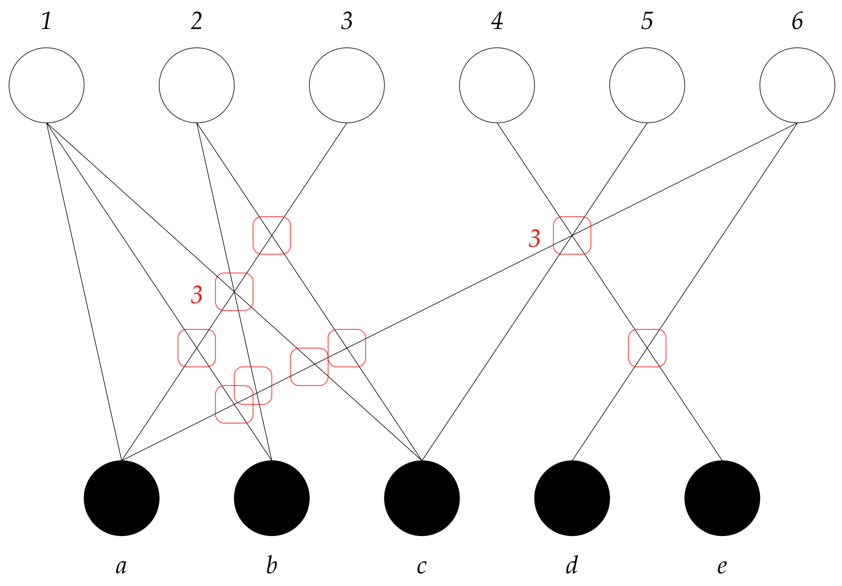


Figure 6.2: Our running example for ONE-SIDED CROSSING MINIMIZATION

more than two edges cross. How often such a crossing actually counts when computing the overall crossing number is shown at the side of a corresponding box that emphasizes the crossing point.

Let us now compute the crossing numbers c_{uv} for this graph.

c_{uv}	a	b	c	d	e
a	–	4	5	0	1
b	1	–	1	0	0
c	3	3	–	0	1
d	3	2	3	–	1
e	2	3	2	0	–

The number of crossings in the given drawing can be hence computed as

$$\begin{aligned} & c_{ab}+c_{ac}+c_{ad}+c_{ae}+c_{bc}+c_{bd}+c_{be}+c_{cd}+c_{ce}+c_{de} \\ = \; & 4+5+0+1+1+0+0+0+1+1 \\ = \; & 13. \end{aligned}$$

Furthermore, for any $u \in V_2$ with $\deg(u) > 0$, let l_u be the leftmost neighbor of u on L_1, and r_u be the rightmost neighbor of u. We call two vertices $u, v \in V_2$ *interfering* or *unsuited* if there exists some $x \in N(u)$ with $l_v \prec x \prec r_v$, or there exists some $x \in N(v)$ with $l_u \prec x \prec r_u$. Otherwise, they are called *suited*. Observe that, for $\{u, v\}$ suited, $c_{uv} \cdot c_{vu} = 0$. Dujmović and Whitesides have shown:

Lemma 6.27 *Let $G = (V_1, V_2, E)$ be an instance of* ONE-SIDED CROSSING MINIMIZATION*, where the ordering of V_1 is fixed. In any optimal ordering $\prec$ of the vertices of V_2, we find $u \prec v$ if $r_u \leq l_v$.*

This means that all suited pairs appear in their *natural ordering.*

This already allows us to formulate a first parameterized algorithm for ONE-SIDED CROSSING MINIMIZATION, which is a simple search tree algorithm. In the course of this algorithm, we will gradually construct a suitable ordering $\prec$ on V_2; when settling the ordering between u and v on V_2, we also say that we *commit* $u \prec v$ or $v \prec u$. A *generalized instance* of ONE-SIDED CROSSING MINIMIZATION therefore contains, besides the bipartite graph $G = (V_1, V_2, E)$, a partial ordering on V_2. A vertex $v \in V_2$ is *fully committed* if, for all $u \in V_2 \setminus \{u, v\}$, $\{u, v\}$ is committed.

Lemma 6.27 allows us to state the following rule:

Reduction rule 60 *For every pair of vertices $\{u, v\}$ from V_2 with $c_{uv} = 0$, commit $u \prec v$.*

Algorithm 55 A search tree algorithm solving OSCM, called OSCM-ST-simple

Input(s): a bipartite graph $G = (V_1, V_2, E)$, an integer k, a linear ordering $\prec_1$ on V_1, a partial ordering $\prec_2$ on V_2
Output(s): YES iff the given OSCM instance has a solution

```
repeat
  Exhaustively apply the reduction rules, adjusting $\prec_2$ and $k$ accordingly.
  Determine the vertices whose order is settled by transitivity and adjust
  $\prec_2$ and $k$ accordingly.
until there are no more changes to $\prec_2$ and to $k$
if $k < 0$ or $\prec_2$ contains both $(x, y)$ and $(y, x)$ then
  return NO.
else if $\exists\{x, y\} \subseteq V_2$ : neither $x \prec_2 y$ nor $y \prec_2 x$ is settled then
  if OSCM-ST-simple$(G, k-1, \prec_1, \prec_2 \cup \{(x, y)\})$ then
    return YES
  else
    return OSCM-ST-simple$(G, k-1, \prec_1, \prec_2 \cup \{(y, x)\})$
  end if
else
  return YES
end if
```

Algorithm 55 is a simple search tree algorithm for ONE-SIDED CROSSING MINIMIZATION that repeatedly uses Rule 60.

Lemma 6.28 ONE-SIDED CROSSING MINIMIZATION *can be solved in time* $\mathcal{O}^*(2^k)$.

Proof. Before any branching can take place, the graph instance is reduced, so that every pair of vertices $\{u, v\}$ from V_2 which is not committed satisfies $\min\{c_{uv}, c_{vu}\} \geq 1$. Therefore, each recursive branch reduces the parameter by at least one. ∎

The soundness of the following rule is also obvious; the rule helps clean up a graph:

Reduction rule 61 *If* (G, k, O) *is a generalized instance of* ONE-SIDED CROSSING MINIMIZATION *where* $v \in V_2$ *is fully committed, reduce to* $(G-v, k, O-v)$.

Let us visualize the work of the reductions with our example:

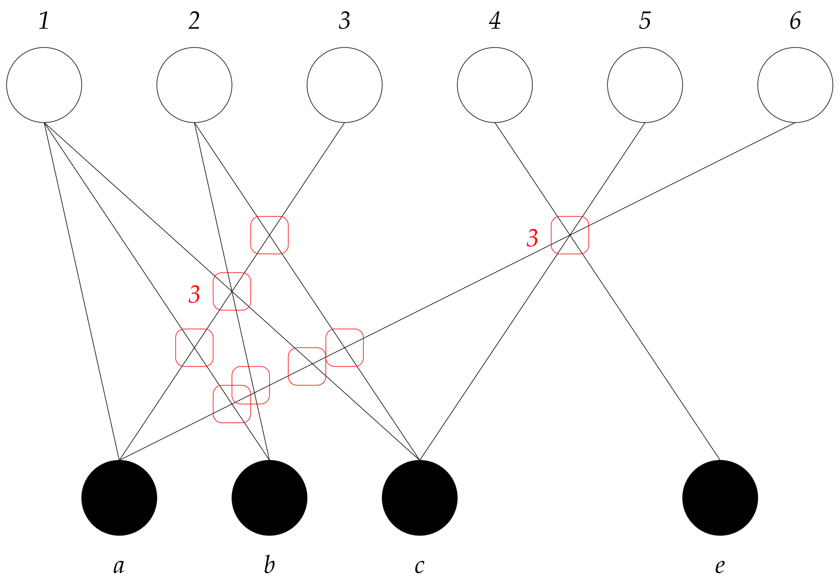


Figure 6.3: Our running example for ONE-SIDED CROSSING MINIMIZATION: how to reduce it

Example 6.29 We continue Ex. 6.26.

Rule 60 settles all relations to d, so that Rule 61 erases d; for clarity, the corresponding crossing numbers were colored blue in the table of crossing numbers given in Ex. 6.26. Moreover, $b \prec e$ will be committed. The corresponding reduced graph is shown in Fig. 6.3.

Our usual question is if and how to improve on the very simple search tree algorithm. Our first observation is that it is not necessary to branch at $\{x, y\} \subset V_2$ with $c_{xy} = c_{yx}$. This means to modifications to our Alg. 55, following the triviality last principle:

- Line 5 should exclude $c_{xy} = c_{yx}$.
- Line 12 should arbitrary commit some $\{x, y\} \subset V_2$ that are not yet committed and recurse; only if all $\{x, y\} \subset V_2$ are committed, YES is to be returned.

These modifications immediately yield an $\mathcal{O}^*(1.6182^k)$ algorithm for ONE-SIDED CROSSING MINIMIZATION. This is also the core of the algorithm proposed by Dujmović and Whitesides [146]. There, more details are discussed, as, for example:

- How to efficiently calculate all the crossing numbers c_{xy} in a preprocessing phase.
- How to integrate branch and cut elements in the algorithm that are surely helpful from a practical perspective.
- How to generalize the algorithm for instances that allow integer weights on the edges (multiple edges).

Further improvements are possible if one gives a deeper analysis of local patterns $\{x, y\} \in V_2$ such that $c_{xy}c_{yx} \leq 2$. This way, we were able to prove [144] that ONE-SIDED CROSSING MINIMIZATION can be solved in time $\mathcal{O}^*(1.4656^k)$.

Instead of actually presenting that algorithm, let us point to another remarkable fact that is already present in Alg. 55 (and that alone indicates that that algorithm is far from optimal): we never actually used the properties of OSCM as a graph drawing problem; we only relied on dealing with the crossing numbers. So, we basically presented an algorithm that solves the following problem (in the spirit inspired by Jünger and Mutzel [248]).

Problem name: POSITIVE WEIGHTED COMPLETION OF AN ORDERING (PCO)
Given: An ordered digraph $P = (V, A)$ and a cost function $\mathfrak{c}$ mapping $A(D([U(P)]^c))$ into the positive integers; by setting $\mathfrak{c}$ to zero for arcs in $A(D(U(P)))$, we can interpret the domain of $\mathfrak{c}$ as $V(P) \times V(P)$.
Parameter: a positive integer k.
Output: Is there a selection A' of arcs from $A(D([U(P)]^c))$ such that the transitive closure $(A' \cup A(P))^+$ is a linear order and

$$\sum \{\mathfrak{c}(a) \mid a \in (A' \cup A(P))^+\} \leq k \ ?$$

To understand this problem, we need to explain some more notions. We call a digraph an *ordered digraph* if its arc relation complies with the axioms of a partial order. A directed graph obtained from an undirected graph G by replacing every edge $\{u, v\}$ by the two arcs (u, v) and (v, u) is denoted by $D(G)$. The undirected graph obtained from a digraph G by putting an edge $\{u, v\}$ whenever there is an arc (u, v) in G is denoted by $U(G)$. The set of arcs of a digraph G is denoted by $A(G)$.

It is quite obvious that ONE-SIDED CROSSING MINIMIZATION can be solved with the help of POSITIVE WEIGHTED COMPLETION OF AN ORDERING; the linear order which is aimed at is the permutation of the vertices on the second layer which minimizes the number of crossings involved, so that the crossing number c_{ab} is the cost of the arc (a, b) in the digraph model. Due to the validity of reduction rule 60, all crossing numbers can be assumed to be greater than or equal to one. Since it is easy to see that POSITIVE WEIGHTED COMPLETION OF AN ORDERING is nondeterministically solvable in polynomial time, and since Eades and Wormald [149] have shown that ONE-SIDED CROSSING MINIMIZATION is $\mathcal{NP}$-hard, we can immediately deduce:

Lemma 6.30 POSITIVE WEIGHTED COMPLETION OF AN ORDERING *is* $\mathcal{NP}$-*complete.*

This makes POSITIVE WEIGHTED COMPLETION OF AN ORDERING an interesting problem to study. In addition to being used as a tool in solving k-ONE-SIDED CROSSING MINIMIZATION, this problem, as a variant of the LINEAR ORDERING PROBLEM, is of independent interest. The LINEAR ORDERING PROBLEM, defined as POSITIVE WEIGHTED COMPLETION OF AN ORDERING where $A = \{\emptyset\}$ (but without the constraint that all arc costs are necessarily positive), is a well-known $\mathcal{NP}$-hard combinatorial optimization problem. This problem has large number of applications in such diverse fields as economy, sociology, graph theory, archaeology, and task scheduling. Applications of the LINEAR ORDERING PROBLEM include the triangulation of input-output matrices in economy, minimizing total weighted completion time in one-machine scheduling and the aggregation of individual preferences in the social sciences. For a good overview on this problem, the reader is referred to the corresponding book of Reinelt [332]. In fact, the famous branch and cut methods was first formulated for and successfully applied to the LINEAR ORDERING PROBLEM by Grötschel *et al.* [218], and these resemble in spirit and methodology the search tree methods we will present below.

Keeping in mind what we said above, Alg. 55 can be read as an algorithm solving POSITIVE WEIGHTED COMPLETION OF AN ORDERING. Hence, we immediately get:

Lemma 6.31 POSITIVE WEIGHTED COMPLETION OF AN ORDERING *is in* $\mathcal{FPT}$.

However, we like to improve on the running time of our algorithm in what follows in two ways:

1. We provide a small kernel for PCO.

2. We give a better search tree algorithm for PCO.

Note that this (on the fly) also solves the corresponding problems for OSCM, albeit the search tree algorithm we shall present has worse running time that what was claimed above for OSCM.

In fact, we can even observe a *kernelization scheme* for POSITIVE WEIGHTED COMPLETION OF AN ORDERING:

Reduction rule 62 *[RRLOq] For any* fixed $q > 1$ *do:*
For each connected component $C \subseteq V$ *of* $[U(P)]^c$ *with* $|C| \leq q$*, solve* PCO *optimally on* $P[C]$*.*

The reduced instance will see the orderings between all pairs of vertices from C settled, and the parameter will be reduced accordingly. Note that this implies that all the vertices of C are isolated in the reduced $[U(P)]^c$ and can thus be deleted by Rule 61 that is also valid for POSITIVE WEIGHTED COMPLETION OF AN ORDERING.

This new set of rules yields a kernel of size $k\frac{q+1}{q}$, since after exhaustive application of this rule and Rule 61, each connected component of $[U(P)]^c$ has at least $(q+1)$ vertices and thus at least q edges. Correspondingly, at least q arcs with a weight of at least one per arc have to be added per component, therefore there are at most k/q components. In other words, a kernel size of

k (where the kernel size is measured in terms of the number of vertices of the corresponding order graph) can be "approximated" with arbitrary precision, at the expense of higher running times (the connected components of size up to q have to be solved), the typical behavior of a kernelization scheme. Still, arbitrary constants q or $q = \log k$ are possible choices for polynomial-time kernelization algorithms.

Lemma 6.32 *The kernelization scheme is sound.*

Proof. Assume that C is a connected component of $[U(P)]^c$. Consider a vertex $x \notin C$. Since C is a connected component in $[U(P)]^c$, x is comparable with each $y \in C$ (otherwise, y and x would be connected in the complement graph). Hence, C can be portioned into $C_\ell = \{y \in C \mid y \prec x\}$ and $C_r = \{y \in C \mid x \prec y\}$. Since $C_\ell \prec x \prec C_r$, either $C_\ell = \emptyset$ or $C_r = \emptyset$; otherwise, there would be no connection between vertices of C_ℓ and vertices of C_r in the complement graph. Hence, x will never be ordered "in-between" two vertices from C. ∎

Therefore, the reduction rule RRLOq is sound and we can conclude:

Theorem 6.33 *Fix some $1 < \alpha \leq 1.5$. Then, each instance of* POSITIVE WEIGHTED COMPLETION OF AN ORDERING *admits a problem kernel of size αk.*

Corollary 6.34 *Fix some $1 < \alpha \leq 1.5$. Then, each instance of* ONE-SIDED CROSSING MINIMIZATION *can be reduced to an instance $(P = (V_2, A), k)$ of* POSITIVE WEIGHTED COMPLETION OF AN ORDERING*, with $|V_2| \leq \alpha k$.*

This has not yet given us a problem kernel for the original problem, since $|V_1|$ (and hence $|E|$) has not yet been bounded by a function of k. With that aim, consider the following simple reduction rule, akin to Buss's rule for VERTEX COVER:

Reduction rule 63 *If $c_{ab} > k$ then do: if $c_{ba} \leq k$ then commit $b \prec a$ else return* NO.

This is clearly a sound rule. Moreover notice, that if a vertex $a \in V_2$ has $\deg(a) \geq 2k + 1$, then for every vertex $b \in V_2$, $c_{ab} > k$ or $c_{ba} > k$ are true. Therefore, after exhaustively performing Rule 63 in combination with Rule 61, all the vertices of V_2 of degree larger than $2k$ will have been removed from the ONE-SIDED CROSSING MINIMIZATION instance. This yields:

Theorem 6.35 *For some $1 < \alpha \leq 1.5$,* OSCM *admits a problem kernel $G = (V_1, V_2, E)$ with $|V_1| \leq 2k|V_2| (\leq 3k^2)$, $|V_2| \leq \alpha k$, and $|E| \leq 2k|V_2|$.*

Now, we come to improving the search tree algorithm.

Theorem 6.36 POSITIVE WEIGHTED COMPLETION OF AN ORDERING *can be solved in time $\mathcal{O}(1.5175^k + kn^2)$, where n is the number of vertices in the accompanying ordered digraph.*

To describe the algorithm for POSITIVE WEIGHTED COMPLETION OF AN ORDERING we will need the following definitions.

A pair of vertices $a, b \in V$ is *transitive* in an ordered digraph $P = (V, A)$ if there exists a vertex $c \in V$ such that $\{a, b\}$ is transitive with respect to c. We say that a pair of vertices $\{a, b\}$ is *strongly dependent* with respect to a vertex c if $\{a, b\}$ is dependent but not transitive with respect to c.

$a, b \in V$ form a i/j pattern in a given PCO instance if $\mathfrak{c}((a, b)) = i$ and $\mathfrak{c}((b, a)) = j$. Furthermore, if $\{a, c\}$ or $\{b, c\}$ form 2/1 pattern, we say that $\{a, b\}$ is *strongly 2/1 dependent* with respect to vertex c. We are now ready to give the algorithm for PCO.

Let us mention once more that it is possible to interleave other (more complicated) instances of the family of reduction rules RRLOq with the search tree algorithm to further compress the instances as far as affordable. This presents a general way of how to actually implement kernelization schemes (the overall algorithm is exponential anyway).

Proof. It is clear that the algorithm is correct if the step (*) is valid. Before showing that the step is indeed valid, we prove the claimed running time by analyzing its exponential part, that is we analyze the size of the search tree.

A node of the search tree branches with three conditions:

1. There is a dependent i/j pattern $\{a, b\}$ such that $i + j \geq 4$.

2. There is a 2/1 pattern $\{a, b\}$ that is transitive.

3. There is 2/1 pattern $\{a, b\}$ strongly 2/1 dependent w.r.t. c.

The smallest branching vectors that may arise in the first case are $(3, 1)$ and $(2, 2)$. These are also the smallest branching vectors arising in the second case, since $\{a, b\}$ is a transitive 2/1 pattern; thus, committing $a \prec b$ or $b \prec a$ commits by transitivity at least one other pair.

Consider now the branching vectors arising in the last case. Since the pair $\{a, b\}$ is strongly 2/1 dependent with respect to vertex c, we know that at least one of the four costs $\mathfrak{c}(a, c)$, $\mathfrak{c}(c, a)$, $\mathfrak{c}(b, c)$, and $\mathfrak{c}(c, b)$ is equal to 2 units, the worst case being then that all the other costs are one. This situation is then best analyzed by considering all possible arc directions when $\mathfrak{c}((a, b)) = \alpha \in \{1, 2\}$, $\mathfrak{c}((b, a)) = 3-\alpha$, $\mathfrak{c}((b, c)) = \beta \in \{1, 2\}$, $\mathfrak{c}((c, b)) = 3-\beta$, $\mathfrak{c}((a, c)) = \mathfrak{c}((c, a)) = 1$. The different arc directions according to which we branch are then:

Algorithm 56 A parameterized algorithm solving POSITIVE WEIGHTED COMPLETION OF AN ORDERING, called PCO

Input(s): an ordered digraph graph $G = (V, A)$, an integer k, a cost function $\mathfrak{c}$

Output(s): YES iff the given PCO instance has a solution

```
repeat
   Exhaustively apply the reduction rules, adjusting $G$, $\mathfrak{c}$, and $k$ accordingly.
   Determine the vertices whose order is settled by transitivity and adjust accordingly.
until there are no more changes
if $k < 0$ then
   return NO.
else if (there is a dependent $i/j$ pattern $\{a, b\}$ such that $i + j \geq 4$) or (there is a 2/1 pattern $\{a, b\}$ that is transitive) then
   Set $\mathfrak{c}((a, b)) = \mathfrak{c}((b, a)) = 0$.
   if PCO($G \cup \{(a, b)\}, k - 1, \mathfrak{c}$) then
      return YES
   else
      return PCO($G \cup \{(b, a)\}, k - 1, \mathfrak{c}$)
   end if
else if there is 2/1 pattern $\{a, b\}$ strongly 2/1 dependent w.r.t. $c$ then
   Set $\mathfrak{c}((a, b)) = \mathfrak{c}((a, b)) = \mathfrak{c}((b, c)) = \mathfrak{c}((c, b)) = 0$.
   for all orderings $O$ between $a, b$ and $b, c$ do
      set $k'$ as updated $k$
      if PCO($G \cup O, k', \mathfrak{c}$) then
         return YES
      end if
   end for
else
   resolve each remaining 2/1 pattern in favor of the cheaper ordering and update $G$ and $k$ accordingly; (*)
   resolve all remaining 1/1 patterns arbitrarily and update $G$ and $k$ accordingly;
   return $k \geq 0$.
end if
```

1. (a, b) and (b, c): by transitivity, this commits (a, c), so that this selection costs $\alpha + \beta + 1$;

2. (a, b) and (c, b), costing $\alpha + (3 - \beta)$,
3. (b, a) and (b, c), costing $(3 - \alpha) + \beta$; and
4. (b, a) and (c, b), where transitivity establishes a cost of $(3 - \alpha) + (3 - \beta) + 1$.

Going through all choices of α and β leaves us with the branching vectors $(2, 4, 4, 4)$ (for $\alpha \neq \beta$) and $(3, 3, 3, 5)$ (for $\alpha = \beta$). Thus overall in the above algorithm the worst case branching is $(2, 4, 4, 4)$ and $(3, 3, 3, 5)$ resulting in the search tree having the size 1.5175^k in the worst case.

We now prove that step (*) of the algorithm is valid. Let a time counter be equal to zero right before step (*) starts. First observe that, at that time there are no two incomparable pairs $\{a, b\}$ and $\{a, c\}$ both forming a 2/1 pattern, as otherwise $\{a, b\}$ would have been committed in the preceding cases. Now consider committing, one by one, each of the remaining 2/1 patterns by their cheaper ordering. In that process, whenever a pair of vertices is committed (either directly or by transitivity) we increase the time by one and assign the current time to the pair being committed. We will now prove that no pair is committed by transitivity. We first show that no 1/1 pattern gets committed (by transitivity) at any time step. Assume the contrary and let i denote the time step when the first 1/1 pattern $\{c, d\}$ gets committed. Since we only commit directly 2/1 patterns, the pair $\{a, b\}$ committed at time $i-1$ forms a 2/1 pattern. Therefore, without loss of generality, committing $a \prec b$ at time $i - 1$ commits by transitivity $c \prec d$ at time i. That further implies that at time $i - 1$, we have $c \prec a$ and $b \prec d$ (note that we may also have $c = a$ and $b \prec d$ or $c \prec a$ and $b = d$). In either case, at least one of these two pairs, $\{c, a\}$ or $\{b, d\}$, say $\{a, c\}$, is incomparable at step 0 as otherwise $\{a, b\}$ would have been committed at a previous step of the algorithm. Thus, $\{a, c\}$ forms either a 2/1 or a 1/1 pattern. Since $\{a, b\}$ forms a 2/1 pattern then by the above observation $\{a, c\}$ cannot form a 2/1 pattern. Furthermore, $\{a, c\}$ cannot stem from a 1/1 pattern since the pair is incomparable at time 0 and is comparable at time $i - 1 > 0$ thus contradicting the fact that $\{c, d\}$ is the 1/1 pattern that is assigned the smallest time stamp. Hence, no 1/1 pattern gets committed (by transitivity) at any time step.

Assume now that some 2/1 pattern $\{g, h\}$ gets committed by transitivity at some time j. Since no 1/1 pattern gets committed at any time step, the pair $\{e, f\}$ committed at time $j - 1$ forms a 2/1 pattern. Therefore, without loss of generality, committing $e \prec f$ at time $j - 1$ commits by transitivity $g \prec h$ at time j. Since $\{e, f\}$ and $\{g, h\}$ form 2/1 patterns, $e \neq g \neq h$ and $f \neq g \neq h$. Furthermore, at time $j - 1$ we know that $g \prec e$ and $f \prec h$ and at least one of these two pairs, say $\{e, g\}$, is incomparable at time 0, as

otherwise $\{g, h\}$ would have been already committed earlier in the course of the algorithm. Since the pair $\{e, g\}$ is incomparable at time 0 and the pair $\{g, h\}$ forms a 2/1 pattern, by the above observation the pair $\{e, g\}$ must form 1/1 pattern. However, by the above demonstration, it is not possible that a 1/1 pattern incomparable at time 0 gets committed at any later time.
∎

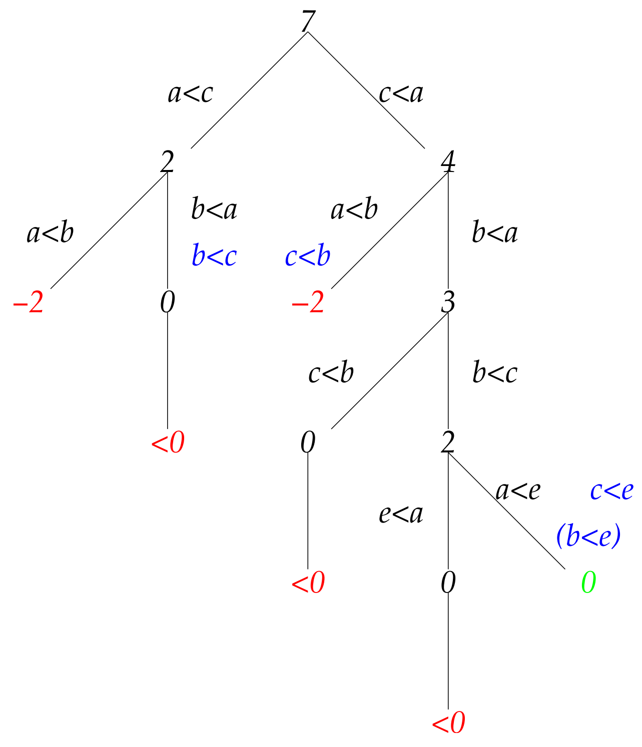


Figure 6.4: A search tree example for ONE-SIDED CROSSING MINIMIZATION.

We can actually apply this algorithm to our Example (in fact, the more complicated cases would not trigger); this gives the search tree of Fig. 6.4. This graphic is to be read as follows:

- Nodes contain the parameter value.
- Edges are labeled with the committed orders; blue labels indicate the orders committed by transitivity.

We start with displaying the reduced generalized graph instance (and the corresponding table), assuming we started with the instance $(G, 7)$. The (minimum) solution found this way is shown in Fig. 6.5.

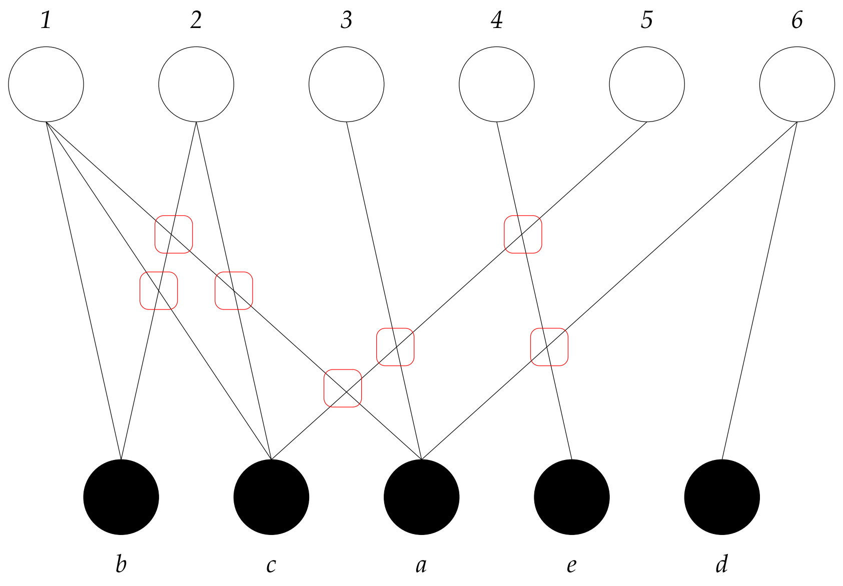


Figure 6.5: An optimal solution to our instance.

As a possible line of future research, we mention a variant of ONE-SIDED CROSSING MINIMIZATION (crossing minimization of two-layered graphs with vertex pairs) that was discussed in [381, 382] with a bio-informatics motivation (visualizing genetic networks to improve the understanding of interactions between genes). It might be that some of our methods apply to that problem, as well, which is interesting since good approximation algorithms are known for ONE-SIDED CROSSING MINIMIZATION (see [380]) but not for the two-layered crossing minimization problem with vertex pairs.

6.4.3 Biplanarization problems

A graph modification variant of ONE-SIDED CROSSING MINIMIZATION was also considered, ONE-LAYER PLANARIZATION, see [300]. To define this problem properly, we need another notion: A bipartite graph $G = (V_1, V_2; E)$ is *biplanar* if the vertices can be placed on two parallel lines L_1 and L_2 in the plane (where the vertices from V_i are placed on L_i) such that there are no edge crossings when edges are drawn as straight-line segments. A graph $G = (V, E)$ is *biplanar* if there is a bipartition $V = V_1 \cup V_2$ of its edge set (i.e., $G[V_1] = G[V_2] = \emptyset$) such that $(V_1, V_2; E)$ is biplanar.

The *biplanarization problem* was discussed in two variants in [142, 148, 176, 177]:

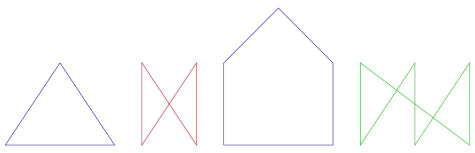
Problem name: TWO-LAYER PLANARIZATION (TLP)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a set $C \subseteq E$, $|C| \leq k$, whose removal makes the graph biplanar?

Problem name: ONE-LAYER PLANARIZATION (OLP)
Given: A bipartite graph $G = (V_1, V_2, E)$, a linear ordering $<$ on V_1
Parameter: a positive integer k
Output: Is there a set $C \subseteq E$, $|C| \leq k$, whose removal allows a biplanar drawing of the graph that respects $<$ on V_1?

In fact, the latter problem is intimately related to ONE-SIDED CROSSING MINIMIZATION: the only difference is the way in which a non-biplanar drawing is penalized; while within OSCM, the number of crossings is counted, within OLP, the number of edges that has to be removed to make the graph drawable without crossings is measured. Hence, a procedure for solving OLP can be also seen as a sub-routine within the Sugiyama approach to layered drawings. Again, the tree case is solvable in polynomial time, see [350].

Let us first study TLP. Dujmović *et al.* mentioned a nice characterization of biplanar graphs in terms of forbidden subgraphs:

Theorem 6.37 *A graph G is biplanar iff it is acyclic and does not contain a 2-claw.*


(a) The smallest four forbidden cycles.

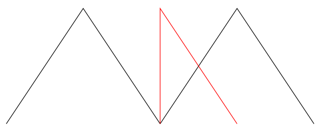

(b) The 2-claw.

Figure 6.6: Two classes of forbidden substructures for biplanar graphs.

Fig. 6.6 contain (some of) these forbidden structures and explains the notion of a *2-claw*. More precisely, there are two slightly different reasons why cycles are forbidden substructures: cycles of odd length (colored red) are forbidden, since biplanarity obviously implies bipartiteness by definition. Cycles of even length cannot be drawn without crossings, in whatever way we might arrange the vertices on the two layers. This is indicated in Fig. 6.6(a):

a drawing is shown with (implicit) layers for C_4 and C_6, indicating that erasing one edge is sufficient to make this structure drawable; observe that due to symmetry, it does not matter which edge we take out. Finally, Fig. 6.6(b) explains the notion of a *2-claw*. More formally, a *2-claw* is a graph consisting of one degree-3 vertex, the *center*, which is adjacent to three degree-2 vertices, each of which is adjacent to the center and one leaf. The 2-claw is composed of three *fingers*, one of them colored red in Fig. 6.6(b) to show the impossibility to biplanarily draw this graphs: when the finger-tip edge of the red finger is removed, however, this graph becomes biplanar.

The basic algorithmic idea would be to regard the edge sets of forbidden substructures as candidate sets and branch on the different possibilities to destroy these structures. The problem is that there are infinitely many forbidden structures, and hence there is no bound on the size of the hyperedges in any HITTING SET instance one tends to form (where the forbidden substructures correspond, as usual, to the candidate sets).

To get a search tree algorithm that runs in $\mathcal{FPT}$ time, the following two (rather easy to check) results from [143] are crucial; to understand them, we need some more notions first.

The *non-leaf degree* of a vertex v in graph G is the number of non-leaf edges at v in G, and is denoted by $\deg'_G(v)$.

A graph is a *caterpillar* (*wreath*, respectively) if deleting all the leaves produces a (possibly empty) path (cycle, respectively). These graphs are visualized in Fig. 6.7. Graphs composed of caterpillar and wreath components are exactly the graphs whose non-leaf degree is bounded by two.

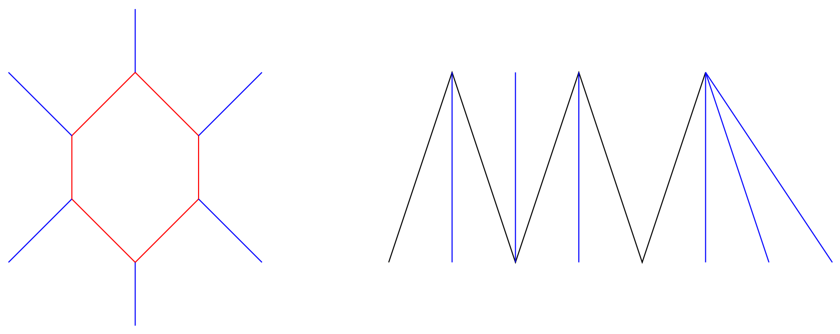


Figure 6.7: Graphs without a vertex of non-leaf degree three or larger.

Lemma 6.38 *If there exists a vertex v in a graph G such that $deg'_G(v) \geq 3$, then G has a 2-claw or a 3- or 4-cycle containing v as a subgraph.*

Lemma 6.39 *For graphs G with $deg'_G(v) \leq 2$ for all vertices v, G consists of a forest of wreaths and caterpillars. Hence, a minimum biplanarizing set of G consists of one cycle edge from each component wreath.*

These two lemmas lead themselves to Alg. 57. There, we use the notation f_C to refer to the edge set of the forbidden structure C, since we are dealing with an edge-deleting problem.

Algorithm 57 A search tree algorithm solving TWO-LAYER PLANARIZATION, called TLP-st-simple

Input(s): a graph $G = (V, E)$, a positive integer k
Output(s): YES iff there is a biplanarization set $B \subseteq E$, $|B| \leq k$

```
if $\exists v \in V : \deg'_G(v) \geq 3$ and $k \geq 0$ then
  Determine 2-claw, 3-cycle or 4-cycle $C$ containing $v$.
  for all edges $e \in f_C$ do
    if TLP-st-simple($G - e, k - 1$) then
      return YES
    end if
  end for
  return NO
else
  return ($k \geq \#$ wreath components)
end if
```

Example 6.40 Let us return to the example graph from Fig. 6.2, this time seen as an TLP instance. There are two forbidden structures to be found in this graph. In Fig. 6.8, we used color coding to emphasize them: three out of the four edges of a C_4 we colored red and the fourth edge green. There is also a 2-claw with center at vertex labeled 1, the edges of which that don't (also) participate in the 4-cycle we colored blue. The green edge is one of the edges that participate in both forbidden structures. Hence, if we remove that edge, we turn the graph into a biplanar graph. Fig. 6.9 shows how the vertices could be reordered so that we obtain a biplanar drawing of the modified graph. In that drawing, we kept the color-coding of the edges, as well as the green edge (that is of course destroying biplanarity of the drawing) to clarify the reordering.

How can we improve on this obvious $\mathcal{O}^*(6^k)$ algorithm? A direct translation into 6-HITTING SET is not possible due to the "postprocessing" of the search tree in the line colored red. However, how would such a translation look like?

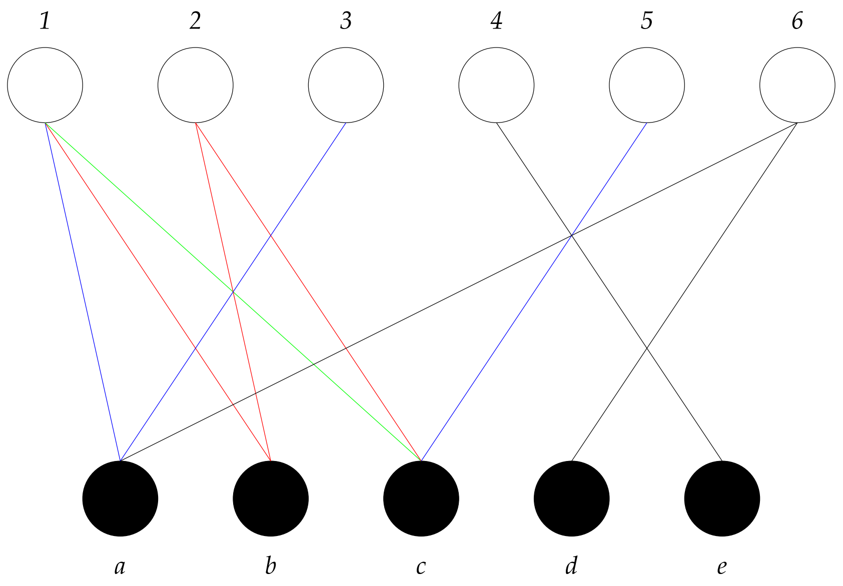


Figure 6.8: Forbidden subgraphs of our sample graph.

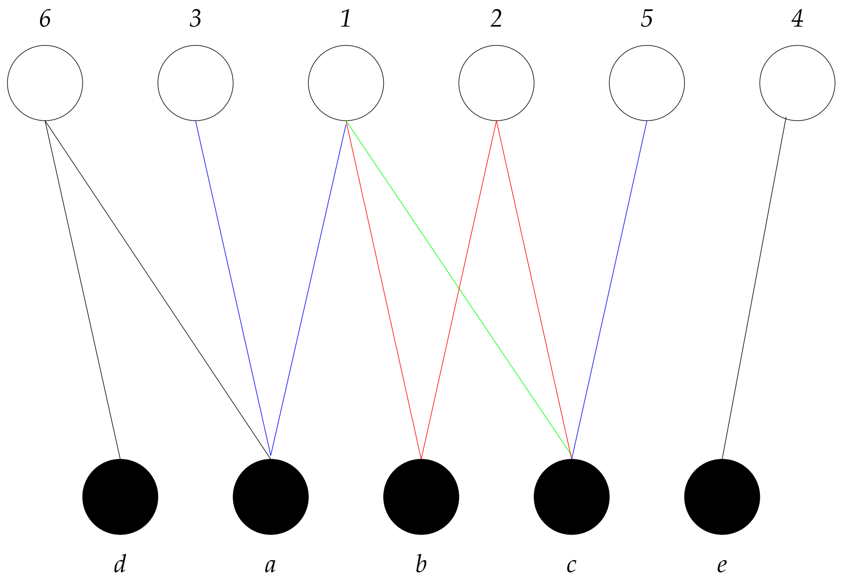


Figure 6.9: Reordering: if the green edge is deleted, the graph drawing becomes biplanar.

- Edges in the original instance correspond to the vertices of the related hypergraph.

- Hyperedges in the hypergraph correspond to the edge sets of the forbidden structures as derived according to Lemma 6.38.

This allows to translate back the reduction rules we used for obtaining better algorithms for HITTING SET. Due to the mentioned postprocessing, we cannot actually delete edges along the search tree. Instead, we will mark them *virtual*;[5] the non-virtual edges of a forbidden structure f are denoted $c(f)$, and the *size* of a forbidden structure is measured by $s(f) = |c(f)|$. Even more, also the edge domination rule that allows to delete hyperedges has to be considered with care.

When reconsidering the correctness proofs of the rules, one observes that the only rule that causes trouble is the vertex domination rule. However, without this rule, we cannot actually gain anything against the simple $\mathcal{O}^*(6^k)$ algorithm.

Nonetheless, we obtain the following reduction rules:

1. <u>structure domination</u>: A forbidden structure f is *dominated* by another structure f' if $c(f') \subset c(f)$. Then, mark f as dominated.

2. <u>small structures</u>: If $s(f) = 1$, put the only non-virtual edge into the solution that is constructed.

3a <u>isolates</u>: If e is an edge of degree zero, then mark e virtual.

Due to the relation with HITTING SET, let us call the number of non-dominated forbidden structures to which a specific edge e belongs is the *degree* of e.

Can we also handle non-virtual edges of degree one (to a certain extent) by reduction rules? We will discuss this point later on.

Let $C = \{c, w_1, w_2, w_3, x_1, x_2, x_3\}$ be a 2-claw centered at c, such that w_i is neighbored (at least) to c and x_i for $i = 1, 2, 3$. We will call $F_i = \{cw_i, w_ix_i\}$ also a *finger* of C, so that the forbidden structure $f_C = F_1 \cup F_2 \cup F_3$ corresponding to C is partitioned into three disjoint fingers. A 2-claw where one or more edges are virtual is called *injured*. Clearly, in an injured 2-claw with five edges, only one of the fingers actually got injured and two fingers are still *pretty finger*s. In an injured 2-claw with four edges, we still have at least one pretty finger left over.

The second ingredient in the approach to hitting set problems described in Chap. 5 are so-called *heuristic priorities*. More specifically, we use the

[5]This idea is similar to the marking procedure we used for our FACE COVER search tree algorithm; note that that problem also relates to HITTING SET, as it is basically RED-BLUE DOMINATING SET on planar graphs.

following rules to select forbidden structures and edges to branch at in case of multiple possibilities:

1. Select a forbidden structure f of smallest size that if possible corresponds to a short cycle.
2. If $s(f) \leq 5$ and if there is another forbidden structure f' with $c(f) \cap c(f') \neq \emptyset$ and $s(f') \leq 5$, modify $f := c(f) \cap c(f')$.
3. Select an edge e of maximal degree within f, if possible incident to the center of the 2-claw f, such that e belongs to a pretty finger.

In the following analysis, assume that we have already branched on all cycles up to length five (see the first heuristic priority). Then, we can apply the following reduction rule for (injured) 2-claws:

3b <u>(injured) 2-claws:</u> If e is an edge of degree one in a non-dominated forbidden structure of size four, five or six corresponding to an (injured) 2-claw, and if e is incident to the center of the corresponding 2-claw, then mark e virtual.

This allows us to state the whole procedure in Alg. 58, where *branch at e* means the following:

```
**if** TLP-st($G - e, k - 1, M, D_F$) **then**
  return YES
**else if** $G[M \cup \{e\}]$ is acyclic **then**
  {Avoid deleting the last edge in a cycle}
  return TLP-st($G, k, M \cup \{e\}, D_F$)
**else**
  return NO.
**end if**
```

In [176], we have shown the following result:

Theorem 6.41 *Given a graph G and an integer parameter k, Alg. 58 when called with TLP-st($G, k, \emptyset, \emptyset$), returns* YES *iff there is a biplanarization set $B \subseteq E$, $|B| \leq k$.*

The running time can be estimated as shown in Chap. 5 for the case of HS; we only sketch the basic steps in what follows. Details can be found in [176] (also cf. [177]).

Let (again) $T^{\ell}(k)$ denote the size of a search tree assuming that at least ℓ forbidden structures in the given instance (with parameter k) have size five. Then, we can prove the recurrences listed in Fig. 6.10. Let us briefly comment on these recurrences:

Algorithm 58 A search tree algorithm for TLP, called TLP-st

Input(s): a graph $G = (V, E)$, a positive integer k, a set of virtual edges M, a list of dominated forbidden structures D_F

Output(s): YES if there is a biplanarization set $B \subseteq E$, $|B| \leq k$ (and it will implicitly produce such a small biplanarization set then) or NO if no such set exists.

```
Exhaustively apply the reduction rules 1., 2., and 3a.; the resulting instance
is also called $(G, k, M, D_F)$.
**if** $\exists\, v \in V$ such that $\deg'_{G[E\setminus M]}(v) \geq 3$ **then**
  **if** possible **then**
    Find a non-dominated cycle $C$ of length at most 5
    Select an edge $e$ from $C$ and branch at $e$
  **else**
    Exhaustively apply all reduction rules
    Select 2-claw $C$ and edge $e$ from $C$ according to heuristic priorities
    Branch at $e$
  **end if**
**else**
  return ($k \geq$ # component wreaths of $G[E \setminus M]$)
**end if**
```

- The recurrence for $T^0(k)$ is due to simple binary branching. In the case that an edge e is not taken into the biplanarization set, one has to observe that two small forbidden structures are created, i.e., structures of size at most five: due to reduction rule 3b., we will branch at some e whose degree is at least two.
- The recurrence for $T^1(k)$ is the most difficult to prove; its correctness again relies on reduction rule 3b.
- The seemingly complicated recurrences for $T^2(k)$ are in fact trivial; they correspond to all possibilities for i such that two small forbidden structures f_1 and f_2 satisfy $i = |c(f_1) \cap c(f_2)|$.

With the methods that were earlier explained, we can show:

Theorem 6.42 *Given a graph G and an integer k, Alg. 58 determines if G can be made biplanar by deleting $\leq k$ edges in $\mathcal{O}(k^2 \cdot 5.1926^k + |G|)$ time, when applied to the problem kernel derived in [143].*

Let us now turn towards the case where the order of the vertices on one layer is fixed, i.e., towards ONE-LAYER PLANARIZATION. In contrast to what

$$\begin{array}{rcl} T^0(k) & \leq & T^0(k-1) + T^2(k) \\ T^1(k) & \leq & 2T^0(k-1) + 2T^1(k-1) + T^2(k-1) \\ T^2(k) & \leq & \max \left\{ \begin{array}{l} 2T^1(k-1) + 3T^2(k-1), \\ T^0(k-1) + 16T^0(k-2), \\ 2T^0(k-1) + 9T^0(k-2), \\ 3T^0(k-1) + 4T^0(k-2), \\ 4T^0(k-1) + T^0(k-2) \end{array} \right\} \end{array}$$

Figure 6.10: Upperbounding the size of the search tree of Alg. 58.

we presented for TLP, we will focus here mainly on the development of a kernelization algorithm. Again, the similarities to HITTING SET problems is helpful but distinctive features have to be observed.

The next two results from [143] give important properties for π-biplanar graphs.

Lemma 6.43 *A bipartite graph $G = (A, B; E)$ with a fixed permutation π of A is π-biplanar if and only if G is acyclic and the following condition holds.*

For every path (x, v, y) of G with $x, y \in A$, and for every vertex $u \in A$ between x and y in π, the only edge incident to u (if any) is uv. $(\star)$

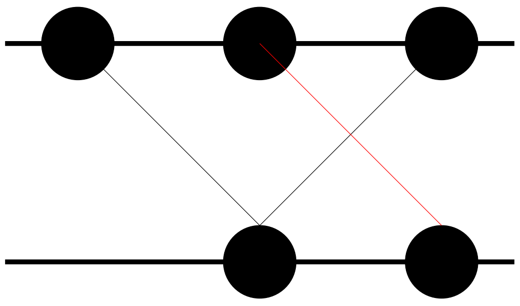


Figure 6.11: A new forbidden substructure for OLP

This means that the 5-vertex graph depicted in Fig. 6.11 (where the upper layer, i.e., the layer that contains three vertices, is fixed with respect of the ordering of the vertices) is a forbidden structure. It is not hard to verify that all cycles but cycles of length four and also 2-claws contain this forbidden substructure, in whatever way we assign the vertices to the layer. This immediately shows that ONE-LAYER PLANARIZATION is indeed "simpler" than TWO-LAYER PLANARIZATION from a parameterized algorithmics standpoint.

Let $G = (A, B; E)$ be a bipartite graph with a fixed permutation of A that satisfies condition $(\star)$. Let $H = K_{2,p}$ be a complete bipartite subgraph of G with $V(H) \cap A = \{x, y\}$, and $V(H) \cap B = \{v \in B : vx \in E, vy \in E, \deg_G(v) = 2\}$, and $|V(H) \cap B| = p$. Then H is called a *p-diamond*. Every cycle of G is in some p-diamond with $p \geq 2$.

Lemma 6.44 *If $G = (A, B; E)$ is a bipartite graph and π is a permutation of A satisfying condition $(\star)$ then the* biplanarization number*, given π, satisfies:*

$$\mathsf{bpr}(G, \pi) = \sum_{\textit{maximal } p\textit{-diamonds of } G} (p - 1) \ .$$

We are now going to derive a kernelization algorithm for OLP. Let us say that an edge e of a bipartite graph G *potentially violates condition* $(\star)$ if, using the notation of condition $(\star)$, $e = e_i$ for $i = 1, 2, 3$, where $e_1 = xv$ or $e_2 = vy$ or $e_3 = uz$ for some u strictly between x and y in π such that $z \neq v$. We will also say that e_1, e_2, e_3 (together) *violate condition* $(\star)$.

According to Lemma 6.43 (as well as the proof of Lemma 6.44 for the last two rules), the following reduction rules are sound, given an instance $(G = (A, B; E), \pi, k)$ of OLP. Analogues to the first three rules are well-known from Hitting Set problems, see Chap. 5.

Reduction rule 64 *If $e \in E$ does not participate in any cycle and does not potentially violate condition $(\star)$, then remove e from the instance (keeping the same parameter k).*

Reduction rule 65 *If $v \in A \cup B$ has degree zero, then remove v from the instance and modify π appropriately (keeping the same parameter k).*

Reduction rule 66 *If $e \in E$ participates in more than k^2 situations that potentially violate condition $(\star)$, then put e into the biplanarization set and modify the instance appropriately (also decreasing the parameter).*

Let $E_\star \subseteq E$ be all edges that potentially violate condition $(\star)$. Let $E_\circ \subseteq E$ be all edges that participate in cycles. Let G_{4c} be generated from those edges from $E_\star \setminus E_\star$ that participate in 4-cycles. By construction, G_{4c} satisfies $(\star)$. Lemma 6.44 shows that the next reduction rule can be applied in polynomial time:

Reduction rule 67 *If* $\mathsf{bpr}(G_{4c}, \pi) > k$*, then* NO.

Lemma 6.45 *Let $G = (A, B; E)$ be a bipartite graph and let π be a permutation of A. Let $v \in B$. Then, there is at most one edge e incident to v that does not potentially violate condition $(\star)$ and participates in cycles of length > 4.*

Theorem 6.46 *Let $G = (A, B; E)$ be a bipartite graph, π be a permutation of A and $k \geq 0$. Assume that none of the reduction rules applies to the* OLP *instance (G, π, k). Then, $|E| \leq k^3$. Hence, $|V| < k^3$. The kernel can be found in time $\mathcal{O}(|G|^2)$.*[6]

Proof. Now consider $E_\star$ as vertex set V' of a hypergraph $G' = (V', E')$ and put $\{e_1, e_2, e_3\}$ into E' iff e_1, e_2, e_3 together violate condition $(\star)$. A subset of edges from E whose removal converts $(A, B; E)$ into a bipartite graph which satisfies condition $(\star)$ is in obvious one-to-one correspondence with a hitting set of the hypergraph G'. Niedermeier and Rossmanith have shown [308, Proposition 1] a cubic kernel for 3-HITTING SET, so that at most k^3 edges are in $E_\star$ (else NO). Their reduction rules correspond to our rules 64 and 66.

If $e = xy \in E_\circ \setminus E_\star$ with $y \in B$ does not belong to a 4-cycle, then Lemma 6.45 shows that there is no other edge $zy \in E_\circ \setminus E_\star$. But since $xy \in E_\circ$, there must be some "continuing edge" zy on the long circle xy belongs to, so that $zy \in E_\star$ follows. We can take zy as a *witness* for xy. By Lemma 6.45, zy can witness for at most one edge from $E_\circ \setminus E_\star$ incident to y and not participating in a 4-cycle.

This allows us to partition $E_\circ$ into three disjoint subsets: (a) $E_\circ \cap E_\star$, (b) $E_{4c} = \{e \in E_\circ \setminus E_\star \mid e \text{ participates in a 4-cycle }\}$: there can be at most $4k$ such edges according to rule 67 and Lemma 6.44, and (c) $E_\circ \setminus E_{4c}$: according to our preceding reasoning, there are at most $|E_\star|$ many of these edges.

Rule 65 allows to conclude the bound on the number of vertices. ∎

Theorem 6.47 *(Dujmović* et al. *[143]) Given a bipartite graph $G = (A, B; E)$, a fixed permutation π of A, and integer k, there is an algorithm that determines if $\mathsf{bpr}(G, \pi) \leq k$ in $\mathcal{O}(3^k \cdot |G|)$ time.*

Can we further improve on this algorithm? Firstly, it is clear that we can combine the search tree algorithm with the kernelization algorithm described above. But furthermore, observe that the search tree algorithm basically branches on all members of $E_\star$, trying to destroy the corresponding triples of edges violating condition $(\star)$. This means that we again take ideas stemming from solutions of the naturally corresponding instance of 3-HITTING SET.

[6]More recently, a quadratic kernel for 3-HITTING SET was derived [310] based on the $2k$-kernel for VERTEX COVER, see Chap. 4. Translating the corresponding reduction rules shows that $|E_\star|$ and hence $|E|$ is in fact upperbounded by $\mathcal{O}(k^2)$.

Unfortunately again, we cannot simply "copy" the currently best search tree algorithm for 3-HITTING SET, see Chap. 5, since destroying triples of edges violating condition ($\star$) might incidentally also destroy more or less of the 4-cycles. As explained in the TLP case, the problem is again the vertex domination rule. In order to gain anything against the previously sketched algorithm, we must somehow at least *avoid branching on vertices of degree one contained in hyperedges of size three.*

Firstly, we can prove a lemma that shows that, whenever we have branched on all hyperedges of size three in the 3-HITTING SET instance (that correspond to situations violating condition ($\star$) in the original OLP instance) that contain vertices of degree at least two, then we have already destroyed all "large" cycles. Then, we investigate the possible interaction between a cycle of length four and a structure violating ($\star$), after having "destroyed" all "mutually interacting" structures violating ($\star$).

Lemma 6.48 *Let $G = (A, B; E)$ be a bipartite graph and π be a fixed permutation of A. Assume that if $h = \{e_1, e_2, e_3\}$ and $h' = \{e'_1, e'_2, e'_3\}$ are two situations violating ($\star$), then $h \cap h' = \emptyset$. Let $C = \{ab, bc, cd, da\}$ be a sequence of edges forming a 4-cycle. Then, there is at most one hyperedge h—among the hyperedges modeling situations violating ($\star$)—such that $C \cap h \neq \emptyset$.*

Hence, after the indicated branching, for each 4-cycle, at most one hyperedge of size three remains such that the corresponding edge sets have nonempty intersection. Since we have to destroy every 4-cycle, the best we then can obviously do is to take out an edge that takes part in the "accompanying" situation violating ($\star$). This can be done completely deterministically due to the preceding lemma. Finally, the only remaining situations correspond to possibly interacting 4-cycles. These can be solved with Lemma 6.44.

In the algorithm depicted in Alg. 59, we again use a set of *virtual edges* M to mark edges which (according to our previous branching) we shall *not* put into the biplanarization set. This part of the input is therefore initialized with $\emptyset$ at the very beginning. The notation $\mathsf{bpr}(G, \pi, M)$ is accordingly understood.

As a subroutine, we have singled out the situation exclusively dealing with the T^0-branching. Note that the branching in the T^2-case can be also viewed as "branching at e" followed by the application of the small structure rule (which is also valid in this case, since it is derived from the 3-HITTING SET reduction rules).

Theorem 6.49 OLP *can be solved in $\mathcal{O}(k^3 \cdot 2.5616^k + |G|^2)$ time.*

Proof. We only have to analyze the running time in the following. As said

Algorithm 59 A search tree algorithm for ONE-LAYER PLANARIZATION, called OLP

Input(s): a bipartite graph $G = (A, B; E)$, a permutation π of A, a positive integer k, a set of virtual edges M

Output(s): NO if $\mathsf{bpr}(G, \pi, M) > k$; otherwise, YES (and it will implicitly produce such a small biplanarization set then)

```
Exhaustively apply the reduction rules; the resulting instance is also called
$(G, \pi, k, M)$.
if $(\star)$ fails for some path $(x, v, y)$ and edge $ab$ with $x \prec a \prec y$ then
  if $k = 0$ then
    return NO
  else if possible then
    choose $(x, v, y)$ and $ab$ such that $\{xv, vy, ab\} \cap M \neq \emptyset$
    if possible then
      choose $(x', v', y')$ and $a'b'$ such that $\{x'v', v'y', a'b'\} \cap M \neq \emptyset$ and
      $\{e\} = \{xv, vy, ab\} \cap \{x'v', v'y', a'b'\} \cap (E \setminus M)$
      {This is a $T^2$-branch}
      if OLP($G - \{e\}$,$\pi$, $k - 1$, $M$)='YES' then
        return YES
      else
        Let $\{e, f\} = \{xv, vy, ab\} \cap (E \setminus M)$.
        Let $\{e, f'\} = \{x'v', v'y', a'b'\} \cap (E \setminus M)$.
        return OLP($G - \{f, f'\}$,$\pi$, $k - 2$, $M \cup \{e\}$)
      end if
    else
      {only "isolated" forbidden structures of size two (also $T^1$)}
      Let $\{e, f\} = \{xv, vy, ab\} \cap (E \setminus M)$.
      if OLP($G - \{e\}$,$\pi$, $k - 1$, $M$)='YES' then
        return YES
      else
        return OLP($G - \{f\}$,$\pi$, $k - 1$, $M$)
      end if
    end if
  else
    {$T^0$-branch}
    return OLP-T0($G, \pi, k, M$)
  end if
else
  return $k \geq \sum_{\text{maximal } p\text{-diamonds of } G} (p - 1)$
end if
```

Algorithm 60 A subroutine for OLP, called OLP-T0

Input(s): a bipartite graph $G = (A, B; E)$, a permutation π of A, a positive integer k, a set of virtual edges M; it has been tested that no reduction rules apply and that no T^ℓ-branching with $\ell > 0$ is possible

Output(s): NO if $\mathsf{bpr}(G, \pi, M) > k$; otherwise, YES (and it will implicitly produce such a small biplanarization set then)

```
if possible then
  choose $(x, v, y)$ and $ab$ such that there are $(x', v', y')$ and $a'b'$ such that
  $e \in \{xv, vy, ab\} \cap \{x'v', v'y', a'b'\}$
  if OLP($G - \{e\}$,$\pi$, $k - 1$, $M$) = 'YES' then
    return YES
  else
    return OLP($G$,$\pi$, $k$, $M \cup \{e\}$)
  end if
else
  {only "isolated conflicts"}
  if $h = \{xv, vy, ab\}$ intersects with some 4-cycle $C$ then
    Let $e$ be a common edge of $h$ and $C$.
  else
    Choose some $e \in h$.
  end if
  return OLP($G - \{e\}$,$\pi$, $k - 1$, $M$)
end if
```

before, branching only actually takes place when we "solve" the corresponding 3-HITTING SET instance. During these recursions, we always assume that, whenever we branch at forbidden structures of size three, there is some element contained in that forbidden structure which actually participates in at least two forbidden structures.

More distinctly, let $T^\ell(k)$ denote the situation of a search tree assuming that at least ℓ forbidden structures in the given instance (with parameter k) have a size of (at most) 2. Of course, $T(k) \leq T^0(k)$. We again analyze the recurrences for T^0, T^1 and T^2, and we use the notions of core and size of a forbidden structure similar to the TLP case.

For T^0, we obtain as in the case of 3-HITTING SET:

$$T^0(k) \leq T^0(k-1) + T^2(k).$$

For T^1, we cannot claim to "gain" any new forbidden structures of size two. Therefore, a trivial branching gives:

$$T^1(k) \leq 2T^0(k-1).$$

For T^2, we distinguish two sub-cases, considering two forbidden structures f_1, f_2 of size two:

1. $c(f_1) \cap c(f_2) = \emptyset$. Then, trivial branching gives:
$$T^2(k) \leq 2T^1(k-1) \leq 4T^0(k-2).$$
2. $\exists e \in c(f_1) \cap c(f_2)$. Branching at e (which our algorithm will do) then yields:
$$T^2(k) \leq T^0(k-1) + T^0(k-2).$$

The first sub-case leads to:
$$T^0(k) \leq T^0(k-1) + 4T^0(k-2) \leq 2.5616^k.$$

The second sub-case gives:
$$T^0(k) \leq T^0(k-1) + (T^0(k-1) + T^0(k-2)) \leq 2.4143^k.$$

So, the first sub-case yields the worst case. ∎

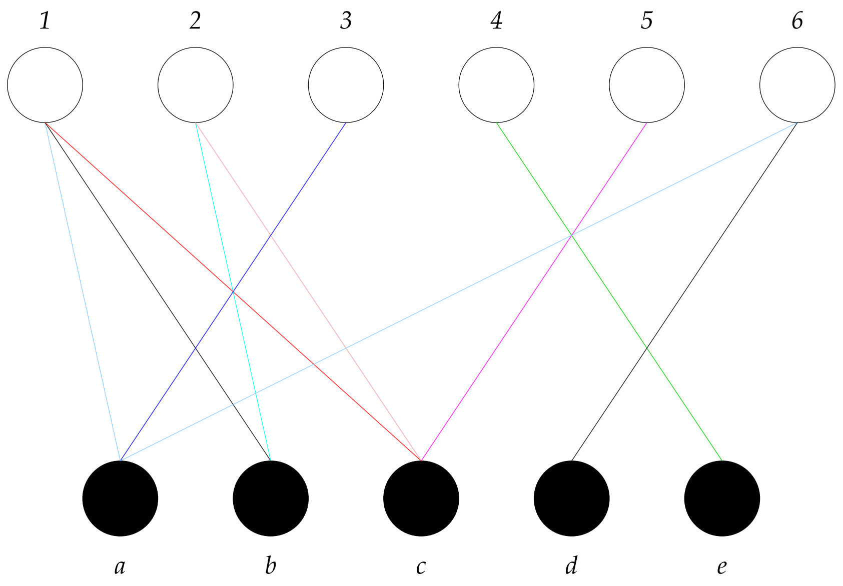
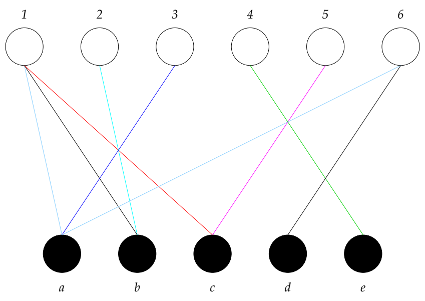
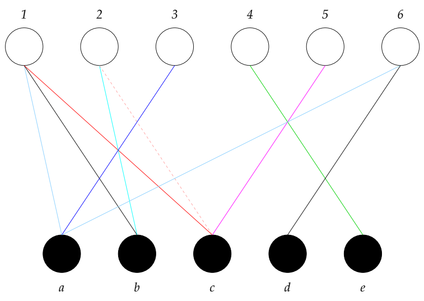


Figure 6.12: Our running example, this time for ONE-LAYER PLANARIZATION

Example 6.50 Let us have another look at Fig. 6.2 and study the behavior of our search tree algorithm.

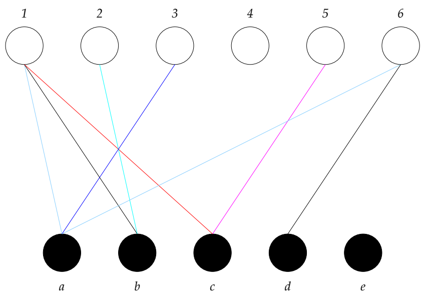


(a) The branch that takes out an edge.

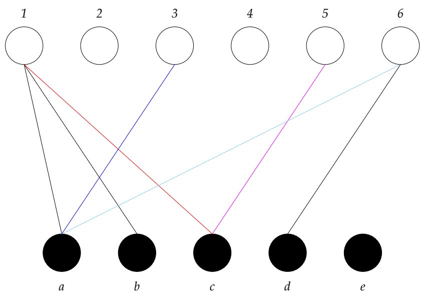


(b) The branch that makes an edge virtual.

Figure 6.13: Two possible first branches at edges of "high degree".



(a) A second-level branch.



(b) A third-level branch.

Figure 6.14: Two further branching levels, only listing the case when a specific edge is deleted.

We can find two classes of forbidden structures, marked with colors in Fig. 6.12: one of them having two edges (colored in light and dark blue) incident with vertex a and one of them having two edges (colored in red for the two possible left branches and colored in magenta for the right arm) incident with c. The remaining colored edges create forbidden structures together with the mentioned edges.

Let us pick one edge that participates in many forbidden structures, as depicted in Fig. 6.13: the edge $\{1, c\}$. Let us further follow what happens in the first branch. We might then select $\{4, e\}$ for branching; in the case we take out this edge, we arrive at Fig. 6.14(a). We follow this branch further, next considering edge $\{2, b\}$. After having deleted that (third) edge, we can color the edge $\{1, a\}$ black again, since all forbidden structures this edge did

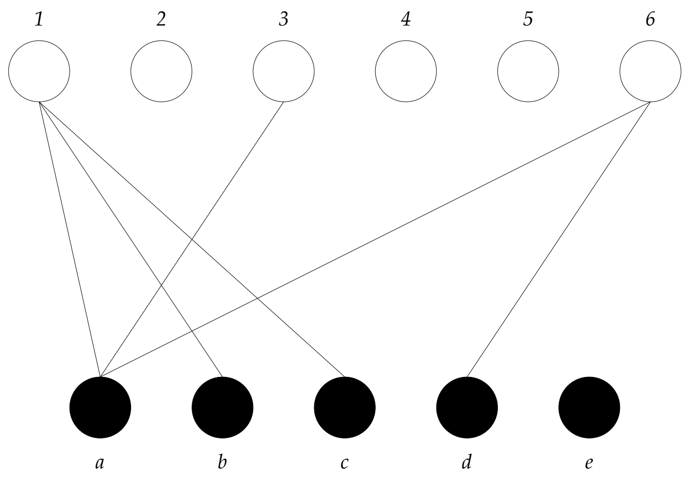


Figure 6.15: One solution to our problem.

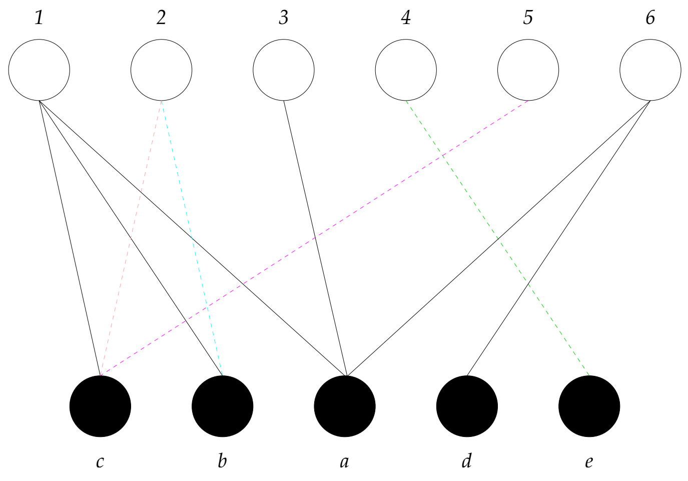


Figure 6.16: One solution to our problem, when rearranged.

participate in have been already hit; formally, this means that the edge $\{1, a\}$ is removed (by rule 64). This is all shown in Fig. 6.14.

Finally, taking out edge $\{5, c\}$ resolves our problem completely, as can be seen in Fig. 6.15: it is obviously possible to reorder the second layer (by swapping b and c to the left) so that the graph be be drawn without crossings. This can be best seen in Fig. 6.16; the edges that were taken out are drawn as dashed lines, keeping their original colors.

We should mention in this place that there is a natural generalization of TLP (and also of crossing minimization problems): namely, one could allow a "fixed" number t of layers (we treated the case $t = 2$), and ask how to optimally draw a graph in a t-planar way by omitting at most k edges (or allowing at most r crossings). This is another example of a natural problem where *two parameters* show up, see Sec. 3.4. This problem has been put into $\mathcal{FPT}$ by a set of twelve authors [142]. This result relies on the fact that graphs that can be drawn on t layers after omitting k edges and when allowing r crossings have only pathwidth bounded by $t + 2k + 2r$. [7]

To conclude our account on crossing minimization, let us mention that a nice collection of papers and problems related to crossing minimization problems of various kinds can be found in [Chapter 11, Site 16]. Overview papers are, amongst others: [349, 106, 360, 361]. Regarding specific planar drawings, not only drawings on two or more layers are of interest, but also, for example, drawings where the vertices are put on the circle; this is related to the problem of turning a graph into a one-outerplanar graph as discussed in Chap. 7; recent papers on this topic (that is up to now untouched from the viewpoint of parameterized complexity) are [38, 107].

6.4.4 More related problems

To conclude this section, let us first return to our starting problem, namely LINEAR ARRANGEMENT. The following example should show further connections and prove that there are really a heap of nice combinatorial problems in this area that have only barely touched by people working in parameterized algorithmics.

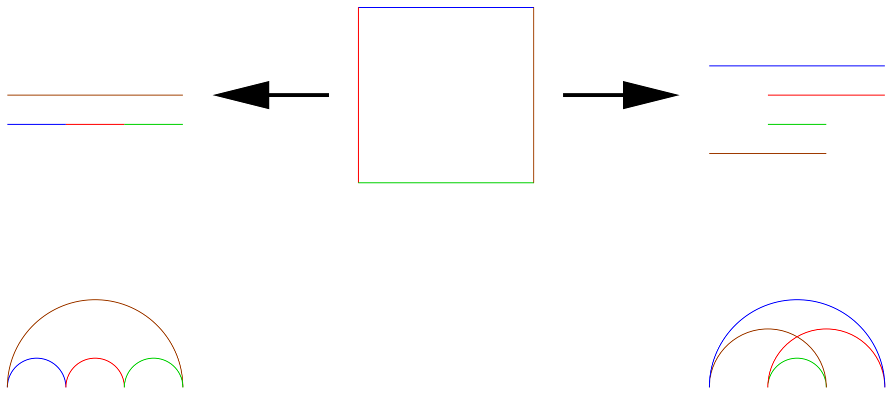


Figure 6.17: Two ways of mapping C_4: a circle interpretation

[7]Pathwidth and related notions are discussed in Chap. 7.

Example 6.51 We studied mappings of C_4 in Ex. 3.2. Here, we give a concrete "drawing interpretation" of such a mapping (different from the "band interpretation" given in Ex. 3.15): namely, if we draw each interval on which an edge is mapped as a half-circle (where all half-circles are drawn to one side of the basic "interval line"), then the mapping shown on the left-hand side incurs no crossing, while the mapping on the right-hand side produces one crossing. This is shown in Fig. 6.17.

Now, as discussed with the bilayer drawings, different criteria for optimality could be investigated: minimize the number of crossings, the number of half-circles that should be removed to ensure drawability.

Observe that the task to remove the fewest number of edges to ensure half-circle drawability is different from the problem LINEAR ARRANGEMENT BY DELETING EDGES discussed above. For example, a star graph S_n with vertices $\{1, \dots, n\}$ and edges $\{1, i\}$ for $i = 2, \dots, n$ can be drawn with half-circles without crossings, while $n - 3$ edges (out of the $n - 1$ edges of S_n) have to be removed to produce a forest of paths.

These problems are also related to the notion of *book embedding*, also see [44, 287, 305].

Note that the considerations on linear arrangements might also influence ideas to introduce parameters that could be meaningfully optimized in 2-layer-drawings. For example, one might wish to minimize the "overall length" of all straight lines involved in such a drawing (assuming that the points are drawn equidistantly on the two lines and that the two lines of the layers have a certain fixed distance; the ratio between the distance between neighbors on a layer and the distance between the two layer lines influences which drawing is preferable); again, different norms could be considered. It is possible to argue that drawings enjoying short overall length of all straight lines are better readable.

Quite recently,[8] we considered the following problem that is motivated by applications that compare two trees with each other,[9] formalizing a method that is particularly popular in bio-informatics:

Problem name: TWO-TREE CROSSING MINIMIZATION (TTCM)
Given: A two-tree (T_1, T_2) with leaf labels Λ
Parameter: a positive integer k
Output: Can (T_1, T_2) be drawn with at most k crossings ?

Here, a *two-tree* simply denotes a pair of rooted binary trees with perfect matching between corresponding leaves of the two trees, where the corre-

[8]in yet unpublished joint work with M. Kaufmann and M. Poths

[9]A nice overview on tree comparison methods can be found in [49].

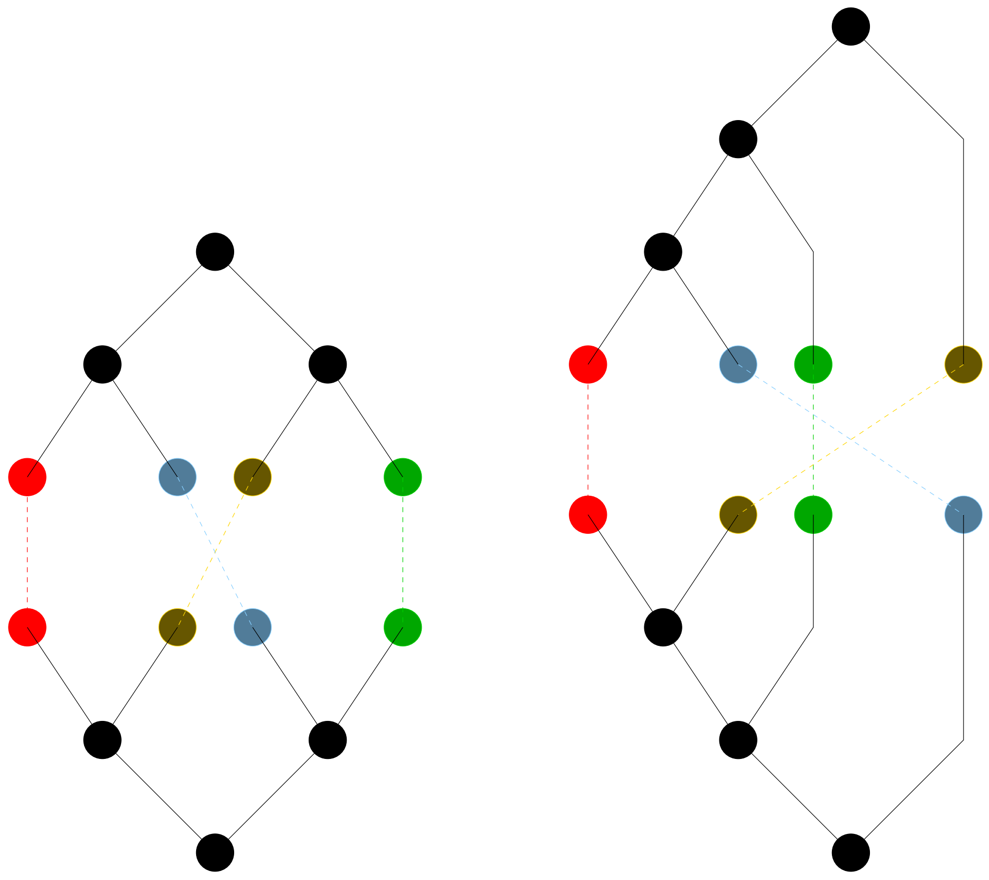

(a) Two balanced trees (b) Two unbalanced trees

Figure 6.18: The two types of contradicting quadruple two-trees.

spondence is given by an appropriate labeling such that only leaves with the same label are matched. Alternatively, we can visualize the matching by assigning labels (colors) to the leaves such that leaves that are matched get the same labels. We discuss only drawings of the following form: The given two unordered binary trees with the same number of leaves have to be embedded layerwise in the plane such that the leaves are aligned in two adjacent layers. Our goal is hence to find two planar embeddings of the two trees such that the crossings of the matching edges are minimized. Two particularly simple examples of two-trees that cannot be drawn without crossings are shown in Fig. 6.18. They are referred to as *quadruple trees* since each of the trees has four leaves.

It is hopefully clear that also TWO-TREE CROSSING MINIMIZATION is basically a restricted reordering problem: namely, the task is to order the leaves in a way that the drawing of the matching edges between corresponding vertices incurs the least number of crossings. The restriction is given by the two trees. Namely, observe that in the case of a binary tree with n leaves, there are exactly 2^{n-1} different permutations of the leaves which are implied the different ordering of the subtrees. This is in contrast to the $n!$ permutations which are possible in general.

We have achieved the following results:

Theorem 6.52 TWO-TREE CROSSING MINIMIZATION *is $\mathcal{NP}$-complete.*

Theorem 6.53 TWO-TREE CROSSING MINIMIZATION *is in $\mathcal{FPT}$. More precisely, the problem can be solved in time $\mathcal{O}^*(c^k)$ for some constant c.*

The algorithm that shows $\mathcal{FPT}$-membership is interesting insofar that the proof of its correctness basically also solves the following related problem (that can be seen as the natural graph modification variant of TTCM):

Problem name: TWO-TREE DRAWING BY DELETING EDGES (TTDE)
Given: A two-tree (T_1, T_2) with leaf labels Λ
Parameter: a positive integer k
Output: Is there a label set $L \subseteq \Lambda$ with $|L| \leq k$ such that the two-tree $(T_1 \langle \Lambda \setminus L \rangle, T_2 \langle \Lambda \setminus L \rangle)$ can be drawn without crossings ?

More specifically, we need the following structural result:

Theorem 6.54 *In each* TWO-TREE CROSSING MINIMIZATION *instance (T_1, T_2) that is not embeddable without crossings, there exists a quadruple $\{a, b, c, d\}$ of leaf labels such that $(T_1 \langle \{a, b, c, d\} \rangle, T_2 \langle \{a, b, c, d\} \rangle)$ is also not embeddable without crossings.*

Here, we use the notation $T\langle L \rangle$ to denote the "binary subtree" of T that contains only the leaves from L and as inner nodes those nodes which are least common ancestors of the L-leaves in T.

We will present a recursive algorithm which either finds such an quadruple, or it provides a crossing-free embedding (drawing) of the two-trees. This algorithm will not only prove this structural result that gives a characterization of drawable two-trees in terms of forbidden substructures, but also provides the backbone of two $\mathcal{FPT}$ algorithms that are presented in the following.

Proof. Let (T_1, T_2) be the two-trees. Let T_1 be the top one and T_2 be the one on the bottom. Without loss of generality, we assume that the set Λ of labels associated to the leaves of T_1 and T_2 has more than one element.

We assume that each inner node provides links to its two children and in addition permanent and temporary information attached to each such link. The permanent information $p(\ell)$ attached to link ℓ is either L, R or $*$. The semantics is as follows:

L This link leads to the left child.

R This link leads to the right child.

$*$ It is not yet determined if this link leads to the left or to the right child.

In the initialization phase of our algorithm, we (arbitrarily) set $p(\ell_1) = L$ and $p(\ell_2) = R$ for the two links emanating from the root of T_2. All other permanent information is set to $*$. This defines the function p (that we use both for T_1 and for T_2). Thus initiated, we call embed(T_1, T_2, p).

As can be seen from how the permanent information is initiated and updated, the following is always true:
Claim 1: Let $\mathfrak{n}$ be an inner node with emanating links ℓ_1 and ℓ_2. Then, one of the following cases is true:

- $p(\ell_1) = L$ and $p(\ell_2) = R$;
- $p(\ell_1) = R$ and $p(\ell_2) = L$; or
- $p(\ell_1) = *$ and $p(\ell_2) = *$.

The temporary information $t(\ell)$ attached to link ℓ is either L, R or M. The semantics is as follows:

L All links below (i.e., in direction to the leaves) are marked L.

R All links below are marked R.

M Mixed case: some links below are marked L and some are marked R.

The temporary information is processed in a bottom-up fashion from the leaves to the root as follows:

1. As described in the main algorithm, the links leading to the leaves of the tree to be processed are assigned either L or R.

2. Let $\mathfrak{n}$ be an inner node where to both links ℓ_1 and ℓ_2 emanating to its two children, the temporary information has been assigned. If $t(\ell_1) = t(\ell_2) = M$, then we have found a quadruple situation corresponding to the balanced tree case in Fig. 6.18. Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here.

3. Let $\mathfrak{n}$ be an inner node (besides the root) where to both links ℓ_1 and ℓ_2 emanating to its two children, the temporary information has been assigned, such that the previous situation does not apply. Then, to the link ℓ that leads to $\mathfrak{n}$ we assign $t(\ell)$ according to the following table:

$t(\ell_1)$	L	L	L	R	R	R	M	M	M
$t(\ell_2)$	L	R	M	L	R	M	L	R	M
$t(\ell)$	L	M	M	M	R	M	M	M	E

Here, E signals the error case we discussed in the previous point.

Interestingly, we can also update the permanent information of two siblings. Let $\mathfrak{n}$ be an inner node where to both links ℓ_1 and ℓ_2 emanating to its two children, the temporary information has been assigned. We update $p(\ell_1)$ and $p(\ell_2)$ according to the following table:

$t(\ell_1)$	L	L	L	L	L	L	L	L	L
$t(\ell_2)$	L	L	L	R	R	R	M	M	M
$p(\ell_1)$	L	R	$*$	L	R	$*$	L	R	$*$
$p(\ell_2)$	R	L	$*$	R	L	$*$	R	L	$*$
$p(\ell_1)$	L	R	$*$	L	E	L	L	E	L
$p(\ell_2)$	R	L	$*$	R	E	R	R	E	R

Observe that there are more cases with the roles of ℓ_1 and ℓ_2 being interchanged. Furthermore, notice that the list of cases of assignments to ℓ_1 and ℓ_2 is complete because of Claim 1. The table should be read as follows: the first four lines give the current values of t and p on the two links. The last two lines give the updated values of p. Here, an E signals that we found a contradictory situation; more specifically (as we will see below), we have found a quadruple situation corresponding to the unbalanced tree case in Fig. 6.18 (as will be explained below). Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here.
Claim 2: Observe that the graph that is induced by the edges (links) to which non-$*$ permanent information has been attached to is a tree before and after each complete bottom-up tree processing (as described above). Moreover, if this induced tree is non-empty, then it also contains the root.

How to actually use these bottom-up processing of the temporary and permanent information is explained in Alg. 61. We will make use of the following property of our algorithm:
Claim 3: Each time that it is again say the upper tree's turn to get new temporary labels, the former root of that tree (and possibly more nodes) will no longer be taken into consideration.

We still have to show that the two aborts (error cases) described above are indeed based on finding a contradictory quadruple two-tree as explained in Fig. 6.18. The proofs below of the following claims show the details of this construction.
Claim 4: Whenever an error occurs within the temporary label information update, we can exhibit a balanced quadruple two-tree.
Claim 5: Whenever a contradiction is found between the temporary label information and the already existent permanent label information, we can exhibit an unbalanced quadruple two-tree.

We still have to show that the two aborts (error cases) described above are indeed based on finding a contradictory quadruple two-tree.

Finding a pair of balanced trees. According to Alg. 61, we have specified a left and right sub-tree say in the lower tree of the two-tree. The corresponding L and R information is propagated as temporary information in the upper tree of the two-tree as described above, up to the point that we encountered two sibling links ℓ_1 and ℓ_2 to which we have attached M as temporary information. Hence, in the upper tree we find four leaf links (that we might identify with the leaf labels) a, b, c, d such that a, b (c, d, resp.) are in the sub-tree emanating from ℓ_1 (ℓ_2, resp.) with $t(a) = t(c) = L$ and $t(b) = t(d) = R$. However, in the lower tree, the leaves with labels a and b are both in the left subtree, and the leaves with labels c, d both in the right subtree. So, both in the lower and in the upper tree, the subtree that is induced by the labels a, b, c, d is a balanced quadruple tree, and the shape of both quadruple trees is contradictory as explained in Fig. 6.18.

Finding a pair of unbalanced trees. A conflict occurs here if there are two sibling links ℓ_1 and ℓ_2 emanating from an inner node such that the bottom-up algorithm infers from the temporary information an order of the links that is different from what was already stored in the permanent information.

Without loss of generality, let us assume that ℓ_1 and ℓ_2 belong to the upper tree and that $p(\ell_1) = L$ and $p(\ell_2) = R$ and that $t(\ell_1) = R$ and $t(\ell_2) = L$. Let x be the node where both ℓ_1 and ℓ_2 emanate. Assume we observed a conflict say at the t_1th level of the recursion. There must have been a point of time (i.e., level) in the recursion that for the first time fixed $p(\ell_1) = L$ and $p(\ell_2) = R$. This point could not be the initialization due to Claim 3. Since we did not observe a conflict at level $t_0 = t_1 - 2$ of the recursion, we either fixed at time t_0 for the first time the permanent information that is at t_1 leading to a conflict or it got at least confirmed when processing level t_0. This fixing (or confirming) was obtained by using the temporary link information (at time t_0). So, at time t_0, there was a leaf labeled a reachable from ℓ_1 that then was also a label of a leaf in the left subtree of the lower tree. Let us now furthermore assume (again without loss of generality by completely symmetric arguments in the other case) that the recursion branch where our observed conflict occurs is along recursing on the right subtree T^ℓ_R of the lower tree and on the left subtree T^u_L of the upper tree (where these subtrees were considered at level $t = t_0 + 1 = t_1 - 1$ of the recursion).

Consider some leaf labeled d in the right subtree T^u_R of the upper tree at level t. Naturally, d is found as a leaf label in the right subtree T^ℓ_R of the lower tree (referring to level t_0), since otherwise we would have observed non-drawability at an earlier stage. Let y be the root of T^ℓ_R. The position of d also permanently fixes the order of children links of y. Now, if the labels of the leaf descendants of the right child of y are only belonging to the label set of the leaves in T^u_R, then we won't get a contradiction at level t_1 as presumed. Hence, upon further recursing on T^u_L in step t_1, we will find both left and right descendants of y that carry labels that can be also found in T^u_L.

Let us further discuss the node x in the upper tree in what follows. There must be a descendant z of x (along ℓ_1) where one of the branches starting at z leads to a (and in fact we may assume all other leaves that can be reached from that branch carry labels that are in T^ℓ_L) and the other branch leads to some leaf with label c that can be also found as leaf label in the right tree at level t_1 of the recursion (since $t(\ell_1) = R$ at level t_1). Moreover, that descendant z is no longer "visible" at level t_1, since the "left descendants" of z do not find their counterparts in T^ℓ_R. Hence, $x \neq z$.

Moreover, by assumption on the temporary information of ℓ_2, there must be a leaf node labeled b that is reachable from ℓ_2; a leaf labeled b can be also reached in the lower tree from the left child of y.

This proves that the labels a, b, c, d as constructed above present a contradictory quadruple two-tree.

∎

This theorem motivates the study of quadruple trees.

Lemma 6.55 *There are only two types of contradicting quadruple trees, the ones depicted in Fig. 6.18.*

Since we have now identified small forbidden structures, we can translate any TWO-TREE DRAWING BY DELETING EDGES instance into 4-HITTING SET as follows: simply cycle through all $\mathcal{O}(n^4)$ possible quadruple two-trees (given a concrete two-tree (T_1, T_2) with n leaves): If a quadruple two-tree is contradictory, then it corresponds to a hyperedge with four vertices, the leaf labels forming that quadruple. All n leaf labels together are the vertices of the hypergraph.

With the help of known parameterized algorithms for 4-HITTING SET as derived in Chap. 5, we can thus show:

Corollary 6.56 TWO-TREE DRAWING BY DELETING EDGES *is in $\mathcal{FPT}$. More precisely, it can be solved in $\mathcal{O}^*(3.115^k)$ time.*

Upon solving TWO-TREE CROSSING MINIMIZATION, the additional complication arises to guarantee that each branch in the search tree (that is performed again with the small contradicting substructures), we actually "gain" one crossing. However, these difficulties can be overcome, as we sketch in the following, this way proving Theorem 6.53.

More precisely, we will sketch a parameterized algorithm that branches on small contradicting structures (primarily, at contradicting quadruples) as long as these incur new crossings. In a second phase, we attempt at drawing the remaining two-tree with using a variant of the algorithm embed, possibly finding new small contradicting structures. The validity of this approach relies on the fact that we are able to separate contradicting structures from the rest of the two-trees by attachment links that are described as follows.

Let u be a node in a tree T. The parent link $\mathfrak{p}(u)$ is the unique edge leading to u in T; if u is the root, then there is no parent link to u. Let G be a subgraph of T. The set of parent links of G, written $\mathfrak{P}(G)$, is the set of all attachment links of pairs of vertices $u \in V(G)$, i.e., $\mathfrak{P}(G) = \{\mathfrak{p}(u) \mid u \in V(G)\}$.

Let u, v be nodes in a tree T. The *attachment link* of u and v, written $\mathfrak{a}(u, v)$, is the unique edge leading to the least common ancestor (lca) of u and v, i.e., $\mathfrak{a}(u, v) = \mathfrak{p}(\text{lca}(u, v))$; if the least common ancestor of u, v is the root, then there is no attachment link to u and v.

Algorithm 61 Procedure "embed-TT".

Input(s): A two-tree (T_1, T_2) and a permanent link information function p.
Output(s): YES iff a crossing-free drawing of (T_1, T_2) respecting p can be obtained. Moreover, either (implicitly) a crossing-free drawing of (T_1, T_2) respecting p is found or a contradictory quadruple two-tree is produced.

```
     Let $r$ be the root of $T_2$.
     if $(T_1, T_2)$ has at most four leaves then
       return answer by table look-up
     else if $r$ has only one child then
 5:    Delete $r$ to produce $T_2'$.
       Modify $p$ accordingly, yielding $p'$.
       embed-TT($T_1, T_2', p'$);
     else
       {Let $\ell_1$ and $\ell_2$ be the two links emanating from the root $x$ of $T_2$.}
10:    if $p(\ell_1) = *$ then
         $p(\ell_1) := L$ and $p(\ell_2) := R$.
       end if
       {Let $T_2^L$ be the subtree of $T_2$ that is emanating from $\ell_L$ with $t(\ell_L) = L$;
       similarly, $T_2^R$ is defined.}
       Let $\Lambda_L := \Lambda(T_L)$ and $\Lambda_R := \Lambda(T_R)$.
15:    Initialize the bottom-up computation of new temporary information
       and (eventually) also permanent information $t$ and $p$ within $T_1$:
         • by setting $t(\ell) = L$ for each link leading to a leaf in $T_1$ with a
           label in $\Lambda_L$;
         • by setting $t(\ell) = R$ for each link leading to a leaf in $T_1$ with a
           label in $\Lambda_R$.
         • by setting $t(\ell) = *$ for each link leading not to a leaf in $T_1$.
       Update the temporary and permanent information within $T_1$ as de-
       scribed in the proof.
       if contradiction is reached then
         Report contradictory quadruple two-tree (obtainable as described in
         the proof).
         return NO
20:    else
         Let $T_1^L = T_1 \langle \Lambda_L \rangle$; $T_1^R = T_1 \langle \Lambda_R \rangle$;
         Let $p_L$ be the permanent information $p$ updated to cover the two-tree
         $(T_1^L, T_2^L)$; $p_R$ is accordingly understood.
         return embed-TT($T_2^L, T_1^L, p_L$) AND embed-TT($T_2^R, T_1^R, p_R$)
       end if
25:  end if
```

Let G be a subgraph of T. The set of attachment links of G, written $\mathfrak{A}(G)$, is the set of all attachment links of pairs of vertices $u, v \in V(G)$, i.e., $\mathfrak{A}(G) = \{\mathfrak{a}(u, v) \mid u, v \in V(G)\}$.

As above, we work with the permanent information $p(\ell)$ attached to an edge ℓ, initialized with $p = *$ and later gradually updated to either L or R. Sometimes, it is more convenient to think of this labeling information being attached to the inner nodes in the sense that a bit (called *flip*) is associated with each inner node that tells, when defined, which child is to the left and which is to the right.

A *connection link* is either a parent link or an attachment link; the corresponding set is denoted by $\mathfrak{C}(G) = \mathfrak{P}(G) \cup \mathfrak{A}(G)$. Given a two-tree (T_1, T_2), our algorithm will basically branch on all possible settings of p to either L or R on the connection links $\mathfrak{C}(G_i)$ (that do not contradict earlier settings $\neq *$) for the subgraphs $G_i = T_i\langle Q\rangle$ of T_i for all possible contradicting quadruples Q. Observe that whenever a parent link of some node is assigned L, then the parent link of its sibling will be assigned R for reasons of consistency (and vice versa).

The problem we are facing is that we have to ensure that the natural parameter of this problem, i.e., the given budget k of tolerable crossings, is decremented in each branching step. So, our strategy will be to only branch at contradicting structures if this gives us a gain in each branch. To simplify matters, we assume that only those leaves that participated in those contradicting structures we earlier branched on have been accounted for in the branching process.

To fully understand Alg. 62, we still have to explain what to do when there are no longer contradicting quadruples or pairs to be found (line 10): we will then start a procedure very similar to embed-TT. The only difference is that it will not find any of the contradictions described in the proof of Theorem 6.54 (since there are no contradicting quadruples). The only contradiction that could arise is that a temporary labeling would contradict the already established permanent labeling. In that case, the permanent labeling would already exist upon first calling embed-TT$'$, so that we face the situation depicted in Fig. 6.19. This figure is understood as follows: the green inner node indicates a flip that has been previously determined. As the labels L and R indicate, this flip would go the other way round according to the temporary link information propagation. The L,R-labeling must have a reason (otherwise, we could just interchange the meaning of L and R): namely, there must be a third leaf (the golden label) that is residing somewhere in the “right branch” of the upper tree but in the “left branch” in the lower tree (on a previous level of recursion). Hence, we also get a crossing if we draw the left part of lower tree coherently with the “green flip.” Upon finding such

Algorithm 62 Procedure "TTCM": embed a two-tree with minimum number of crossings.

Input(s): A two-tree (T_1, T_2) and a permanent link information function p. A parameter k. Let L be the set of labels that have been already treated and k' be the original parameter.

Output(s): YES iff a drawing of (T_1, T_2) with at most k crossings respecting p can be obtained.

```
    if $k < 0$ then
      return NO
    else if $k = 0$ then
      return embed-TT$(T_1, T_2, p)$
 5: else
      if there is a small contradicting structure $S$ then
        branch on all possible flips for $\mathfrak{C}(T_i\langle S\rangle)$ with recursive calls on
        TTCM, where $S$ is deleted from the new created instances and $p$
        and $L$ are accordingly modified.
        {To actually compute the new parameter value, $k'$ and the crossings
        in the two-tree $(T_1\langle L\rangle, T_1\langle L\rangle)$ may be used.}
      else
10:     return embed-TT$'(T_1, T_2, p, L, k, k')$
        {$L, k, k'$ are only needed if embed-TT$'$ recursively calls TTCM.}
      end if
    end if
```

an erroneous situation, we would consider the leaves labeled red, blue and golden as a contradicting structure and branch on all connection points as before, recursively calling TTCM again.

Unfortunately, the constant c in Theorem 6.53 is still quite astronomic due to the many flips that have to be tested according to our approach. We can see our result therefore as a preliminary one, producing a classification of the problem but not a really good algorithm.

As in the case of TWO-SIDED CROSSING MINIMIZATION or TWO-LAYER PLANARIZATION, it also makes sense to consider the case when the ordering on one layer is fixed:

Problem name: ONE-TREE CROSSING MINIMIZATION (OTCM)
Given: A two-tree (T_1, T_2) with leaf labels Λ, where the ordering of the vertices of T_1 is fixed
Parameter: a positive integer k
Output: Can (T_1, T_2) be drawn with at most k crossings ?

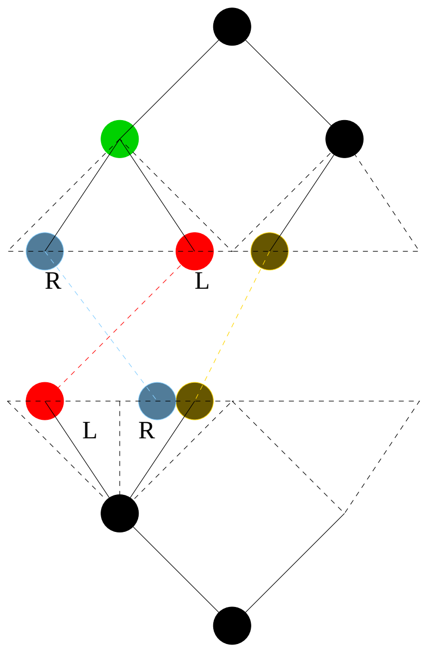


Figure 6.19: A further contradiction for embed-TT′.

We can show the following result:

Theorem 6.57 *In time $\mathcal{O}(n \log^2 n)$, we can solve the* ONE-TREE CROSSING MINIMIZATION *problem with n-leaf-trees.*

This improves on an earlier result of Dwyer and Schreiber [147].

Problem name: ONE-TREE DRAWING BY DELETING EDGES (OTDE)
Given: A binary tree T_1 with leaf labels Λ, a linear ordering $\prec$ on Λ
Parameter: a positive integer k
Output: Is there a label set $L \subseteq \Lambda$ with $|L| \leq k$ such that the tree $T_1 \langle \Lambda \setminus L \rangle$ can be drawn without crossings in the plane, so that the leaves in $\Lambda \setminus L$ are arranged according to the ordering $\prec$ on some line ?

Let us now turn to the corresponding graph-modification variant:
Some straightforward case analysis shows:

Lemma 6.58 *When the order of the leaf labels is fixed, then there is only one possible contradictory situation in a tree with three leaves, and this situation is shown in Fig. 6.20.*

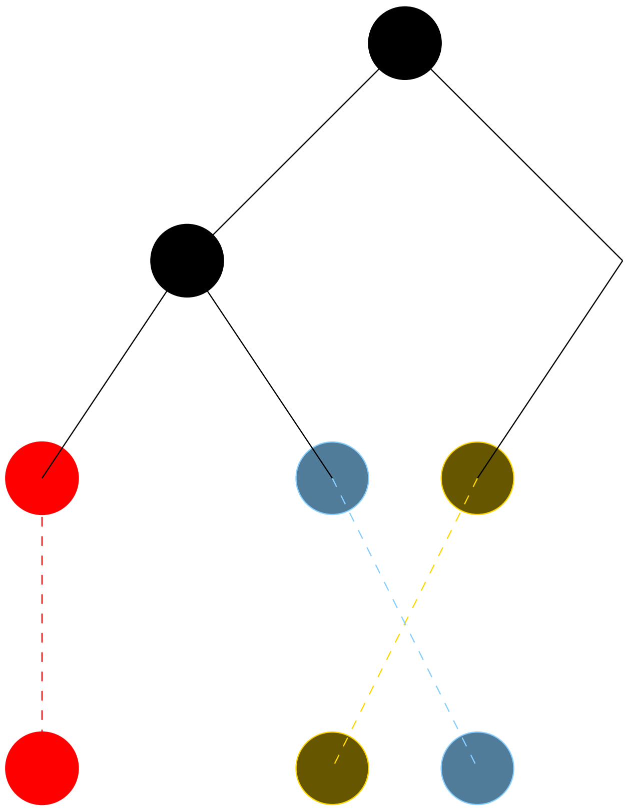


Figure 6.20: The contradicting triple tree.

In the following theorem formulation, we simply assume that only the ordering $\prec$ induced by the fixed tree on one side is given.

Theorem 6.59 *In each* ONE-TREE DRAWING BY DELETING EDGES *instance* $(T, \prec)$ *that is not embeddable without crossings, there exists a triple* $\{a, b, c\}$ *of leaf labels which is, with its induced tree structure* $T\langle\{a, b, c\}\rangle$ *also not embeddable without crossings (when respecting* $\prec$*).*

Again, we find small forbidden substructures (here: triple trees) that allows to transfer the problem to HITTING SET:

Corollary 6.60 ONE-TREE DRAWING BY DELETING EDGES *is in* $\mathcal{FPT}$. *More precisely, it can be solved in* $\mathcal{O}^*(2.179^k)$ *time.*

Interestingly, we do not know if TWO-TREE DRAWING BY DELETING EDGES or ONE-TREE DRAWING BY DELETING EDGES are $\mathcal{NP}$-hard. In view of our $\mathcal{FPT}$-results and due to the interest in these problems from bioinformaticians, this is a nice open problem in classical complexity theory.

6.5 Summary

We conclude this chapter by summarizing the methodology we explored up to now to obtain parameterized algorithms.

As a first step, to get acquainted with the problem, it is always useful to collect a couple of reduction rules and prove their correctness (soundness). The search for these rules can be guided by a couple of meta-heuristics. Let k be (as usual) the parameter.

- *Think big*: What happens if an instance if very big?

  If dealing with a graph problem, this could mean:

  - The instance is *globally big.* For example, this means that the instance has many vertices (compared to k).
  - The instance is *locally big.* For example, this means that the instance has vertices of high degree (compared to k).

  Then, one can often conclude that some properties must be true, say, certain vertices must go into a cover set we are going to construct. Conversely, if an instance is not big (in whatever sense), this often gives at least a hint how to proceed to actually find a set of rules that provides a kernelization. For example, in the case of VERTEX COVER, Buss' rule is a simple result of the idea of thinking (locally) big. Observe that Buss' rule on its own does not provide a complete kernelization, as there could be lots of isolate vertices left over. Here, the next idea comes into play (and is kind of complimenting):

- *Think small*: What happens if an instance is very small?

  - If the instance is *globally small*? Well, this is a rephrasement of the idea of kernelization itself, so usually then an instance can be solved by brute force.
  - The instance is *locally small.* For example, this means that the instance has vertices of low degree (compared to k).

  The consideration of locally small instances often gives particularly easy if not trivial reduction rules. For example, in the case of VERTEX COVER, we can state a rule that simply deletes isolates.

  Having thought both big and small, it is often possible to obtain a kernelization lemma of the following form:

Lemma 6.61 *(*schematic kernelization lemma*) If a problem instance is neither (locally) small nor locally big, it is globally small, i.e., of size limited by $f(k)$.*

For instance, Buss' rule and the isolates rule provide a quadratic kernel for VERTEX COVER.

In general especially the idea of thinking locally small or big turns out to be possibly surprisingly fruitful, so let us classify this idea a bit further. The general idea can be phrased as finding a *small* set of elements C such that one of the following possibilities is true:

1. All (optimal) solutions are containing C.

   Such rules can be useful for purposes of enumeration or counting (see Section 8.2).

   A typical example here is again Buss' reduction rule: if v is a vertex of degree $\deg(v) > k$, then v must go into *any* vertex cover of size at most k.

2. Some (optimal) solutions are containing C.

   An example of this type can be also found with VERTEX COVER, namely the crown rule: it might well be that there are minimal vertex covers that contain vertices from the crown, but at least one minimum vertex cover does not contain any crown vertices at all.

From the point of view of the classification we are discussing here, the VC reduction rule formed according the often quoted Theorem of Nemhauser and Trotter has undergone a kind of metamorphosis: in its original form (see Thm. 4.32), it was only clear that some minimum cover contains $C = C_{BP}^{\text{AND}}(G)$ (in the notation of that theorem), while only recently that theorem got sharpened: all minimum covers contain C, see Remark 4.37.

Usually, C actually forms a *problem kernel*, but this need not be the case; for what we said above, it would be sufficient to have the set C singled out; it might still be useful to keep some other parts of the instance for checking purposes.

In this more general form, the search tree development technique known as forming *candidate set*s C is quite similar. What is the decisive property of a candidate set (for a decision problem)? It should be the case that there is some (optimal) solution that contains at least one of the candidates. For the purposes of enumeration or counting (discussed in Chap. 8), we would need a stronger form of candidate set: all (optimal) solutions should contain at least one of the candidates. Whenever we found such a candidate set and when this set is small (in the sense that its size only depends on the parameter or, even better, is only bounded by a constant), then we can "afford" branching according to all possibilites. Notice that some of our most demanding results are in fact theorems that bound the size of a (natural) candidate set.

In the next phase of algorithm design, one should look for improvements of the current algorithm. This could

- either be done by a computer-aided approach to look for further improvements on the search tree part, as developed in [211, 161],
- or it is "manually" done. We have seen several examples of the latter methodology in this Habilitationsschrift.

We have in passing discussed another classification of search tree development in the preceding two chapters:

- In what we called the bottom-up approach to search trees, the branching on candidate sets is refined by looking more deeply into local situations. The complicated thing is then to prove the correctness of the resulting algorithm, i.e.: are still all cases covered by the suggested branching? The computer-aided approach mentioned above tries to mechanize the search for better branchings according to this methodology.
- In a suitably tweaked top-down analysis, we try to keep the algorithm structure simple, but we add further reduction rules and branch according to heuristic priorities (notice that the sequence of branchings on candidate sets bears a potential source of nondeterminism). The tricky part is then to prove the claimed bounds on the search tree size.

Chapter 7

Graph parameters

In this chapter, we discuss parameterizations that qualify, as discussed in Chap. 3, as *implicit internal* namely, parameters like treewidth, branchwidth, etc. as defined for graphs. However, they are of utmost interest for the development of efficient parameterized algorithms:

1. When parameterized with say the treewidth of a graph, problems on graphs tend to be (to say the least) parameterized tractable by means of dynamic programming techniques. We will refer to such algorithms (e.g.) as *treewidth-based algorithms*.

2. Often, relations between the actual entity of a graph we are interested in and the (tree)width of a graph can be shown, and this relation can be turned into an algorithm that uses the treewidth-based algorithm mentioned in 1.

3. Sometimes, practice shows that certain graphs simply are of bounded treewidth, which might even lead to problems that are efficiently solvable in the classical, non-parameterized sense. A nice example can be found in [369], where the treewidth of graphs induced by structured programs is investigated.

4. Another "non-standard application" is exhibited in [195, 215, 217], where M. Grohe and his coauthors showed (amongst others) that certain model-checking and database problems become tractable when viewed from the angle of treewidth. Connection to logical problems are also discussed in [179, 286].

It is also worthwhile mentioning that recently structural connections between the treewidth of a graph and certain properties of the so-called Sherali-Adams reformulation procedure for 0/1 integer programs was obtained [48];

such insights might help link treewidth-based algorithms with another, practically very successful technique for solving hard problems, namely (integer) linear programming (ILP).

In order to keep this exposition short, we will mainly focus on one of the mentioned graph parameters, namely treewidth. Similar techniques and comments apply to other parameters, as well. More details on further parameters can be found in the following papers:

branchwidth The use of branchwidth (as opposed to treewidth) for obtaining parameterized algorithms was initiated by Fomin and Thilikos [192, 189]. Admittedly, the most efficient parameterized algorithms for VERTEX COVER and for DOMINATING SET on planar graphs are actually based on branchwidth. However, we still assume that treewidth is a better known and (arguably) easier understandable notion, so that we focus on that parameter here.

local treewidth The notion of treewidth has be generalized towards a local variant (which basically means that bounded treewidth should hold in any neighborhood). This notion has been under extensive study from the parameterized algorithmics community in recent years, see [121, 122, 112, 191, 216].

cliquewidth The notion of *cliquewidth* also allows for efficient algorithms; however, besides being a natural *implicit internal* parameter, we don't know of any applications that use cliquewidth (replacing treewidth or branchwidth) to obtain parameterized algorithms for problems that are parameterized in the "usual way." The corresponding body of literature is also quite impressive; the foundations were laid by Courcelle with his series of (now 16) papers starting with "The monadic second-order logic of graphs" see [102, 103] for the latest published entries. Further rather recent works that may serve for a start are: [55, 104, 105]. One of the drawbacks is probably that there is no known efficient recognition algorithm for graphs of bounded cliquewidth. [1]

For a nice source on graphs of bounded cliquewidth, see [Chapter 11, Site 5]. The notions of treewidth, of bandwidth, and of cliquewidth have been recently linked for some special graph classes in [280].

[1]There seems to be a recent break-through in [316] where an algorithm is presented that either shows that the given graph does not have cliquewidth k or it gives a decomposition that shows that the graph has at most cliquewidth $2^{O(k)}$. This might make cliquewidth amenable to the parameterized approach.

outerplanarity Due to the deep connections between outerplanarity and treewidth (as explained below), we won't elaborate on specific dynamic programming algorithms; details on such dynamic programming for VERTEX COVER can be found in the seminal work of Baker [32]. For DOMINATING SET, a specific algorithm (supplementing the paper of Baker) has been designed [253] that is useful for better parameterized algorithms for PLANAR DOMINATING SET.

Let us furthermore two concepts that are very much related to what we are presenting in this chapter.

bidimensionality The recent sequence of papers of Demaine, Fomin and their coauthors have provided solid grounds to see *bidimensionality* as a further basic technique in this area. We can only refer the interested reader to these papers here [112, 117, 120, 118, 191].

non-planar graphs We will mostly consider planar graphs in what follows. However, there are many related graph classes that can be treated alike, as has been exposed in many recent papers. The following list is probably far from exhaustive: [121, 117, 113, 115, 120, 119].

This chapter is organized as follows: In Section 7.1, we explain the notion of treewidth. How this can be used to obtain fixed-parameter results is exhibited in Sec. 7.2. We then show in Sec. 7.3 how treewidth relates to the structural properties of planar graphs. Section 7.4 contains a more concrete example, a(nother) parameterized algorithm for PLANAR DOMINATING SET. The next two sections, Sec. 7.5 and 7.6, show more general techniques how the ideas presented so far can be used to obtain parameterized algorithms of the form $\mathcal{O}^*(c^{\sqrt{k}})$. In Sec. 7.7, we once more return to PDS, exhibiting how the involved (huge) constants can be significantly lowered. Section 7.8 explains how these techniques can be used for related problems like RED-BLUE DOMINATING SET and FACE COVER. We conclude with a section shortly discussing $\mathcal{FPT}$ results for graph classes different from planar graphs.

A nice presentation of most results that are exhibited in this chapter is also contained in J. Albers PhD dissertation [7].

7.1 Treewidth

We are now going to study the basic notions of a tree decomposition, and based thereupon, of the treewidth of a graph. These notions were originally introduced by Robertson and Seymour along their way of developing graph minor theory, see [340, 341], although basically the same notions have been independently developed by others under the name of *partial k-trees* [27, 363, 364, 365] and algorithms for *recursive graph* classes [58]. More recent surveys include [54, 53, 261].

Most of the results of this section (and the subsequent ones) are, more concretely, from [8, 14, 15, 179].

Definition 7.1 Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle\{X_i \mid i \in I\}, T\rangle$, where each X_i is a subset of V, called a *bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:

1. $\bigcup_{i\in I} X_i = V$;
2. for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;
3. for all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The *width of the tree decomposition* $\langle\{X_i \mid i \in I\}, T\rangle$ equals

$$\max\{|X_i| \mid i \in I\} - 1.$$

The *treewidth* of G is the minimum k such that G has a tree decomposition of width k, also written $\operatorname{tw}(G)$ for short.

A tree decomposition with a particularly simple structure is given by the following definition.

Definition 7.2 A tree decomposition $\langle\{X_i \mid i \in I\}, T\rangle$ with a distinguished root node r is called a *nice tree decomposition* if the following conditions are satisfied:

1. Every node of the tree T has at most 2 children.
2. If a node n has two children n' and n'', then $X_n = X_{n'} = X_{n''}$ (in this case n is called a *join node*).
3. If a node n has one child n', then either
   (a) $|X_n| = |X_{n'}| + 1$ and $X_{n'} \subset X_n$ (in this case n is called an *insert node* or an *introduce node*), or
   (b) $|X_n| = |X_{n'}| - 1$ and $X_n \subset X_{n'}$ (in this case n is called a *forget node*).

Observe that each node in a nice tree decomposition is either a join node, an insert node, a forget node, or a leaf node. In [54], it is—in addition—required for a nice tree decomposition that each leaf node has a bag of size one associated to it; from the point of view of presentation of algorithms, this might further simplify the exposition, since the initialization action in leaf nodes would become completely trivial. More precisely, the leaf node action required by our somewhat broader definition in a leaf node with bag X can be replaced by a sequence of at most $|X|$ nodes, one of them (the new leaf) having a bag containing only one element and the other nodes being insert nodes. Conversely, one could also require that the bag that is associated to the root is empty. Then, the final value of the computation can be trivially looked up from the table of the root.

It is not hard to transform a given tree decomposition into a nice tree decomposition. More precisely, the following result holds (see [261, Lemma 13.1.3])

Lemma 7.3 *Given a tree decomposition of a graph G that has width k and $\mathcal{O}(n)$ nodes, where n is the number of vertices of G. Then, we can find a nice tree decomposition of G that has also width k and $\mathcal{O}(kn)$ nodes in time $\mathcal{O}(kn)$.*

Related to the notion of a tree decomposition is the following one:

Definition 7.4 Let $G = (V, E)$ be a graph. A *path decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T\rangle$ such that $\langle \{X_i \mid i \in I\}, T\rangle$ is a tree decomposition of G where T is a path.

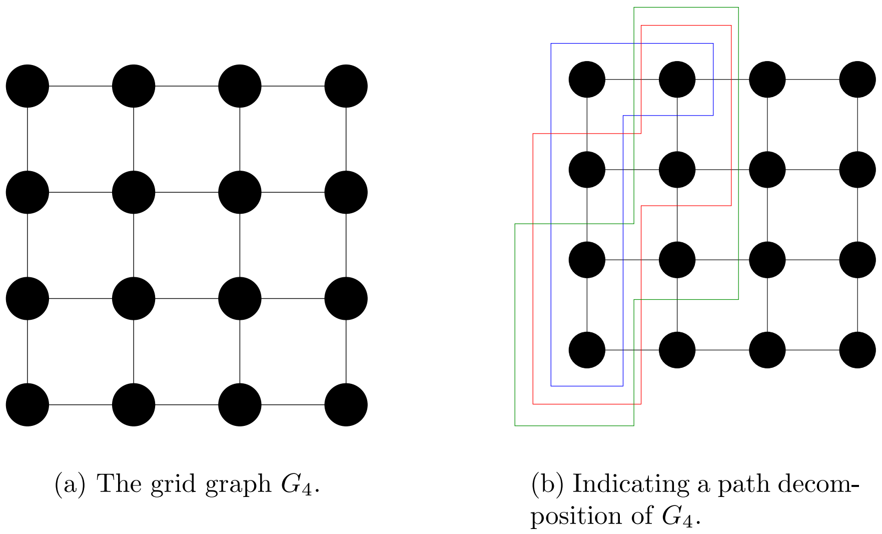
Accordingly, the *pathwidth* of G is the minimum k such that G has a path decomposition of width k.

Quite recently, an alternative notion—*persistence*— was coined [139]; however, recognition of graphs of persistence pathwidth bounded by k is parameterized intractable (in contrast to what we know about the corresponding notions based on treewidth), so that we do not elaborate on this notion here, although it might actually better capture the intuition of path-similarity that is behind the notion of pathwidth (as also discussed in other places in this book).

Let us exemplify these notions with some examples:

Example 7.5 Let us consider the $n \times n$ *grid graph.* Formally, this graph $G_n = (V_n, E_n)$ can be defined as follows:

- $V_n = \{(i, j) \mid 1 \leq i, j \leq n\}$. V_n can be viewed as indices of a square-like grid.


(a) The grid graph G_4.

(b) Indicating a path decomposition of G_4.

Figure 7.1: Grids: graphs with pathwidth $\mathcal{O}(\sqrt{|V|})$.

- $E_n \ni \{(i,j),(i',j')\}$ if one of the following condition is true:
  - $i' = i+1$ and $i < n$ and $j' = j$, OR
  - $j' = j+1$ and $j < n$ and $i' = i$.

  There are no other edges in E_n than the ones described.

The graph G_4 is shown in Fig. 7.1(a).

How does a tree decomposition of G_4 look like? In actual fact, we are going to describe a path decomposition of that graph in what follows. We will use $\mathcal{O}(|V_n|) = \mathcal{O}(n^2)$ many bags. We "walk" along our graph in a column-wise fashion, starting with the first column, and within each column, in a row-wise manner, starting with the first row. This way, we visit the vertices of the graph in the following way:

$$(1,1),(2,1),\dots,(n,1),(1,2),(2,2),\dots,(n,1),(n,2),\dots,(n,n).$$

Hence, we can speak about the ith vertex x_i that we visit; for instance, $(2,2)$ is the $(n+2)$nd visited vertex. Let now X_i, $1 \le i \le n^2 - n$ collect all vertices that are visited in steps $i,\dots,i+n$. Considering the path P given by $X_1, X_2, \dots, X_{n^2-n}$ in that order, the reader may verify that

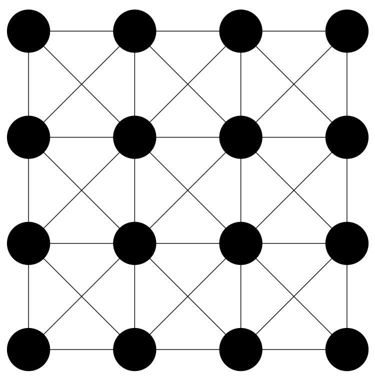
$$\langle \{X_i \mid 1 \le i \le n^2 - n\}, P\rangle$$

is a path decomposition of G_n. Hence, both the pathwidth and the treewidth of G_n are upperbounded by n. Let us assume in the following that X_1 is the root of this special tree.

In the concrete example of G_4, we outlined the first four bags by color-coding in Fig. 7.1(b), coloring them, in sequence, blue, red, and green.

Observe that this tree decomposition is not a nice tree decomposition. But it is an easy task to transform the given one into a nice one: just introduce the bags $X_i' = X_i \setminus \{x_i\}$ inbetween X_i and X_{i+1}.

This makes the node corresponding to X_i an insert node (since $X_i' \subseteq X_i$ and X_i' is the child of X_i), except the very last X_i, and X_i' is a forget node (since $X_i' \subseteq X_{i+1}$ and X_{i+1} is a child of X_i').


(a) The grid graph G_4' with diagonals.

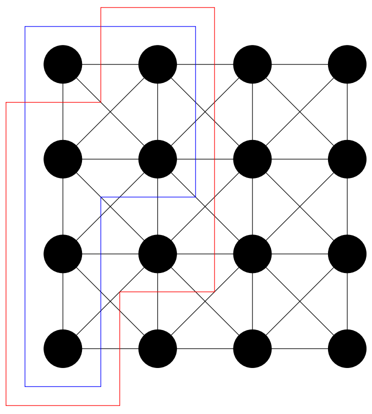

(b) Indicating a path decomposition of G_4'.

Figure 7.2: Grids with diagonals: graphs with pathwidth $\mathcal{O}(\sqrt{|V|})$.

Example 7.6 Let us consider the $n \times n$ *grid graph with diagonals*. Formally, this graph $G_n' = (V_n, E_n')$ can be defined as follows:

- $V_n = \{(i,j) \mid 1 \leq i,j \leq n\}$. V_n can be viewed as indices of a square-like grid.
- $E_n' \ni \{(i,j),(i',j')\}$ if one of the following condition is true:
  - $i' = i + 1$ and $i < n$ and $j' = j$, OR
  - $j' = j + 1$ and $j < n$ and $i' = i$, OR
  - $i' = i + 1$ and $i < n$ and $j' = j + 1$ and $j < n$, OR

– $i' = i + 1$ and $i < n$ and $j' = j - 1$ and $j > 1$.

There are no other edges in E'_n than the ones described.

The graph G'_4 is shown in Fig. 7.2(a).

How does a tree decomposition of G'_4 look like? Two bags of a possible path decomposition are shown in Fig. 7.2(b). As can be seen, the bags are one element larger than in the previous, similar example. The reason is that otherwise the diagonal connections $\{(i, j), (i + 1, j + 1)\}$ would not be contained in any bag. The formal details of the indicated path decomposition are left to the reader. Anyways, the upperbound of the treewidth is again linearly growing with n, i.e., with the square root of the number of vertices of G'_n.

Grid graphs and their "relatives" are not arbitrarily chosen as examples for tree decompositions; the papers on bidimensionality mentioned above, along with [339] can be also read as showing that grid graphs are—in a sense—the essential examples of planar graphs. As we will see in Sec. 7.3, in fact all planar graphs enjoy the property of having a treewidth limited essentially by the square root of the number of its vertices.

Let us remark (once more) the relation between pathwidth and other graph parameters like cutwidth (see Chap. 3). From an algorithmic perspective, the efficient computability of cutwidth in bounded degree graphs of small treewidth established in [367] is remarkable in this respect.

Let us close our list of examples with two examples that appear to be less symmetric than the previous ones:

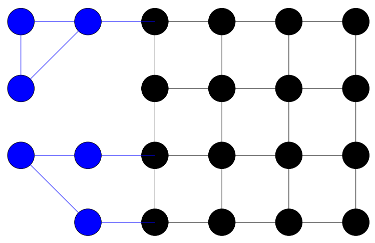


Figure 7.3: An grid with some additions in blue.

Example 7.7 Fig. 7.3 basically shows again the grid graph G_4 (drawn in black) together with some "additions", drawn in blue. These additions are attached to the G_4 by either a vertex or an edge. This means that, whatever tree decomposition we have found for G_4, we can modify it to get a valid tree decomposition of the whole graph as follows:

- Find a bag that contains the attachment vertex or the attachment edge.
- Connect this bag to a bag that only contains the attachment vertex or edge.
- From this starting point, attach a tree decomposition of the two parts of the blue addition.

It is an easy exercise to see that the overall width of the tree decomposition is not increased by the chosen attachments. Observe that we actually get a *tree* decomposition this way, not (again) a path decomposition.

Finally, let us turn to the concrete problem to find a *nice* tree decomposition.

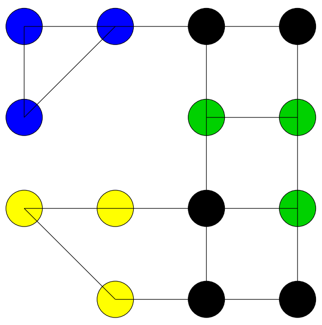


Figure 7.4: A modified grid.

Example 7.8 Let us have a look at Fig. 7.4. It shows a graph G whose vertices can be addressed, similar to the grid G_4, by pairs of numbers (i, j), with $1 \leq i, j \leq 4$. It is actually a simplification of the graph considered in the previous example.

How would a nice tree decomposition of width two look like? The two leaf nodes would correspond to bags drawn in blue and in yellow in Fig. 7.4. For example, the "blue leaf" would contain the bag with the vertices $(1, 1)$, $(1, 2)$, and $(2, 1)$ of G. The sequence of ancestor nodes could contain the following vertices in their corresponding bags:

- $(1, 1)$, $(1, 2)$;
- $(1, 1)$, $(1, 2)$, $(1, 3)$;
- $(1, 2)$, $(1, 3)$;

- $(1,2)$, $(1,3)$, $(1,4)$;
- $(1,3)$, $(1,4)$;
- $(1,3)$, $(1,4)$, $(2,3)$;
- $(1,4)$, $(2,3)$;
- $(1,4)$, $(2,3)$, $(2,4)$;
- $(2,3)$, $(2,4)$;
- $(2,3)$, $(2,4)$, $(3,4)$.

The last mentioned bag is colored green in the figure. The "yellow leaf" would contain the bag with the vertices $(3,1)$, $(3,2)$, and $(4,2)$ of G. The sequence of ancestor nodes could contain the following vertices in their corresponding bags:

- $(3,2)$, $(4,2)$;
- $(3,2)$, $(4,2)$, $(3,3)$;
- $(4,2)$, $(3,3)$;
- $(4,2)$, $(3,3)$, $(4,3)$;
- $(3,3)$, $(4,3)$;
- $(3,3)$, $(4,3)$, $(4,4)$;
- $(3,3)$, $(4,4)$;
- $(3,3)$, $(4,4)$, $(3,4)$;
- $(3,3)$, $(3,4)$;
- $(3,3)$, $(3,4)$, $(2,3)$;
- $(3,4)$, $(2,3)$;
- $(2,3)$, $(2,4)$, $(3,4)$.

In our case, the tree decomposition can be either interpreted as a path decomposition or as a "true" tree decomposition, where a possible root is colored green.

7.2 Dynamic programming

Let us show how to devise dynamic programming algorithms for graphs that are given together with a suitable tree decomposition. More general scenarios for developing treewidth-based algorithms are exposed in [27, 134, 225, 364, 363, 365].

We shall focus on the following four problems:

1. VERTEX COVER: In fact, since we are not putting emphasis on the standard parameter, basically the same dynamic programming approach solves INDEPENDENT SET and CLIQUE, as well.

2. DOMINATING SET: While the dynamic programming solution of VERTEX COVER is close to trivial, this is no longer the case with DOMINATING SET; in fact, the constants claimed in [9] were flawed due to a misreading of earlier published results, and the corrections (as explained below) published in the journal version [8] of that extended abstract contain ideas that were used in other dynamic programming scenarios, as well [193].

3. ROMAN DOMINATION: We consider this problem, since it is—first of all—one of the seemingly uncountable number of variations of domination problems. Secondly, it has a nice story coming with it; namely, it should reflect the idea of how to secure the Roman Empire by positioning the armies (legions) on the various parts of the Empire in a way that either a specific region r is also the location of at least one army or one neighboring region r' has two armies, so that it can afford sending one army to the region r without an army (in case of an attack) without loosing the capability to defend itself (since the second army located in r' will stay there). Thirdly, the dynamic programming algorithm previously published in [318] appears to be incorrect.

4. DOMINATING SET OF QUEENS: Here, we will see how we can also establish an efficient solution to the original problem (i.e., not a problem that is "re-parameterized" by treewidth) through a treewidth-based algorithm.

We have seen all the problems in further sections; let us briefly remind the possibly least known of these problems, ROMAN DOMINATION:

Problem name: ROMAN DOMINATION (ROMAN)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *Roman domination* function R such that $\sum_{x \in V} R(x) \leq k$?

Let us first link ROMAN DOMINATION with DOMINATING SET:

Lemma 7.9 *If $D \subseteq V$ is a Roman domination set for $G = (V, E)$ (with respect to a Roman domination function R, i.e., $D = D_R$), then D is also a dominating set. Moreover, if $\sum_{x \in D_R} R(x) \leq k$, then $|D| \leq k$.*

Of course, in the context of this chapter, we will be also interested in an alternative parameterization, namely a *parameterization by the treewidth* of the underlying graph.

More specifically, a typical problem of interest would be the following one:

Problem name: VERTEX COVER, PARAMETERIZED BY TREEWIDTH (VCTW)
Given: A graph $G = (V, E)$ together with some tree decomposition
Parameter: a positive integer k that equals the width of the tree decomposition
Output: What is the size of a minimum *vertex cover* $C \subseteq V$?

Theorem 7.10 *An instance $G = (V, E)$ of* VERTEX COVER, PARAMETERIZED BY TREEWIDTH *can be solved in time $\mathcal{O}(2^{\text{tw}(G)}|V|)$.*

Proof. Recall that we can assume that the graph G is given by a nice tree decomposition with underlying tree T. Arbitrarily, we select a root r in T.

We will maintain a table for each node n in this decomposition. The rows of this table can be viewed as corresponding to mappings $\alpha : X \to \{0, 1\}$, where X is the bag that is associated to the node n. The value $\sum_{x \in X} \alpha(x)$ can interpreted as being the number of vertices that go into the vertex cover from bag X, assuming the particular assignment α. In other words, each vertex x can carry two values, 1 and 0, reflecting that x is going into the vertex cover (if $\alpha(x) = 1$) or not (if $\alpha(x) = 0$).

Let $T[n]$ be the subtree of T that contains n and all nodes in T that belong to the component of $T - n$ to which r does not belong. Moreover, $G\langle n \rangle$ is the subgraph of G that is induced by the union of all bags that correspond to nodes of $T[n]$.

In the table for n, we store

Algorithm 63 A schematics for algorithms based on a given tree decomposition.

Input(s): a graph G, together with a tree decomposition
Output(s): Whatever to be solved...

```
Find a nice tree decomposition $TD = \langle\{X_i \mid i \in I\}, T\rangle$ of $G$.
Perform a depth-first search along $T$, choosing an arbitrary root $r$ for $T$
as starting point; i.e., call: dfs($G, TD, r$).
Read off a solution from the table that belongs to $r$.

subroutine dfs($G, TD, n$)
if current node $n$ is a leaf then
  perform leaf node actions and return corresponding table
else if current node $n$ has one child $n'$ then
  table$_{n'}$ :=dfs($G, TD, n'$)
  if $n$ is forget node then
    perform forget node actions and return corresponding table, based on
    table$_{n'}$
  else
    perform insert node actions and return corresponding table, based on
    table$_{n'}$
  end if
else
  {Current node $n$ has two children $n'$ and $n''$}
  table$_{n'}$ :=dfs($G, TD, n'$)
  table$_{n''}$ :=dfs($G, TD, n''$)
  perform join node actions and return corresponding table, based on
  table$_{n'}$ and on table$_{n''}$
end if
```

- either the number that corresponds to the size of a minimum vertex cover of $G\langle n\rangle$ that is consistent with a particular assignment α on the bag X associated to n,

- or we store ∞ in the case that there is no way to construct a valid vertex cover assuming the assignment α.

Formally, the fact that the tables constructed as described below verify these properties can be easily shown by tree induction following the algorithm details that follow; the leaf node description would correspond to the induction basis, and the other three cases cover the induction step. Finally, this then implies that we can determine the size of a minimum vertex cover of

$G = G\langle r\rangle$ by one final sweep through the table that has been computed for the root node r.

In a nice tree decomposition, we have to specify how to deal with the following four situations:

leaf nodes At a leaf node with bag X, we build up a table that stores, for all mappings $\alpha : X \to \{0,1\}$, the value $\sum_{x\in X}\alpha(x)$, interpreted as being the number of vertices that go into the vertex cover, assuming the particular assignment α. In the case that a particular α assigns 0 to to neighboring vertices $x, y \in X$, ∞ will be stored in the corresponding table entry.

forget nodes We have two bags: the parent bag X we currently work on, and an already processed child node with associated bag Y with a corresponding table table$_Y$. By definition, $Y \cup \{x\} = X$. Consider a specific mapping $\alpha : X \to \{0,1\}$. The value α_Y of the restriction of α to Y can be looked up in table$_Y$. If $\alpha(x) = 0$, the value of α in table$_X$ is α_Y iff, for all vertices $z \in N[x] \cap X$, $\alpha(z) = 1$; otherwise, the value of α would be ∞. If $\alpha(x) = 1$, the value of α in table$_X$ is α_Y plus one. Observe that a table entry ∞ in table$_X$ is possible for a specific mapping α if either table$_Y(\alpha_Y) = \infty$ or α assigns zero to all vertices from $N[x] \cap X$.

insert nodes We have again two bags: the parent bag X we currently work on, and an already processed child node with associated bag Y with a corresponding table table$_Y$. By definition, $X \cup \{x\} = Y$. Consider a specific mapping $\alpha : X \to \{0,1\}$. α can be extended in two ways to a mapping on Y: $\alpha_{x=i}$ is identical to α on X and defines $\alpha_{x=i}(x) = i$ for $i = 0, 1$. The value of α that should be put into table$_X$ is then

$$\min\{\mathrm{table}_Y(\alpha_{x=0}), \mathrm{table}_Y(\alpha_{x=1})\}$$

join nodes We have now three nodes: the parent node n and the two children nodes n' and n''. To all these nodes, the same bag X is associated.

Since the validity of the assignments was already tested in the children nodes, we can compute the table entry table$_n$ for assignment $\alpha : X \to \{0,1\}$ from the corresponding tables table$_{n'}$ and table$_{n''}$ for the same assignment as follows:

$$\mathrm{table}_n(\alpha) = \mathrm{table}_{n'}(\alpha) + \mathrm{table}_{n''}(\alpha) - \sum_{x\in X}\alpha(x).$$

Observe that if α is invalid for one of the children n or n', then at least one of the table entries $\text{table}_{n'}(\alpha)$ or $\text{table}_{n''}(\alpha)$ is ∞, and this will be inherited to $\text{table}_{n}(\alpha)$ by the usual computation rules.

How to combine these ingredients within a general schematics for solving tree decomposition based algorithms is shown in Alg. 63. The claimed running time for VERTEX COVER, PARAMETERIZED BY TREEWIDTH follows from the fact that there are only $\mathcal{O}(|V|)$ many tree nodes to be processed according to Lemma 7.3 and that for each node, each of the at most $2^{\text{tw}(G)}$ many table entries incurs only a constant processing time (assuming sufficiently quick access to the table entries in the children nodes' tables). ∎

Let us illustrate this algorithm for computing a minimum vertex cover by continuing with Example 7.8.

Example 7.11 Let us start with listing a small table that contains all possible choices for the leaf node whose bag is colored blue in Fig. 7.4:

$(1,1)$	$(1,2)$	$(2,1)$	table
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	2
1	0	0	∞
1	0	1	2
1	1	0	2
1	1	1	3

Its parent node is a forget node and contains the bag with the vertices $(1,1)$ and $(1,2)$. The accordingly smaller table looks as follows:

$(1,1)$	$(1,2)$	table
0	0	∞
0	1	2
1	0	2
1	1	2

This table was obtained from the previous one basically by minimizing over two neighboring rows. The next parent node to be processed has the bag with the vertices $(1,1)$, $(1,2)$, and $(1,3)$ and is hence an insert node, followed by another forget node.

(1,1)	(1,2)	(1,3)	table
0	0	0	∞
0	0	1	∞
0	1	0	2
0	1	1	3
1	0	0	∞
1	0	1	3
1	1	0	2
1	1	1	3

(1,2)	(1,3)	table
0	0	∞
0	1	3
1	0	2
1	1	3

Adding $(1, 4)$ gives us again first an insert and then a forget node:

(1,2)	(1,3)	(1,4)	table
0	0	0	∞
0	0	1	∞
0	1	0	3
0	1	1	4
1	0	0	∞
1	0	1	3
1	1	0	3
1	1	1	4

(1,3)	(1,4)	table
0	0	∞
0	1	3
1	0	3
1	1	4

Adding $(2, 3)$ gives us again first an insert and then a forget node; not to loose sight of the optimal solutions, we start adding them in one extra column:

(1,3)	(1,4)	(2,3)	table
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	4
1	0	0	3
1	0	1	4
1	1	0	4
1	1	1	5

(1,4)	(2,3)	table	a corresponding solution
0	0	3	$\{(1,1),(1,2),(1,3)\}$
0	1	4	$\{(1,1),(1,2),(1,3),(2,3)\}$
1	0	4	$\{(1,1),(1,2),(1,3),(1,4)\}$
1	1	4	$\{(1,1),(1,2),(2,3),(1,4)\}$

The final change in this branch is dealing with adding $(2, 4)$ and then $(3, 4)$ to our considerations:

(1,4)	(2,3)	(2,4)	table
0	0	0	∞
0	0	1	4
0	1	0	∞
0	1	1	5
1	0	0	∞
1	0	1	5
1	1	0	4
1	1	1	5

(2,3)	(2,4)	table
0	0	∞
0	1	4
1	0	4
1	1	5

(2,3)	(2,4)	(3,4)	table
0	0	0	∞
0	0	1	∞
0	1	0	4
0	1	1	5
1	0	0	∞
1	0	1	5
1	1	0	5
1	1	1	6

The minimal solutions we might store in addition are, in the sequence of non-∞ table entries:

- $\{(1,1),(1,2),(1,3),(2,4)\}$,
- $\{(1,1),(1,2),(1,3),(2,4),(3,4)\}$,
- $\{(1,1),(1,2),(1,3),(2,3),(3,4)\}$,
- $\{(1,1),(1,2),(1,3),(2,3),(2,4)\}$,
- $\{(1,1),(1,2),(1,3),(2,3),(2,4),(3,4)\}$.

We can make similar considerations starting with the yellow-colored bag as pertaining to the other leaf node. This would first affect the vertices $(3,1)$, $(3,2)$, $(4,2)$, and then $(3,3)$:

$(3,1)$	$(3,2)$	$(4,2)$	table
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	2
1	1	1	3

$(3,2)$	$(4,2)$	table
0	0	1
0	1	2
1	0	2
1	1	2

$(3,2)$	$(4,2)$	$(3,3)$	table
0	0	0	∞
0	0	1	2
0	1	0	∞
0	1	1	3
1	0	0	2
1	0	1	3
1	1	0	2
1	1	1	3

Now, the vertex $(4,3)$ comes into play:

$(4,2)$	$(3,3)$	table
0	0	2
0	1	2
1	0	2
1	1	3

$(4,2)$	$(3,3)$	$(4,3)$	table
0	0	0	∞
0	0	1	3
0	1	0	∞
0	1	1	3
1	0	0	∞
1	0	1	3
1	1	0	3
1	1	1	4

$(3,3)$	$(4,3)$	table
0	0	∞
0	1	3
1	0	3
1	1	3

Possible solutions to the three non-∞ table entries are, in this sequence:

- $\{(3,1),(3,2),(4,3)\}$,
- $\{(3,2),(4,2),(3,3)\}$,
- $\{(3,1),(3,3),(4,3)\}$.

Then, vertices $(4,4)$ and $(3,4)$ are treated:

$(3,3)$	$(4,3)$	$(4,4)$	table
0	0	0	∞
0	0	1	∞
0	1	0	3
0	1	1	4
1	0	0	∞
1	0	1	4
1	1	0	3
1	1	1	4

$(3,3)$	$(4,4)$	table
0	0	3
0	1	4
1	0	3
1	1	4

$(3,3)$	$(4,4)$	$(3,4)$	table
0	0	0	∞
0	0	1	4
0	1	0	∞
0	1	1	5
1	0	0	∞
1	0	1	4
1	1	0	4
1	1	1	5

The non-∞ entries of the following table correspond to minimal solutions as listed below:

- $\{(3,1),(3,2),(4,3),(3,4)\}$,
- $\{(3,1),(3,2),(4,3),(4,4)\}$,
- $\{(3,1),(3,2),(4,3),(3,4),(4,4)\}$.

Then, we have to integrate $(2,3)$:

$(3,3)$	$(3,4)$	table
0	0	∞
0	1	4
1	0	4
1	1	5

$(3,3)$	$(3,4)$	$(2,3)$	table
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	5
1	0	0	4
1	0	1	5
1	1	0	5
1	1	1	6

$(3,4)$	$(2,3)$	table
0	0	4
0	1	5
1	0	5
1	1	5

Possible solutions to the table entries are, in this sequence:

- $\{(3,1),(3,3),(4,3),(4,4)\}$,
- $\{(3,1),(3,3),(4,3),(4,4),(2,3)\}$,
- $\{(3,1),(3,3),(4,3),(4,4),(3,4)\}$,
- $\{(3,1),(3,2),(4,3),(3,4),(2,3)\}$.

Adding $(2,4)$ gives the following table, to which side we put the table values that we obtained in the other branch, as well as the final values of the root of this tree decomposition; we use color-coding to denote these

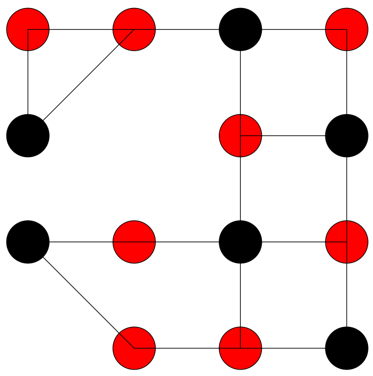


Figure 7.5: An optimal vertex cover of the modified grid.

branches. Therefore, we get as optimal size for a vertex cover: eight, and there are obvious various different solutions attaining this value.

(2, 3)	(2, 4)	(3, 4)	table	table	table	a possible minimal solution
0	0	0	∞	∞	∞	—
0	0	1	∞	∞	∞	—
0	1	0	5	4	8	L_1
0	1	1	6	5	9	—
1	0	0	∞	∞	∞	—
1	0	1	5	5	8	L_2
1	1	0	6	5	9	—
1	1	1	6	6	9	—

A possible optimal solution can be seen in Fig. 7.5; the red vertices form a vertex cover. From the table, we can in fact read off two alternative minimum solutions:

$$\begin{aligned} L_1 &= \{(1,1),(1,2),(1,3),(3,1),(3,3),(4,3),(4,4),(2,4)\} \\ L_2 &= \{(1,1),(1,2),(1,4),(2,3),(3,4),(3,1),(3,2),(4,3)\} \end{aligned}$$

Observe that the solution given in Fig. 7.5 is different from the ones whose computation is sketched in the tables. The reason is that in the table computations, sometimes minimal solutions are "forgotten" if there are other minimal solutions that are no worse when looked at them from the perspective of the corresponding "interface bag."

Let us now turn to DOMINATING SET: is it (again) sufficient to only store two possible "states" per graph vertex, i.e., being in the dominating set or not? Unfortunately, this is not the case, since along the course of a treewidth-based algorithm, we may encounter vertices in a bag that are not in the dominating set according to the specific assignment under consideration in the present bag, nor they are dominated; however, they might be already "happy" since they have already been dominated according to a previous assignment (that has been already "forgotten" within the present bag) or they are still "unhappy", i.e., they still have to be dominated by vertices that are not yet visible in the current bag. This basically means that there are now three states, say $\{1, 0, \hat{0}\}$, where the assignment of 0 means that the vertex is already dominated at the current stage of the algorithm, and $\hat{0}$ means that, at the current stage of the algorithm, we still ask for a domination of this vertex.[2] We ask the reader to fill in the details for the four different situations of a treewidth-based algorithm.

Let us only point to the following additional complication when dealing with join nodes: if we update an assignment that maps vertex x onto 0, it is not necessary that both children assign 0 to x; it is sufficient that one of the two branches does, while the other assigns $\hat{0}$. So, while it is clear that in the initialization phase and when processing insert or forget nodes, $\mathcal{O}(3^{\text{tw}(G)})$ time is needed to perform the other node actions, a direct implementation of what we said in the previous sentence would amount in spending $\mathcal{O}(9^{\text{tw}(G)})$ time for the join node processing. How this could be sped up to $\mathcal{O}(4^{\text{tw}(G)})$ is explained in [8]. However, observe that in any case here a small path decomposition offers an edge over a tree decomposition of the same size, since then no join nodes would have to be processed.

Let us explain the mentioned $\mathcal{O}(4^{\text{tw}(G)})$ complexity for the join nodes a bit more in detail, deviating in our explanation from [8] in some details. The basic trick can be boiled down to the reinterpretation of the assignment of $\hat{0}$ to vertex x: we assume that we don't know if x has already been dominated at the current stage of the algorithm. This means that, whenever we have to assignments α and α' such that we can obtain α from α' by replacing some 0 by $\hat{0}$, then the table value (this time denoting the number of vertices that have been put into the dominating set at this stage of the algorithm) of α is greater than or equal to the table value of α'. Therefore, we only need to distinguish two cases whenever we have a $\hat{0}$ assignment for some vertex x in the parent node: either the left child assigns $\hat{0}$ to x and the right child assigns 0 to x, or left child assigns 0 to x and the right child assigns $\hat{0}$ to x. Hence, we can compute the number of all necessary checks for assignments to

[2]This necessity of two states for a vertex not being in the dominating set has been overlooked in [318], so that the running times that are claimed in that paper for Roman domination are better than ours as explained below.

k vertices (in the parent node) by the following recursion, where we assume to have chosen a specific vertex x:

- either 1 or 0 is assigned to x; then, the same assignment must have been made in the two children;
- or $\hat{0}$ is assigned to x; then, we have two possible assignments in the child nodes: 0 to x in the left child and $\hat{0}$ to x in the right child or vice versa.

In each case, we can now assume we have $k-1$ vertices to which values must be assigned. All in all, for the number $T(k)$ of situations we have to check to compute the values in a parent node with k vertices, we have obtained the recursion $T(k) = 4T(k-1)$, where naturally $T(0) = 1$ ends the recursion. Hence, $T(k) = 4^k$ as claimed.

Theorem 7.12 MINIMUM DOMINATING SET*, parameterized by the treewidth* $\mathrm{tw}(G)$ *of the input graph* G*, can be solved in time* $\mathcal{O}(4^{\mathrm{tw}(G)}|V(G)|)$*.*

A graph-structural reason for DOMINATING SET being more difficult than VERTEX COVER[3] can be found in the fact that, whenever a graph G can be covered by k vertices, then any subgraph can be covered by at most k vertices; yet, there may be subgraphs of a graph G that can be dominated by k vertices that need more vertices for domination; such phenomena were recently studied in detail in [229].

In the case of ROMAN DOMINATION, a similar problem occurs within the join nodes. Moreover, notice that we now have two different states for a vertex that is in the dominating set, depending on whether it hosts one or two armies. Again, we leave details to the reader; however, the naive $\mathcal{O}(16^{\mathrm{tw}(G)})$ algorithm should be in principle clear from what we said. Notice again that this algorithm can be substantially improved by the techniques we just mentioned. Namely, assume that we assign one out of $\{0, \hat{0}, 1, 2\}$ to each vertex in a bag. The semantics of 0 an $\hat{0}$ is not changed. The assignments 1 and 2 count the number of armies put on the corresponding vertex. Observe that a vertex can be only dominated by a neighbor vertex to which we assigned a 2. Using the same tricks as explained in the previous case, we get the following recursion for the number $T(k)$ of situations we have to check to compute the values in a parent node with k vertices:

[3]an observation that cannot be only seen in the more complex dynamic programming for the treewidth-bounded case, but also in the fact that on general graphs DOMINATING SET is parameterized intractable, yet VERTEX COVER is in $\mathcal{FPT}$, VERTEX COVER enjoys a smaller kernel than PLANAR DOMINATING SET etc.

- We get three $T(k-1)$-branches for assigning $0, 1, 2$ to the parent node, since the child nodes must assign the same values to the chosen vertex.
- We get two more $T(k-1)$-branches for assigning $\hat{0}$ to the parent node.

All in all, we have $T(k) = 5^k$. Since the other node updates can be even done in time $\mathcal{O}(4^k)$, we can conclude:

Theorem 7.13 MINIMUM ROMAN DOMINATION, *parameterized by the treewidth* $\mathrm{tw}(G)$ *of the input graph* G*, can be solved in time* $\mathcal{O}(5^{\mathrm{tw}(G)}|V(G)|)$*.*

Finally, we turn towards the *Queen Domination Problem* that is defined as follows:

Problem name: DOMINATING SET OF QUEENS (QDS)
Given: An $n \times n$ chessboard C
Parameter: a positive integer k
Output: Is it possible to place k queens on C such that all squares are dominated ?

However, we do not start, at this point, from the Queen chessboard graph, but rather with the $n \times n$ planar grid with diagonals that corresponds to a chessboard in a straightforward interpretation. As mentioned, this graph has pathwidth $n+1$ (see Ex. 7.6). For our dynamic programming algorithm (solving the problem in time $\mathcal{O}(c^n)$ for a c to be determined), we endow each vertex of this graph (i.e., each square on the chessboard) with the following information: for each of the at most four directions (neighbors) of that a specific square might still "influence" in the future of the algorithm (namely, if we take the way of passing through a grid that was indicated in Ex. 7.6, then these possible influence directions are: the upper right, right, lower right and lower directions), we keep track if the present square is dominated (and will hence enable domination of squares in the indicated direction, including the possibility that the square is hosting a queen) or if it still has to be dominated (and if so, from which direction). Excluding the possibility that a vertex (square) is not dominated at all, this leads to $2^4 - 1 = 15$ different "states" of a square. Notice again that we are now talking about a path decomposition based algorithm, i.e., there are no join nodes that need to be processed. Hence, we can conclude:

Theorem 7.14 MINIMUM DOMINATING SET OF QUEENS*, parameterized by the treewidth* $\mathrm{tw}(G)$ *of the input graph* G*, can be solved in time* $\mathcal{O}(15^{\mathrm{tw}(G)}|V(G)|) = \mathcal{O}(15^n n^2)$*, given an* $n \times n$ *chessboard.*

Note that this also directly leads to a solution to the original parameterized problem. We have already seen in Theorem 4.27 that QDS has a small kernel. Of course, this shows that DOMINATING SET OF QUEENS is in $\mathcal{FPT}$. More precisely, we would have a direct $\mathcal{O}(2^{(2k+1)^2} + n)$ algorithm for DOMINATING SET OF QUEENS by testing all placements of queens on the $(2k+1) \times (2k+1)$ chessboard. We could do better along the lines of our previous reasoning, however:

Theorem 7.15 DOMINATING SET OF QUEENS *can be solved in time* $\mathcal{O}(15^{2k} + n)$.

Proof. First, apply the reduction rule 25, see Theorem 4.27. Then, $n \leq 2k+1$. Hence, the dynamic programming algorithm we sketched above can be assessed to run in the claimed time. ∎

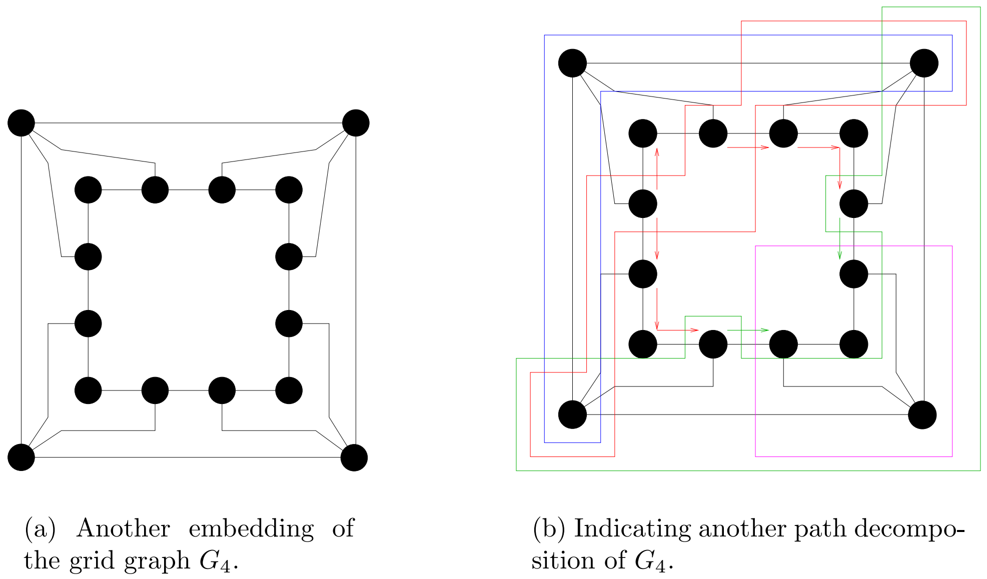
Having seen the previous examples, it might appear that all graph problems, at least those graph problems that are easy to solve on trees, should be easily solvable on graphs with a bounded treewidth. In fact, finding efficient tree algorithms say for DOMINATING SET often predates their solution on graphs of bounded treewidth, see [101, 303] for algorithms solving MINIMUM DOMINATING SET on trees in linear time. Let us remind, however, that there are some problems that, albeit efficiently solvable on trees, it is unknown if there are efficient treewidth-based algorithms. One particular example is the LINEAR ARRANGEMENT problem, see [5, 99, 352] for efficient algorithms on trees, and the discussion in [239, 240] for the obvious problems of finding efficient treewidth-based algorithms for this problem.

7.3 Planar graphs and their structure

Let us first improve our understanding of planar graphs. First, observe that there can be many ways in which a graph can be embedded into the plane.

Example 7.16 Let us continue with Example 7.5. More precisely, let us have another look at the grid graph G_4 that has been drawn as a planar graph already in Fig. 7.1(a). Namely, Fig. 7.6(a) shows another way of embedding G_4: it is basically turned inside-out.

Interestingly, different embeddings, i.e., finally different drawings, do have their bearing at the way we see graphs. For example, recall the strategy with which we constructed a tree decomposition of G_4: we took the leftmost "column" plus the "next" vertex to the upmost right into the first bag, so that we could forget about the leftmost top vertex in the next bag, and so forth. If we apply the same strategy to this alternative embedding, we might


(a) Another embedding of the grid graph G_4.

(b) Indicating another path decomposition of G_4.

Figure 7.6: Grids: graphs with pathwidth $\mathcal{O}(\sqrt{|V|})$.

get another tree decomposition of G_4, as indicated in Fig. 7.6(b). In that picture, the first bag is colored blue, then comes the red one; this is gradually changed into the green bag, as indicated by the red arrows. Further changes let us finally arrive at the pink bag. Observe that the biggest bag is the green one, and this determines the width of the tree decomposition, which is four (as above). In contrast to the previously described decomposition, this decomposition appears to be fairly asymmetric, in fact, along the lines of the sketch, a decomposition is obtainable such that only one of the bags (namely, the green one) contains five vertices.

Planar graphs are interesting in our context, since most $\mathcal{NP}$-hard graph problems remain $\mathcal{NP}$-hard when restricted to planar graphs (a notable exception being MAXIMUM CUT, see [223]). However, from a parameterized perspective, one can sometimes observe that problems that are W[1]-hard for general graphs fall into $\mathcal{FPT}$ when restricted to planar graphs; examples being INDEPENDENT SET and DOMINATING SET, as well as their variants. In fact, the derivation of the according $\mathcal{FPT}$ algorithms is somewhat the core of this chapter. In a certain sense, these problems are even easier than say VERTEX COVER on general graphs, since they allow for algorithms with a running time of $\mathcal{O}^*(c^{\sqrt{k}})$, which is improbable for problems on general graphs, see [69].

7.3.1 Outerplanarity: the definition

Definition 7.17 A crossing-free embedding of a graph G in the plane is called *outerplanar embedding* if each vertex lies on the boundary of the outer face. A graph G is called an *outerplanar graph* if it admits an outerplanar embedding in the plane.

The following generalization of the notion of outerplanarity was introduced by Baker [32].

Definition 7.18 A crossing-free embedding of a graph G in the plane is called *r-outerplanar* if:

- for $r = 1$, the embedding is outerplanar, and,
- for $r > 1$—inductively—when removing all vertices on the boundary of the outer face and their incident edges, the embedding of the remaining subgraph is $(r-1)$-outerplanar.

A graph G is called *r-outerplanar* if it admits an r-outerplanar embedding. The smallest number r, such that G is r-outerplanar is called the *outerplanarity* (number).

In this way, we may speak of the layers $L_1, \ldots, L_r$ of an embedding of an r-outerplanar graph:

Definition 7.19 For a given r-outerplanar embedding of a graph $G = (V, E)$, we define the ith *layer* L_i inductively as follows. Layer L_1 consists of the vertices on the boundary of the outer face, and, for $i > 1$, the layer L_i is the set of vertices that lie on the boundary of the outer face in the embedding of the subgraph $G - (L_1 \cup \ldots \cup L_{i-1})$.

Let us look at grid graphs again to see the general flavor of these notions.

Example 7.20 What about our favorite graph, the grid graph G_4, in terms of outerplanarity? Starting with the embedding from Fig. 7.1, Fig. 7.7 shows the two layers of G_4 by coloring them red and blue. It is not too hard to see in this case that there is no embedding that might display smaller outerplanarity.

In general terms, the grid graph G_n has outerplanarity $\lceil n/2 \rceil$. Hence, the outerplanarity grows linearly with the square root of the number of vertices of G_n.

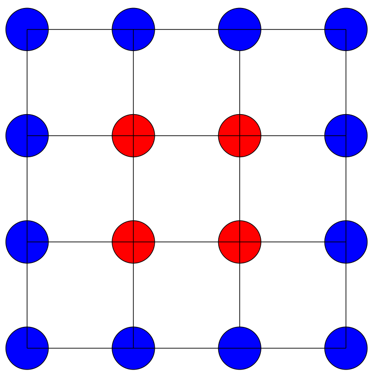


Figure 7.7: An example graph of outerplanarity two.

To understand the techniques used in the following sections, it is helpful to consider the concept of a *layer decomposition* of an r-outerplanar embedding of graph G. A layer decomposition of an r-outerplanar embedding of graph G is a forest of height $r-1$. The nodes of the forest correspond to different connected components of the subgraphs of G induced by a layer. For each layer with vertex set L_i, suppose the connected components of the subgraph of G induced by L_i have vertex sets $C_{i,1}, \ldots, C_{i,\ell_i}$, i.e., $L_i = \bigcup_{j=1}^{\ell_i} C_{i,j}$. Then, we have ℓ_i nodes that represent the nodes of layer L_i, one for each such connected component. Such a set $C_{i,j}$ will be called a *layer component.* Each layer component node $C_{1,j}$ will be the root of a tree in the forest, so we will have one tree per connected component of G, representing the exterior vertices of the component. Two layer component nodes will be adjacent if they contain vertices that share a common face. This means that a layer component node $C_{i,j}$ can only be adjacent to layer component nodes of the form $C_{i-1,j'}$ or $C_{i+1,j''}$; if $C_{i,j}$ is adjacent to $C_{i-1,j'}$, then the vertices of $C_{i,j}$ lie within the area formed by the subgraph induced by $C_{i-1,j'}$. Note that the layer component nodes on the i'th level of the forest correspond to the layer components of the form $C_{i,j}$. One easily observes that the planarity of G implies that the layer decomposition must indeed be a forest.

We need some further notation for the construction in the next sections.

A layer component $C_{i,j}$ of layer L_i is called a *non-vacuous layer* if there are vertices from layer L_{i+1} in the interior of $C_{i,j}$ (i.e., in the region enclosed by the subgraph induced by $C_{i,j}$). In other words, L_i is non-vacuous iff the corresponding component node in the layer decomposition has a child.

Lemma 7.21 *Let $\emptyset \neq C \subseteq C_{i,j}$ be a subset of a non-vacuous layer component $C_{i,j}$ of layer i, where $i \geq 2$. Then, there exists a unique smallest (in* *number of vertices) cycle $B(C)$ in layer L_{i-1}, such that C is contained in the region enclosed by $B(C)$. No other vertex of layer L_{i-1} is contained in this region.*

Definition 7.22 For each non-empty subset C of a non-vacuous layer component of layer i ($i \geq 2$), the set $B(C)$ as given in Lemma 7.21 is called the *boundary cycle* of C (which we will also call *ring* of C.

A *ring decomposition* of an r-outerplanar graph G is a directed forest $\mathcal{F}_G$ of height $r-1$ which is described as follows: the nodes of the trees are rings of vertices of G and the different trees (may) correspond to different components of G. If B' and B'' are boundary cycles, then B'' is a child of B' iff $B(x) = B'$ for any $x \in B''$. The ith layer of $\mathcal{F}_G$ consists of c_i boundary cycles $B(i,1), \ldots, B(i,c_i)$ from the ith layer L_i of G, and some *interconnection pattern* $J_i = L_i \setminus B_i$ with $B_i = \bigcup_{j=1}^{c_i} B(i,j)$.

Since different components of a graph can be processed separately in the algorithms presented below, we can assume that $\mathcal{F}_G$ is in fact a (directed) tree. Notationally, we stress this by writing $\mathcal{T}_G$ instead of $\mathcal{F}_G$. If $\mathcal{T}_G$ happens to be a path, we will call G *simple r-outerplanar.* The tree $\mathcal{F}'_G$ is obtained from $\mathcal{F}_G$ by attaching a child to any leaf $B(i,j)$ of $\mathcal{F}_G$.

With respect to the "history" of the papers that deal with hard problems on planar graphs, first the notion of layer decomposition was used, but that was later (more precisely, in [179]) replaced by the obviously related notion of a ring decomposition, since that one allowed arguments of a more geometric flavor.

Let us finally stress the fact that a ring decomposition may be different from the so-called layer decomposition introduced in [8]; e.g., two rings may share an edge, see Fig. 7.8.

The reader who wishes a concrete visualization of ring structures might wish to have a look at Fig. 7.9. The idea given by onion rings is very well reflecting the ring structure of a planar graph. We will return to this issue later on.

7.3.2 Outerplanarity versus treewidth

We provide now a *constructive* proof of how to upperbound the treewidth of a planar graph by its outerplanarity. The proof is based upon the proof in [54, Theorem 83]. To be more precise, we show:

Theorem 7.23 *Let an r-outerplanar graph $G = (V,E)$ be given together with an r-outerplanar embedding. A tree decomposition $\langle\{X_i \mid i \in I\}, T\rangle$, of width at most $3r-1$ and with $|I| = \mathcal{O}(n)$ of G, can be found in $\mathcal{O}(rn)$ time.*

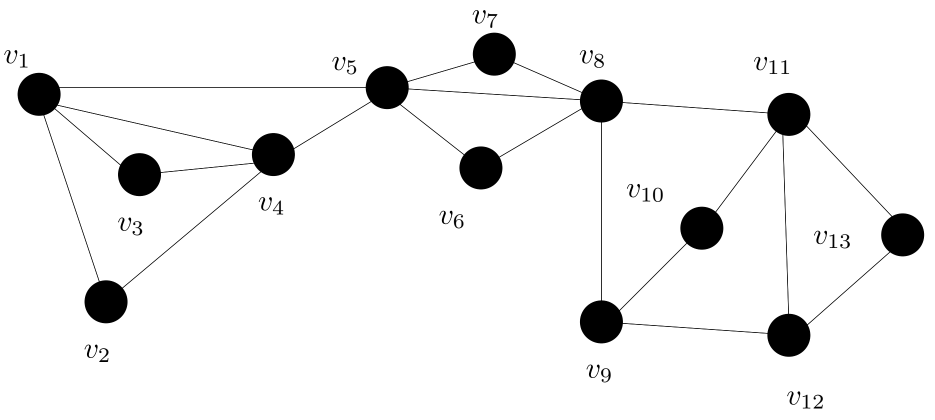


Figure 7.8: $\mathcal{F}_G$ has only two non-connected nodes, corresponding to $B(v_3) = (v_1, v_2, v_4)$ and $B(v_{10}) = (v_8, v_9, v_{12}, v_{11})$. Note that $J_1 = \{v_5, v_6, v_7, v_{13}\}$ and $J_2 = L_2 \setminus B_2 = \{v_3, v_{10}\}$. If there was another edge between v_2 and v_9, then $B(v_3) = B(v_4) = B(v_6) = (v_1, v_2, v_9, v_8, v_5, v_1)$, and hence $B(v_3)$ and $B(v_{10})$ would share the edge $\{v_8, v_9\}$.

Figure 7.9: Visualizing the ring structure of a planar graph

We make the assumption that with the embedding, for each vertex v, the incident edges of v are given in clockwise order as they appear in the embedding. Most (linear time) graph planarity testing and embedding algorithms automatically yield such orderings of the edge lists (see [37]).

Now, we discuss the proof of Theorem 7.23. For the construction of the desired tree decomposition we proceed in several steps, see Alg. 64.

Algorithm 64 Construct a tree decomposition for an r-outerplanar graph

Input(s): an r-outerplanar graph G

Output(s): a tree decomposition of G of width at most $3k-1$

```
1: Determine the layers of the given graph $G$.
2: Embed $G$ into a graph $H$, whose degree is bounded by 3 and whose
   outerplanarity number doesn't exceed the one of $G$.
3: Construct a suitable maximal spanning forest for $H$.
   This step is done inductively proceeding along the different layers of $H$ in a
   way that the so-called "edge and vertex remember numbers" are kept small.
4: Using this spanning tree, determine a tree decomposition of $H$.
5: Turn this tree decomposition into a tree decomposition of $G$.
```

The details of the steps in Alg. 64 are discussed in the following.

Determining the layers of the embedded graph. Without loss of generality, we assume G is connected. Suppose G is given with an r-outerplanar embedding with, for every vertex, a clockwise ordering of its adjacent edges. With the help of these orderings, one can build, in $\mathcal{O}(|V|)$ time, the *dual graph* G^*,[4] with pointers from edges of G to the two adjacent faces in the dual. We can first partition the set of faces into 'layers': put a face f in layer L^*_{i+1} if the distance of this face in the dual graph to the exterior face is i. This distance can be determined in linear time using breadth-first search on the dual graph.

Now, a vertex v of G belongs to layer L_i for the smallest i such that v is adjacent to a face f_v in L^*_i. Note that faces can belong to layer L^*_{r+1}, but not to layers L^*_s with $s > r+1$.

Embedding G in a graph H of degree three. The next step is to construct an r'-outerplanar (where $r' \le r$) graph $H = (V_H, E_H)$ of degree at most three that contains G as a *minor*, i.e., G can be obtained from H by a series of vertex deletions, edge deletions, and edge contractions.

[4]Given a plane graph G, its dual G^* is formed by considering the faces of G as the vertices of G^* and by putting an edge between two "face-vertices" iff there is a common boundary edge of the two faces in G. Please note that G^* depends on the given embedding of G; moreover, G^* is often seen as a multigraph (e.g., loops may occur if there are edges in G that lead to leaf vertices within a face; double edges may be due to faces that are neighbored via two different edges). For our purpose, the simple graph that corresponds to such a multigraph is sufficient.

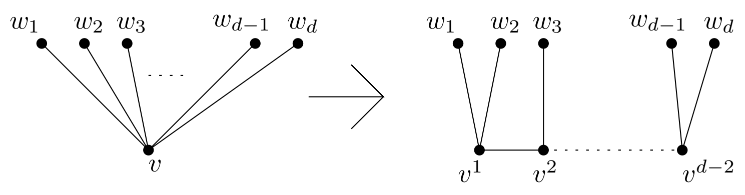


Figure 7.10: Replacing a vertex by a path with vertices of degree 3.

In order to do this, every vertex with degree $d \geq 4$ is replaced by a path of $d - 2$ vertices of degree 3, as shown in Fig. 7.10. This is done in such a way that the graph stays r'-outerplanar. More precisely, for each vertex v of layer L_i we determine the face f_v in L_i^* as described above. We then find two successive edges $\{x, v\}$, $\{v, y\}$ that are border to f_v. Now, let x take the role of w_1 and y take the role of w_d in Fig. 7.10. Observe that, in this manner, all vertices v^i on the newly formed path are adjacent to face $f_v \in L_i^*$. Let H be the graph obtained after replacing all vertices of degree at least four in this manner.

Note that H has the same set of faces as G (i.e., H^* and G^* share the same set of vertices) and that faces adjacent in G are still adjacent in H (i.e., G^* is a subgraph of H^*). Hence, H^* can be obtained from G^* by possibly adding some further edges. Clearly, the minimum distance of a vertex in H^* to the exterior face vertex may only decrease (not increase) compared to the corresponding distance in G^*. Since the layer to which a vertex belongs is exactly one more than the minimum distance of the adjacent faces to the exterior face, the outerplanarity number r' of H is bounded by r.

Constructing a suitable maximal spanning forest for H. At this point, we have an r'-outerplanar graph $H = (V_H, E_H)$ of maximum degree 3. We now construct a maximal spanning forest T for H that yields small so-called "edge and vertex remember numbers." This step is done inductively along the different layers proceeding from inside towards the exterior.

Observe that, when removing all *edges* on the exterior face of an s-outerplanar graph of maximum degree three, we obtain an $(s - 1)$-outerplanar graph when $s > 1$; we obtain a forest when $s = 1$.

Thus, we can partition the edges into $r' + 1$ sets $E_1, \ldots, E_{r'+1}$, with E_1 the edges on the exterior face, and E_i the edges on the exterior face when all edges in $E_1 \cup \ldots \cup E_{i-1}$ are removed. Again, using the dual, this partition can be computed in $\mathcal{O}(n)$ time.

Now, we form a sequence of forests. We start with forest $T_{r'+1}$, which consists of all edges in $E_{r'+1}$. (Note that these are the interior edges of an outerplanar graph of maximum degree 3, so $T_{r'+1}$ is acyclic.)

When we have T_i, $1 < i \leq r' + 1$, we form T_{i-1} in the following way: add a maximal set of edges from E_{i-1} to T_i such that no cycles are formed. Note that in this way, each T_i is a maximal spanning forest of the subgraph formed by the edges in $E_i \cup \ldots \cup E_{r'+1}$; we call this subgraph H_i. As usual, maximality is meant with respect to set inclusion; a maximal spanning forest of an arbitrary graph hence can be found in $\mathcal{O}(n)$ time using a standard depth first search approach, but here we need such a forest constructed in a specific way.

It is not hard to see that one such step can be done in $\mathcal{O}(n)$ time; as we do at most r such steps, the time to build T_1 becomes $\mathcal{O}(rn)$.

Definition 7.24 For a graph $G = (V, E)$, and a forest $T = (V, F)$ that is a subgraph of G, define the *edge remember number* $\text{er}(G, T, e)$ of an edge $e \in F$ (with respect to G and T) as the number of edges $\{v, w\} \in E \setminus F$ such that there is a simple path in T from v to w that uses e. The edge remember number of T (with respect to G) is $\text{er}(G, T) = \max_{e \in F} \text{er}(G, T, e)$.

The *vertex remember number* $\text{vr}(G, T, v)$ of a vertex $v \in V$ (with respect to G and T) is the number of edges $\{v, w\} \in E \setminus F$ such that there is a simple path in T from v to w that uses v. The vertex remember number of T (with respect to G) is $\text{vr}(G, T) = \max_{v \in V} \text{vr}(G, T, v)$.

One may observe that the construction of the trees T_i is the one used in the proofs given in [54, Section 13]. The following result is proved in [54, Lemma 80]. Note that in order to obtain this result it is essential that the degree of H is bounded by 3:

Lemma 7.25 *(i) For every i, $1 \leq i \leq r' + 1$, $\text{er}(H_i, T_i) \leq 2(r' + 1 - i)$.*
(ii) For every i, $1 \leq i \leq r'$, $\text{vr}(H_i, T_i) \leq 3(r' + 1 - i) - 1$. ∎

Without loss of generality, we can suppose that G and, therefore, H is connected and, hence, we have a spanning tree T_1 of H with $\text{er}(H, T_1) \leq 2r'$ and $\text{vr}(H, T_1) \leq 3r' - 1$.

Deriving a tree decomposition from the spanning forest. We now apply the following result of [54, Theorem 71] to the graph H and the spanning forest T_1 in order to obtain a tree decomposition in time $\mathcal{O}(rn)$ of width bounded by

$$\max(\text{vr}(H, T_1), \text{er}(H, T_1) + 1) \leq 3r' - 1 \leq 3r - 1.$$

For the sake of completeness of the whole construction, we outline the easy proof.

Theorem 7.26 *Let $T = (V, F)$ be a maximal spanning forest for the graph $G = (V, E)$. Then, a tree decomposition with width smaller than or equal to* $\max\{\mathrm{vr}(G,T), \mathrm{er}(G,T) + 1\}$ *and with $\mathcal{O}(n)$ nodes can be determined in $\mathcal{O}(\mathrm{vr}(G,T) \cdot n)$ time.*

Proof. Our aim is to construct a tree decomposition $\langle \{X_i \mid i \in I\}, T' \rangle$ of G. Let $T' = (V \cup F, F')$ with

$$F' = \{\{v, e\} \mid v \in V, e \in F, \exists w \in V : e = \{v, w\}\}$$

be the tree obtained by subdividing every edge of T. The bags X_i for $i \in I := V \cup F$ are obtained as follows. For every $v \in V$, add v to X_v. For every $e = \{v, w\} \in F$, add v and w to X_e. Now, for every edge $e = \{v, w\}$ in E but not in F, add v to all sets X_u and X_e, with $u \in V$ or $e \in F$ on the path from v to w in T.

Using standard graph algorithmic techniques, the path between two vertices in a tree can be found in time proportional to the length of that path; since each vertex in T can contribute to at most $\mathrm{vr}(G,T)$ such paths, the running time is bounded by $\mathcal{O}(\mathrm{vr}(G,T) \cdot n)$.

It is easy to check that this yields a tree decomposition. Its bags have size $|X_v| \leq 1 + \mathrm{vr}(G,T)$ (for all $v \in V$) and $|X_e| \leq 2 + \mathrm{er}(G,T)$ (for all $e \in E$). Hence, the resulting treewidth is at most $\max(\mathrm{vr}(G,T), \mathrm{er}(G,T) + 1)$. ∎

Undoing the minor operations. Finally, the tree decomposition of H can be turned into a tree decomposition of G of equal or smaller width by replacing every occurrence of a vertex v^i in a bag X_i by an occurrence of the corresponding vertex v (see e.g., [54, Lemma 16].) This again costs time linear in the size of all bags X_i in the tree decomposition, i.e., $\mathcal{O}(rn)$ time.

Altogether this establishes the correctness of Theorem 7.23.

7.3.3 Outerplanarity and other graph parameters

The concept of outerplanarity is quite useful for showing that particular problems are in $\mathcal{FPT}$. Let us clarify this with a number of examples. They comprise rather folklore results.

Proposition 7.27 *A plane graph with a dominating set of size of at most k is r-outerplanar with $r \leq 3k$.*

Proof. By definition of outerplanarity, each vertex in a dominating set can only dominate vertices from at most three layers, so that there cannot be more than $3k$ layers in any YES-instance of PLANAR DOMINATING SET. ∎

From what we have seen in the preceding subsection, we may deduce:

Corollary 7.28 *If a planar graph $G = (V, E)$ has a k-dominating set, then its treewidth is bounded by $9k - 1$.* ∎

We will give a considerably stronger bound later.

Corollary 7.29 PLANAR DOMINATING SET *is in $\mathcal{FPT}$.*

Let us remark that, from an algorithmic point of view, the following observation might be even more useful, although it delivers the same worst-case bound in the case of simple r-outerplanar graphs.

Lemma 7.30 *A plane graph G with a dominating set of size of at most k has a ring decomposition $\mathcal{F}_G$ with a dominating set of size of at most k.*

The proof for this result can be taken as a blueprint for similar statements below. It is explicitly contained in Alg. 65.

Algorithm 65 An $\mathcal{FPT}$ algorithm for PLANAR DOMINATING SET

Input(s): a planar graph G, a positive integer k
Output(s): YES if G has a dominating set of size at most k and NO, otherwise

```
Find an arbitrary planar embedding of $G$.
Determine the outerplanarity $out$ of that embedding.
if $out > 3k$ then
  return NO
else
  Determine minimum dominating set $D$ of $G$ with the help of an
  outerplanarity-based algorithm.
  return $|D| \leq k$
end if
```

For specific types of domination problems, the bounds can be better:

Proposition 7.31 *A plane graph with a connected dominating set of size of at most k is r-outerplanar with $r \leq k + 2$.*

Corollary 7.32 PLANAR CONNECTED DOMINATING SET *is in $\mathcal{FPT}$.*

Similarly, one can show:

Proposition 7.33 *A plane graph with a maximal matching of size of at most k is r-outerplanar with $r \leq 2k$.*

Proof. Consider two subsequent layers L_1, L_2 in one connected component of a plane graph. If none of the edges *within* the layer (i.e., both endpoints are within either L_1 or L_2) are in the matching, then at least one of the edges interconnecting the layers (i.e., one endpoint is in L_1 and the other one is in L_2) is in the matching by maximality. ∎

Therefore, we get an $\mathcal{FPT}$-classification of the determination of a *minimum maximal matching* (standard parameterization) in a plane graph.

In the following, we are aiming at improved relations between graph parameters (like domination number) and treewidth in the case of planar graphs.

7.4 Domination number versus treewidth

In this section, we explain in details the strategy used in [8] to obtain an $\mathcal{O}^*(c^{\sqrt{k}})$ algorithm for PLANAR DOMINATING SET. In particular, we show that a planar graph with domination number k has treewidth of at most $\mathcal{O}(\sqrt{k})$. This improves Corollary 7.28 considerably. In the next section, we will show how this new result can be turned into a constructive algorithm. Combining the results of this section with the treewidth-based algorithms we saw above, we can present an algorithm having time complexity $4^{\mathcal{O}(\sqrt{k})}n$. This obviously gives an asymptotic improvement of the $\mathcal{O}(8^k n)$ search tree algorithm, see Chap. 5.

This section is organized as follows. In a first subsection, we make some general observations on how to construct tree decompositions using separators. The following two subsections show that, in a planar graph which admits a k-dominating set, we can find small separators layerwisely. Finally, the results are pieced together to prove our main result in this section.

7.4.1 Separators and treewidth

Here, the main idea is to find small separators of the graph and to merge the tree decompositions of the resulting subgraphs.

To simplify notation, we write $A \uplus B$ for the disjoint union of two sets A and B. Graph separators are defined as follows.

Definition 7.34 Let $G = (V, E)$ be an undirected graph. A *separator* $S \subseteq V$ of G divides V into two *parts* (or *chunks*) $A_1 \subseteq V$ and $A_2 \subseteq V$ such that[5]

[5]In general, of course, A_1, A_2 and S will be non-empty. In order to cover boundary cases in some considerations below, we did not put this into the separator definition.

- $A_1 \uplus S \uplus A_2 = V$, and
- no edge joins vertices in A_1 and A_2.

Later, we will write δA_1 (or δA_2) as shorthand for $A_1 \uplus S$ (or $A_2 \uplus S$, respectively). The triple (A_1, S, A_2) is also called a *separation* of G.

Clearly, this definition can be generalized to the case where a separator partitions the vertex set into ℓ subsets instead of only two. We refer to such separators simply by *ℓ-separator*; $\ell = 2$ then is the separator in Definition 7.34.

For any given separator splitting a graph into different components, we obtain a simple upper bound for the treewidth of this graph which depends on the size of the separator and the treewidth of the resulting components.

Proposition 7.35 *If a connected graph can be decomposed into components of treewidth of at most t by means of a separator of size s, then the whole graph has treewidth of at most $t + s$.*

Proof. The separator splits the graph into different components. Suppose we are given the tree decompositions of these components of width at most t. The goal is to construct a tree decomposition for the original graph. This can be achieved by firstly merging the separator to every bag in each of these given tree decompositions. In a second step, add some arbitrary connections preserving acyclicity between the trees corresponding to the components. It is straightforward to check that this forms a tree decomposition of the whole graph of width at most $t + s$. ∎

For plane graphs, there is an iterated version of this observation can be similarly shown.

Proposition 7.36 *Let G be a plane graph with layers L_i, $(i = 1, \dots, r)$. For $i = 1, \dots, \ell$, let $\mathcal{L}_i$ be a set of consecutive layers, i.e.,*

$$\mathcal{L}_i = \{L_{j_i}, L_{j_i+1}, \dots, L_{j_i+n_i}\},$$

such that $\mathcal{L}_i \cap \mathcal{L}_{i'} = \emptyset$ for all $i \neq i'$. Moreover, suppose G can be decomposed into components, each of treewidth of at most t, by means of separators $S_1, \dots, S_\ell$, where $S_i \subseteq \bigcup_{L \in \mathcal{L}_i} L$ for all $i = 1, \dots, \ell$. Then, G has treewidth of at most $t + 2s$, where $s = \max_{i=1,\dots,\ell} |S_i|$.

7.4.2 Finding separators layerwisely

In the following, we assume that our graph G has a fixed plane embedding with r layers. We show that the treewidth cannot exceed $\mathcal{O}(\sqrt{k})$ if a dominating set of size k exists. This implies that if we have an optimal tree-decomposition we can find the domination number fast using standard dynamic programming techniques (as explained above).

Let the layers be $L_1, L_2, \dots, L_T$, with L_1 being the outer layer.

We assume that we have a dominating set D of size at most k. Let k_i be the number of vertices of $D_i = D \cap L_i$. Hence, $\sum_{i=1}^{r} k_i = k$. In order to avoid case analysis, we set $k_0 = k_{r+1} = k_{r+2} = 0$. Moreover, let c_i denote the number of non-vacuous layer components of layer L_i.

Our approach is based on finding small separators in G layerwisely. More precisely, for each $i = 1, \dots, r-2$, we want to construct a set $S_i \subseteq L_{i-1} \cup L_i \cup L_{i+1}$ separating layer L_{i-1} from layer L_{i+2} in such a way that the total size of these sets can be bounded by some linear term in k. The key idea for proving that S_i separates layers L_{i-1} from L_{i+2} relies on a close investigation of the paths leaving layer L_{i-1} to the interior of the graph. Each such path passes a "first" vertex in layer L_i. This particular vertex can be dominated by vertices from D_{i-1}, D_i, or D_{i+1}. It turns out that, in order to cut this particular path, the set S_i has to contain the vertices of the sets D_{i-1}, D_i, and D_{i+1} *plus* some suitably chosen pairs of neighbors of any of these vertices. This results in letting S_i be the union of so-called "upper," "lower," and "middle" triples. We will carry out the to some extent technically complicated step in what follows.

Upper triples. An *upper triple* for layer L_i is associated to a non-vacuous[6] layer component $C_{i+1,j}$ of layer L_{i+1} and a vertex $x \in D_{i-1}$ that has a neighbor on the boundary cycle $B(C_{i+1,j})$ (see Fig. 7.11). Then, clearly, $x \in B(B(C_{i+1,j}))$, by definition of a boundary cycle. Let x_1 and x_2 be the neighbors of x on the boundary cycle $B(B(C_{i+1,j}))$. Starting from x_1, we go around x up to x_2 so that we visit all neighbors of x in layer L_i. We note the neighbors of x on the boundary cycle $B(C_{i+1,j})$. Going around gives two outermost neighbors y and z on this boundary cycle. The triple, then, is the three-element set $\{x, y, z\}$. In case x has only a single neighbor y in $B(C_{i+1,j})$, the "triple" consists of only $\{x, y\}$ (let, by default, $z = y$).

Definition 7.37 For each non-vacuous layer component $C_{i+1,j}$ of L_{i+1} and

[6]Note that here, as well as in the definitions of middle and upper triples, all vacuous components of layer L_{i+1} are of no interest to us, since we want to find a set of vertices separating levels L_{i-1} from L_{i+2}.

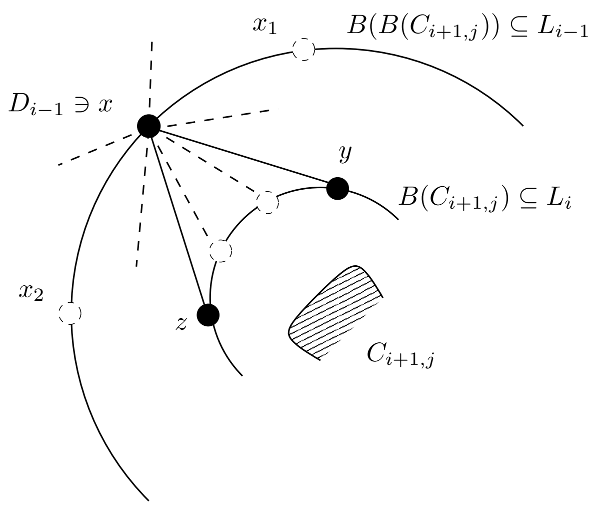

x_1
$B(B(C_{i+1,j})) \subseteq L_{i-1}$
$D_{i-1} \ni x$
y
$B(C_{i+1,j}) \subseteq L_i$
x_2
z
$C_{i+1,j}$

Figure 7.11: Upper triples.

each vertex $x \in D_{i-1}$ with neighbors in $B(C_{i+1,j})$, the set $\{x, y, z\}$ as described above is called an *upper triple* of layer L_i.

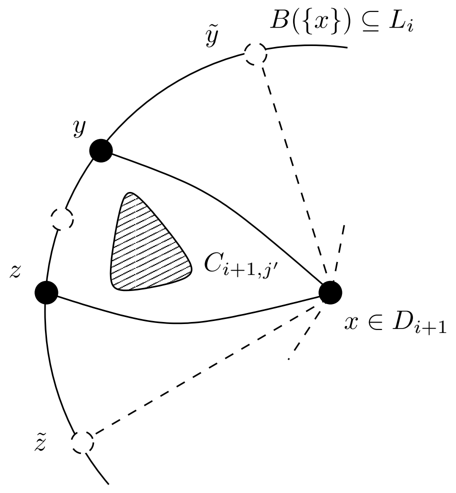


Figure 7.12: Lower triples.

Lower triples. A *lower triple* for layer L_i is associated to a vertex $x \in D_{i+1}$ and a non-vacuous layer component $C_{i+1,j}$ of layer L_{i+1} (see Fig. 7.12). We only consider layer components $C_{i+1,j}$ of layer L_{i+1} that are enclosed by the boundary cycle $B(\{x\})$. For each pair $\tilde{y}, \tilde{z} \in B(\{x\}) \cap N(x)$ (where $\tilde{y} \neq \tilde{z}$), we consider the path $P_{\tilde{y},\tilde{z}}$ from $\tilde{y}$ to $\tilde{z}$ along the cycle $B(\{x\})$, taking the direction such that the region enclosed by $\{\tilde{z}, x\}$, $\{x, \tilde{y}\}$, and $P_{\tilde{y},\tilde{z}}$ contains the layer component $C_{i+1,j}$. Let $\{y, z\} \subseteq B(\{x\}) \cap N(x)$ be the pair such that the corresponding path $P_{y,z}$ is shortest. The triple, then, is the three-element set $\{x, y, z\}$. If x has no or only a single neighbor y in $B(\{x\})$, then the "triple" consists only of $\{x\}$, or $\{x, y\}$ (by default, in these cases, we let $x = y = z$, or $y = z$, respectively).

Definition 7.38 For each vertex $x \in C_{i+1,j}$ of D_{i+1} and each non-vacuous layer component $C_{i+1,j}$ that is enclosed by $B(\{x\})$, the set $\{x, y, z\}$ as described above is called a *lower triple* of layer L_i.

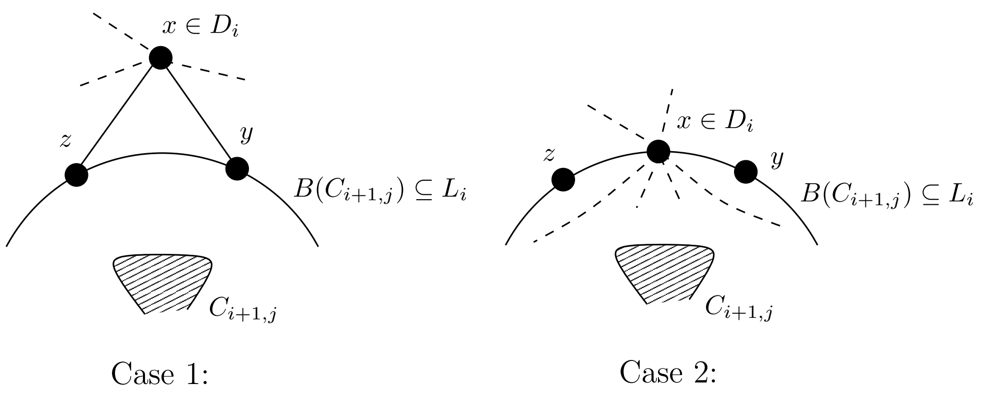


Figure 7.13: Middle triples.

Middle triples. A *middle triple* for layer L_i is associated to a non-vacuous layer component $C_{i+1,j}$ and a vertex $x \in D_i$ that has a neighbor in $B(C_{i+1,j})$ (see Fig. 7.13). Note that, due to the layer model, it is easy to see that a vertex $x \in D_i$ can have at most two neighbors y, z in $B(C_{i+1,j})$. Depending on whether x itself lies on the cycle $B(C_{i+1,j})$ or not, we obtain two different cases which are both illustrated in Fig. 7.13. In either of these cases, the middle triple is defined as the set $\{x, y, z\}$. Again, if x has none or only a single neighbor y in $B(C_{i+1,j})$, then the "triple" consists only of $\{x\}$, or $\{x, y\}$, respectively (by default, in these cases, we let $x = y = z$, or $y = z$, respectively).

Definition 7.39 For each non-vacuous layer component $C_{i+1,j}$ and each vertex $x \in D_i$, the set $\{x, y, z\}$ as described above is called a *middle triple* for layer L_i.

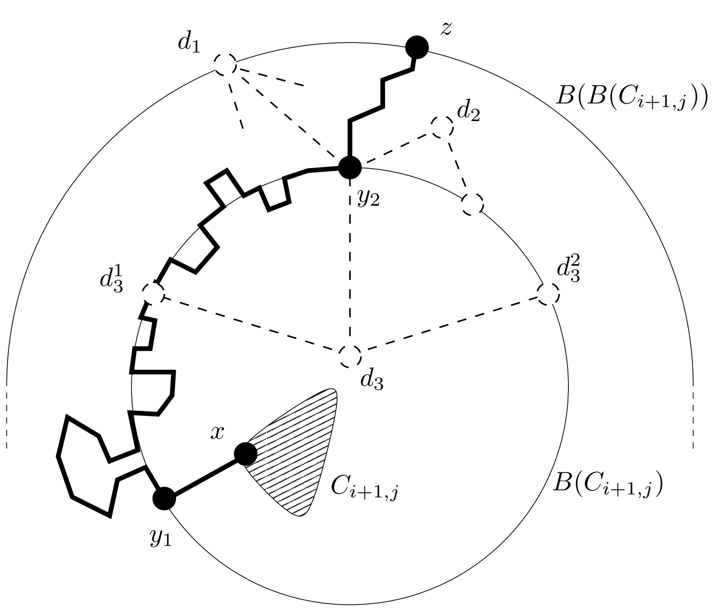


Figure 7.14: S_i separates L_{i-1} and L_{i+2}.

Definition 7.40 We define the set S_i as the union of all upper triples, lower triples and middle triples of layer L_i.

In the following, we will show that S_i is a separator of the graph. Note that the upper bounds on the size of S_i, which are derived afterwards, are crucial for the upper bound on the treewidth derived later on.

Proposition 7.41 *The set S_i separates vertices of layers L_{i-1} and L_{i+2}.*

Proof. Suppose there is a path P (with no repeated vertices) from layer L_{i+2} to layer L_{i-1} that avoids S_i. This clearly implies that there exists a path P' from a vertex x in a non-vacuous layer component $C_{i+1,j}$ of layer L_{i+1} to a vertex $z \in B(B(C_{i+1,j}))$ in layer L_{i-1} which has the following two properties:

- $P' \cap S_i = \emptyset$.
- All vertices inbetween x and z belong to layer L_i or to vacuous layer components of layer L_{i+1}.

This can be achieved by simply taking a suitable sub-path P' of P. Let y_1 (and y_2, respectively) be the first (last) vertex along the path P' from x to z that lies on the boundary cycle $B(C_{i+1,j}) \subseteq L_i$ (see Fig. 7.14).

Obviously, y_2 cannot be an element of D since, then, it would appear in a middle triple of layer L_i and, hence, in S_i. We now consider the vertex that dominates y_2. This vertex can lie in layer L_{i-1}, L_i, or L_{i+1}.

Suppose first that y_2 is dominated by a vertex $d_1 \in L_{i-1}$. Then, d_1 is in $B(B(C_{i+1,j}))$, simply by definition of the boundary cycle (see Fig. 7.14). Since G is planar, this implies that y_2 must be an "outermost" neighbor of d_1 among all elements in $N(d_1) \cap B(C_{i+1,j})$. If this were not the case, then there would be an edge from d_1 to a vertex on $B(C_{i+1,j})$ that leaves the closed region bounded by $\{d_1, y_2\}$, the path from y_2 to z, and the corresponding path from z to d_1 along $B(B(C_{i+1,j}))$. Hence, y_2 would be in the upper triple of layer L_i which is associated to the layer component $C_{i+1,j}$ and d_1. This contradicts the assumption that P' avoids S_i.

Now, suppose that y_2 is dominated by a vertex $d_2 \in D_i$ (see Fig. 7.14). By definition of the middle triples, this clearly implies that y_2 is in the middle triple associated to $C_{i+1,j}$ and d_2. Again, this contradicts the fact that $P' \cap S_i = \emptyset$.

Consequently, the dominating vertex d_3 of y_2 has to lie in layer L_{i+1}. Let $\{d_3, d_3^1, d_3^2\}$, where $d_3^1, d_3^2 \in N(d_3) \cap B(C_{i+1,j})$, be the lower triple associated to layer component $C_{i+1,j}$ and d_3 (see Fig. 7.14). By definition, $C_{i+1,j}$ is contained in the region enclosed by $\{d_3^1, d_3\}, \{d_3, d_3^2\}$, and the path from d_3^2 to d_3^1 along $B(C_{i+1,j})$, which—assuming that $y_2 \notin \{d_3, d_3^1, d_3^2\}$—does not hit y_2 (see Fig. 7.14). We now observe that, whenever the path from y_1 to y_2 leaves the cycle $B(C_{i+1,j})$ to its exterior, say at a vertex q, then it has to return to $B(C_{i+1,j})$ at a vertex $q' \in N(q) \cap B(C_{i+1,j})$. Otherwise, we would violate the fact that, by definition, $B(C_{i+1,j}) \subseteq L_i$. This, however, shows that the path P' has to hit either d_3^1 or d_3^2 on its way from y_1 to y_2. Since $d_3^1, d_3^2 \in S_i$, this case also contradicts the fact that $P' \cap S_i = \emptyset$. ∎

7.4.3 An upper bound for the size of the separators

In this subsection, we want to show that the sum of the cardinalities of all separators can be bounded by some linear term in k.

Lemma 7.42 $|S_i| \leq 5(k_{i-1} + k_i + k_{i+1}) + 12c_{i+1}$.

A proof of this assertion can be found in [8].

For the number c_i of non-vacuous components in layer i, we have the following estimate.

Lemma 7.43 $c_i \leq k_i + k_{i+1} + k_{i+2}$.

Proof. By definition, c_i refers to only non-vacuous layer components in layer L_i, i.e., there is at least one vertex of layer L_{i+1} contained within each such layer component. Such a vertex can only be dominated by a vertex from layer L_i, L_{i+1}, or L_{i+2}. In this way, we get the claimed upper bound. ∎

Combining the two previous results yields the bound claimed at the beginning of this subsection.

Proposition 7.44 $\sum_{i=1}^{r} |S_i| \leq 51k$, *where r is the number of layers of the graph.*

Proof. This follows directly when we combine the previous two lemmas, noting that $\sum_{i=1}^{r} k_i = k$. ∎

We are now ready to state the main result from [8] relating the domination number and the treewidth of a planar graph. Note that the following theorem gives a decisive asymptotic improvement of Corollary 7.28.

Theorem 7.45 *A planar graph with domination number k has treewidth of at most $6\sqrt{34}\sqrt{k} + 8$.*

Proof. Using the separators S_i as found in the previous subsections, we consider the following three sets of vertices: $\mathbb{S}_1 = S_1 \cup S_4 \cup S_7 \cup \ldots$, $\mathbb{S}_2 = S_2 \cup S_5 \cup S_8 \cup \ldots$, and $\mathbb{S}_3 = S_3 \cup S_6 \cup S_9 \cup \ldots$. Since, by Proposition 7.44, $|\mathbb{S}_1| + |\mathbb{S}_2| + |\mathbb{S}_3| \leq 51k$, one of these sets has size at most $\frac{51}{3}k$, say $\mathbb{S}_\delta$ (with $\delta \in \{1, 2, 3\}$).

Let δ and $\mathbb{S}_\delta$ be obtained as above. Let $d := \frac{3}{2}\sqrt{34}$. We now go through the sequence $S_{1+\delta}, S_{4+\delta}, S_{7+\delta}, \ldots$ and look for separators of size at most $s(k) := d\sqrt{k}$. Due to the estimate on the size of $\mathbb{S}_\delta$, such separators of size at most $s(k)$ must appear within each $n(k) := \frac{51}{3} \cdot \frac{1}{d\sqrt{k}} \cdot k = \frac{1}{3}\sqrt{34}\sqrt{k}$ sets in the sequence. In this manner, we obtain a set of disjoint separators of size at most $s(k)$ each, such that any two consecutive separators from this set are at most $3n(k)$ layers apart. Clearly, the separators chosen in this way fulfill the requirements in Proposition 7.36.

Observe that the components cut out in this way each have at most $3(n(k) + 1)$ layers and, hence, their treewidth is bounded by $9(n(k) + 1) - 1$ due to Prop. 7.27.

Using Proposition 7.36, we can compute an upper bound on the treewidth $\mathrm{tw}(k)$ of the originally given graph with domination number k:

$$\begin{aligned}
\mathrm{tw}(k) &\leq 2s(k) + 9(n(k) + 1) - 1 \\
&= 2(\frac{3}{2}\sqrt{34}\sqrt{k}) + 9(\frac{1}{3}\sqrt{34}\sqrt{k}) + 8 \\
&= 6\sqrt{34}\sqrt{k} + 8.
\end{aligned}$$

This proves our claim. ∎

How did we come to the constants? We simply computed the minimum of $2s(k) + 9(n(k) + 1) - 1$ (the upper bound on the treewidth) given the bound $s(k)n(k) \leq \frac{51}{3}k$. This suggests $s(k) = d\sqrt{k}$, and d is optimal when $2s(k) = 9n(k) = 9 \cdot \frac{51}{3} \cdot k \cdot s(k)^{-1}$, so, $2d = \frac{153}{d}$, i.e., $d = \frac{3}{2}\sqrt{34}$.

Observe that the tree structure of the tree decomposition obtained in the preceding proof corresponds to the structure of the layer decomposition forest.

Remark 7.46 *Up to constant factors, the relation exhibited in Theorem 7.45 is optimal. This can be seen, for example, by considering a complete grid graph G_ℓ of size $\ell \times \ell$, i.e., with ℓ^2 vertices. It is known that* $\mathrm{tw}(G_\ell) \geq \ell$ *(see [54, Corollary 89]) and the* domination number $\gamma(\cdot)$ *obeys* $\gamma(G_\ell) \in \Theta(\ell^2)$, *see [228, Theorem 2.39]. Therefore, there is an infinite family of planar graphs $\hat{G}_{k_i}$ with domination number k_i such that $\hat{G}_{k_i}$ has treewidth of* $\Omega(\sqrt{k_i})$.

7.4.4 A treewidth-based algorithm for PLANAR DOMINATING SET

In this section, we outline our fixed parameter algorithm for solving PLANAR DOMINATING SET constructively. The input instance to the algorithm consists of a planar graph $G = (V, E)$ and a positive integer k. The algorithm determines whether G admits a dominating set of size at most k, and, if so, constructs such a set. The running time for the algorithm will be $4^{\mathcal{O}(\sqrt{k})}n$, where $n = |V|$.

The key idea for our algorithm is to construct a tree decomposition of the stated width. However, Theorem 7.45 is a pure existence theorem which cannot be made constructive directly. There is one point that needs specific attention. As we do not start with the dominating set given, we cannot construct the upper, middle, and lower triples. Instead, we have to compute the minimum size separator $\hat{S}_i$ between L_{i-1} and L_{i+2} directly, and use that set instead of S_i as defined in the proof of Subsection 7.4.2. Such a minimum size separator can be computed with well known techniques based on maximum flow (see, e.g., [250]).

Our algorithm proceeds in the following steps:

1. Embed the planar graph $G = (V, E)$ crossing-free into the plane. Determine the outerplanarity number r of this embedding and all layers $L_1, \ldots, L_r$. By default, we set $L_i = \emptyset$ for all $i < 0$ and $i > r$.

2. If $r > 3k$ then `exit` (there exists no k-dominating set). This step of the algorithm is justified by Proposition 7.27.

3. For $\delta = 1, 2, 3$ and $i = 0, \ldots, \lfloor \frac{r}{3} \rfloor - 1$, find the minimum separator $\hat{S}_{3i+\delta}$, separating layers $L_{(3i+\delta)-1}$ and $L_{(3i+\delta)+2}$. Let $s_{3i+\delta} = |\hat{S}_{3i+\delta}|$.

4. Check whether there exists a $\delta \in \{1, 2, 3\}$ and an increasing sequence $(i_j)_{j=1,\ldots,t}$ of indices in $\{0, \ldots, \lfloor \frac{r}{3} \rfloor - 1\}$, such that

$$\begin{aligned} s_{3i_j+\delta} &\leq s(k) := \frac{3}{2}\sqrt{34}\sqrt{k} \quad \text{for all } j = 1, \ldots, t \text{ and} \\ |i_{j+1} - i_j| &\leq n(k) := \frac{1}{3}\sqrt{34}\sqrt{k} \quad \text{for all } j = 1, \ldots, t-1. \end{aligned}$$

   If the answer is "no," then `exit` (there exists no k-dominating set). This step of the algorithm is justified by the considerations in the proof of Theorem 7.45.

5. Consider the separators $\mathcal{S}_j := \hat{S}_{3i_j+\delta}$ for $j = 1, \ldots, t$ (by default, $\mathcal{S}_j = \emptyset$ for all other j) and, for each $j = 0, \ldots, t$, let G_j be the subgraph cut out by the separators $\mathcal{S}_j$ and $\mathcal{S}_{j+1}$ or, more precisely, let

$$G_j := G - \Big(\bigcup_{\ell=(3i_j+\delta)-1}^{(3i_{(j+1)}+\delta)+1} L_\ell \setminus (\mathcal{S}_j \cup \mathcal{S}_{j+1}) \Big).$$

   Note that G_j is at most $3(n(k)+1)$-outerplanar.

6. Construct tree decompositions $\mathcal{X}_j$ for G_j ($j = 0, \ldots, t$) with $\mathcal{O}(n)$ nodes each. For this step, we refer to Theorem 7.23.

7. Construct a tree decomposition $\mathcal{X}$ of G with $\mathcal{O}(n)$ nodes using the separators $\mathcal{S}_1, \ldots, \mathcal{S}_t$ and the tree decompositions $\mathcal{X}_j$ by the separator merging technique described in the proof of Proposition 7.36.

8. Solve the MINIMUM DOMINATING SET problem for G with tree decomposition $\mathcal{X}$ as described in Sec. 7.2.

Details on implementing Step 1 can be found in [8].

As to Step 3 of the algorithm, we want to remark that such a minimum size separator can be computed with well known techniques based on maximum flow (see, e.g., [250]) as follows: For given values of i and δ, we first, build the graph G', induced by the vertices in $L_{3i+\delta-1} \cup L_{3i+\delta} \cup L_{3i+\delta+1} \cup L_{3i+\delta+2}$. Then, we contract all vertices in $L_{3i+\delta-1}$ to one vertex s, and all vertices in $L_{3i+\delta+2}$ to one vertex t. We look for a minimum s-t-separator in the resulting graph but, if this separator has size more than $s(k)$, then we just note that no separator of size at most $s(k)$ exists (compare with Step 4). Finding the separator or determining that no small enough separator exists can be done with the following standard method. Build a directed graph G'' in the following way. Every vertex v in the graph is replaced by two vertices v^- and v^+, with an edge from v^- to v^+, and an edge $\{v, w\}$ is replaced by two edges (v^+, w^-), (w^+, v^-). As explained in [292, Lemma 11, page 83], the minimum s-t separator in G' can be found by applying a maximum flow algorithm in G''. To keep the running time in the stated bounds, we use the Ford-Fulkerson maximum flow algorithm, but stop as soon as a flow value of more than $s(k)$ is used since, in such a case, we can conclude that no s-t-separator of size at most $s(k)$ exists. As each flow augmentation step costs time linear in the number of vertices and edges of G'' and increases the flow value by one, the time used by this procedure is $\mathcal{O}(n' \cdot s(k))$, with $n' = |L_{3i+\delta-1} \cup L_{3i+\delta} \cup L_{3i+\delta+1} \cup L_{3i+\delta+2}|$.

As for every vertex in G, there are at most four combinations of δ and i in Step 3 such that it belongs to $L_{3i+\delta-1} \cup L_{3i+\delta} \cup L_{3i+\delta+1} \cup L_{3i+\delta+2}$ and the total time of step 3 is bounded by $\mathcal{O}(n \cdot s(k)) = \mathcal{O}(n\sqrt{k})$.

The correctness of the algorithm above follows from our considerations in the previous sections.

Steps 1-7 allow us to construct a tree decomposition of width $6\sqrt{34}\sqrt{k}+8$ of G in $\mathcal{O}(\sqrt{k}n)$ time. The running time for the last step in the algorithm is $\mathcal{O}(4^{6\sqrt{34}\sqrt{k}}n)$.

Summarizing these observations, we obtain the following theorem.

Theorem 7.47 PLANAR DOMINATING SET *can be solved in time* $\mathcal{O}(c^{\sqrt{k}}n)$, *where* $c = 4^{6\sqrt{34}}$. *Moreover, if the domination number obeys* $\gamma(G) \leq k$, *a minimum size dominating set can be constructed within the same time.* ∎

The constant c above is $4^{6\sqrt{34}}$, which is rather large. However, a more refined analysis can help reduce this constant, as we will see below.

7.5 The beauty of small kernels: separator theorems

The strategy explained in the previous section appears to be pretty special to PLANAR DOMINATING SET. Can we obtain similar results say for VERTEX

COVER on planar graphs? In this and the following section, we are going to explain the more general ideas how to exploit small kernels from [14, 15, 190]. More specifically, in this section we exhibit how to use classical separator theorems plus nice kernelization results to get parameterized algorithms with running times of the form $\mathcal{O}^*(c^{\sqrt{k}})$. The next section will use strategies inspired from the notion of outerplanarity that we showed above. This way, better running times are achievable.

7.5.1 Classical separator theorems

One of the most useful algorithmic techniques for solving computational problems is divide and conquer. In the case of planar graph problems, the famous planar separator theorem of Lipton and Tarjan [275, 276] laid the basis for the divide and conquer approach. The techniques we develop here all are based on the existence of "small" graph separators, which means that the separator size $|S|$ is bounded by $o(|V|)$. It is well-known that, for general graphs, small separators need not exist; consider, for example, the complete graph K_n, which does not possess small separators. It is of key importance for our purpose that, in particular, planar graphs do have small separators. Therefore, it is not only of pure mathematical importance to study graph classes which are guaranteed to have small generators.

Definition 7.48 According to Lipton and Tarjan [275], an $f(\cdot)$*-separator theorem (with constants* $\alpha < 1$*,* $\beta > 0$*)* for a class $\mathbb{G}$ of graphs which is closed under taking vertex-induced subgraphs is a theorem of the following form: If G is any n-vertex graph in $\mathbb{G}$, then there is a separation (A_1, S, A_2) of G such that

- neither A_1 nor A_2 contains more than αn vertices, and
- S contains no more than $\beta f(n)$ vertices.

Again, this definition easily generalizes to ℓ-separators with $\ell > 2$.

Stated in this framework, the planar separator theorem due to Lipton and Tarjan [275] is a $\sqrt{\cdot}$-separator theorem with constants $\alpha = 2/3$ and $\beta = 2\sqrt{2} \approx 2.83$. The current record for $\alpha = 2/3$ is $\beta = \sqrt{2/3}+\sqrt{4/3} \approx 1.97$ [128]. Djidjev has also shown a lower bound of $\beta \approx 1.55$ for $\alpha = 2/3$ [130]. For $\alpha = 1/2$, the "record" is $\beta = 2\sqrt{6} \approx 4.90$ [18, 54]. A lower bound of $\beta \approx 1.65$ is known in this case [353]. For $\alpha = 3/4$, the best known value for β is $\sqrt{2\pi/\sqrt{3}} \cdot (1+\sqrt{3})/\sqrt{8} \approx 1.84$ with a known lower bound of $\beta \approx 1.42$, see [353]. The results are summarized in Table 7.1.

β	$\alpha = \frac{2}{3}$		$r(\frac{2}{3}, \beta)$
upper bounds	$\sqrt{\frac{2}{3}} + \sqrt{\frac{4}{3}}$	[128]	10.74
lower bounds	1.55	[130]	8.45
	$\alpha = \frac{1}{2}$		$r(\frac{1}{2}, \beta)$
upper bounds	$\sqrt{24}$	[18, 54]	16.73
lower bounds	1.65	[353]	5.63
	$\alpha = \frac{3}{4}$		$r(\frac{3}{4}, \beta)$
upper bounds	$\sqrt{\frac{2\pi}{\sqrt{3}} \cdot \frac{1+\sqrt{3}}{\sqrt{8}}}$	[353]	13.73
lower bounds	1.42	[353]	10.60

Table 7.1: Summary of various $\sqrt{\cdot}$-separator theorems with their constants α and β. Here, $r(\alpha, \beta)$ denotes the ratio $r(\alpha, \beta) = \beta/(1 - \sqrt{\alpha})$, which is of central importance to the running time analysis of our algorithms, cf. Proposition 7.63.

In order to develop a flexible framework, we will do our calculations below always with the parameters α and β left unspecified up to the point where we try to give concrete numbers in the case of VERTEX COVER on planar graphs, which will serve as a running example. Also, we point out how the existence of ℓ-separators for $\ell > 2$ might improve the running time. In principle, our results also apply to graph problems for graphs from other graph classes with $\sqrt{\cdot}$-separator theorems as listed above. As indicated in [328], separator based techniques can be also used to solve counting problems instead of decision problems. This idea might be also interesting (but is yet unexplored) in the context of parameterized counting, see Chap. 8.

A separator theorem derived in the treewidth schema

In order to give the reader a better taste what separator theorems look like, let us review one that was given by Bodlaender in his introduction into treewidth [54], since this uses notions we have exhibited up to now and can be seen as showing techniques that will be useful later on in this chapter.

To this end, notice that there exist a slightly different notion of separator (also called *separator of type 2*): S is a separator of type 2 in a graph $G = (V, E)$ yielding chunks of size at most $\chi|V|$ if no connected component of $G - S$ has more than $\chi|V|$ vertices.

The following lemma can be relatively easily shown by using greedy techniques:

Lemma 7.49 *If $G = (V, E)$ has a separator S of type 2, yielding chunks of size at most $1/2 \cdot |V|$, then G has a separation (A_1, S', A_2) such that neither A_1 nor A_2 have more than $2/3 \cdot |V|$ elements.*

The first theorem we need is the following one that relates treewidth and separators of type 2 (see [54, Theorem 19]):

Theorem 7.50 *If $\mathrm{tw}(G) \leq k$, then there is a type-2 separator S, $|S| \leq k+1$, in G that yields chunks of size at most $1/2 \cdot (|V| - k)$.*

Namely, observe that in a graph of treewidth at most k, a suitably chosen bag may serve as a separator of type 2.

Algorithm 66 An algorithm for determining a small separator in a plane graph

```
Input(s): a plane $n$-vertex graph $G$
Output(s): a type-2 separator $S$, $|S| \leq 2\sqrt{6n}$ of type 2, yielding chunks of
  size at most $1/2 \cdot |V|$.
  Let $c = 1/2 \cdot \sqrt{n}$.
  Let $L_1, \dots, L_r$ be the layer decomposition of the plane graph $G$.
  Determine $s$ be such that $|\bigcup_{i<s} L_i| \leq 1/2 \cdot |V|$ and that $|\bigcup_{i>s} L_i| \leq 1/2 \cdot |V|$.
  if $|L_s| \leq c\sqrt{n}$ then
    output $L_s$
  else if $|L_i| > c\sqrt{n}$ for all $1 \leq i \leq r$ then
    $\{r \leq 1/c \cdot \sqrt{n}$ (since there are only $n$ vertices in the graph)$\}$
    $\{\mathrm{tw}(G) \leq 3r - 1 \leq 3/c\sqrt{n} - 1\}$
    Thm. 7.50 guarantees the existence of a sufficiently small separator $S$ of
    type 2 which the algorithm returns.
  else
    Let $s_1 < s$ be maximal with the property that $|L_{s_1}| \leq c\sqrt{n}$.
    Let $s_2 > s$ be minimal with the property that $|L_{s_2}| \leq c\sqrt{n}$.
    Let $L = \bigcup_{s_1<i<s_2} L_i$.
    Find small separator $S$ of type 2 in $G[L]$ according to Thm. 7.50.
    $\{$Notice that $\mathrm{out}(G[L]) \leq s_2 - s_1 - 1 \leq 1/c \cdot \sqrt{n}$, hence $\mathrm{tw}(G[L]) \leq$
    $3/c\sqrt{n} - 1.\}$
    return $L_{s_1} \cup L_{s_2} \cup S$
  end if
```

Theorem 7.51 *Alg. 66 produces to every plane n-vertex graph G a type-2 separator of size $2\sqrt{6n}$, yielding chunks of size at most $1/2 \cdot n$.*

With Lemma 7.49, we may conclude:

Corollary 7.52 *Planar graphs admit an $\sqrt{n}$-separator theorem with constants $\alpha = 2/3$ and $\beta \approx 4.90$.*

Variants of separator theorems

Cycle separators. In the literature, there are many separator theorems for planar graphs which guarantee that all the vertices of the separator lie on a simple cycle, provided that the given graph is biconnected or even triangulated. In fact, the current "record holder" in the case of $\alpha = 2/3$ yields a cycle separator, see [128]. From an algorithmic perspective, as explained below, the requirements of having biconnected or triangulated graphs are rarely met: even if the original graph was biconnected or triangulated, subgraphs which are obtained by recursive applications of separator theorems to a larger graph are not biconnected or triangulated in general. Therefore, we consider the following definition appropriate for our purposes:

Definition 7.53 We will call a separator S of a planar graph G *cycle separator* if there exists a triangulation $\hat{G}$ of G such that S forms a simple cycle in $\hat{G}$.

Note: some triangulation of a given planar graph can be computed in linear time.

Remark 7.54 *It will turn out that it is of special value (concerning the design of divide and conquer algorithms) to have separators that form simple cycles (within some triangulation of the given graph G), since then the Jordan curve theorem applies (for planar graphs), which basically means that the separator S splits G into an "inside"-part A_1 and an "outside"-part A_2. If G is a subgraph of a larger planar graph $\tilde{G}$, then this implies that each vertex v of $\tilde{G}$ that has neighbors in A_1 has no neighbors in A_2 and vice versa. This observation is important, since it means that a local property pertaining to vertex v of $\tilde{G}$ (like: v belongs to a dominating set or not) can only influence vertices in δA_1 or vertices in δA_2.*

Weighted separation. It is also possible to incorporate *weights* in most separator theorems. For our purposes, weights are nonnegative reals assigned to the vertices in a graph such that the sum of all weights in a graph is bounded by one. For weighted graphs, an $f(\cdot)$-separator theorem with constants α and β for graph class $\mathbb{G}$ guarantees, for any n-vertex graph $G \in \mathbb{G}$, the existence of a separation (A_1, S, A_2) of G such that

- neither A_1 nor A_2 has weight more than α, and
- S contains no more than $\beta f(n)$ vertices.

Other graph classes with separator theorems. Similar to the case of planar graphs, $\sqrt{\cdot}$-separator theorems are also known for other graph classes, e.g., for the class of graphs of bounded genus, see [131]. More generally, Alon, Seymour and Thomas proved a $\sqrt{\cdot}$-separator theorem for graph classes with an excluded complete graph minor [20, 21]. There are also interesting graph classes which are not minor closed for which $\sqrt{\cdot}$-separator theorems are known, as, e.g., the classes of ℓ-map graphs, see [92, 93].[7] Many comments of this paper apply to these more general situations, too.

7.5.2 Select&verify problems and glueability

For the approach outlined in this section, we want to describe necessary properties for graph problems that allow for separator based algorithms.

Select&verify graph problems

Definition 7.55 A set $\mathcal{G}$ of tuples (G, k), G an undirected graph with vertex set $V = \{v_1, \dots, v_n\}$ and k a positive real number, is called a *select&verify (graph) problem* if there exists a pair $(P_\cdot, \text{opt})$ with $\text{opt} \in \{\min, \max\}$, such that $P_\cdot$ is a function that assigns to G a polynomial time computable function of the form $P_G = P_G^{\text{sel}} + P_G^{\text{ver}}$, where $P_G^{\text{sel}} : \{0,1\}^n \to \mathbb{R}_{\geq 0}$, $P_G^{\text{ver}} : \{0,1\}^n \to \{0, \pm\infty\}$, and

$$(G,k) \in \mathcal{G} \quad \Leftrightarrow \quad \begin{cases} \text{opt}_{\vec{x}\in\{0,1\}^n} P_G(\vec{x}) \leq k & \text{if opt} = \min \\ \text{opt}_{\vec{x}\in\{0,1\}^n} P_G(\vec{x}) \geq k & \text{if opt} = \max\,. \end{cases}$$

For $\vec{x} = (x_1, \dots, x_n) \in \{0,1\}^n$ with $P_G(\vec{x}) \leq k$ if opt $=$ min and with $P_G(\vec{x}) \geq k$ if opt $=$ max, the vertex set *selected* by $\vec{x}$ and *verified* by P_G is $\{v_i \in V \mid x_i = 1, 1 \leq i \leq n\}$. A vector $\vec{x}$ is called *admissible* if $P_G^{\text{ver}}(\vec{x}) = 0$.

The intuition behind the term $P_\cdot = P_\cdot^{\text{sel}} + P_\cdot^{\text{ver}}$ is that the "selecting function" $P_\cdot^{\text{sel}}$ counts the size of the selected set of vertices and the "verifying function" $P_\cdot^{\text{ver}}$ verifies whether this choice of vertices is an admissible solution.

Observe that every select&verify graph problem that additionally admits a problem kernel of size $p(k)$ is solvable in time $\mathcal{O}(2^{p(k)}p(k) + T_K(n,k))$.

[7] The ℓ basically refers to the maximal size of a complete subgraph of a map graph.

Example 7.56 We now give some examples for select&verify problems by specifying the function $P_G = P_G^{\mathrm{sel}} + P_G^{\mathrm{ver}}$. In all cases below, the "selecting function" P_G for a graph $G = (V, E)$ will be

$$P_G^{\mathrm{sel}} = \sum_{v_i \in V} x_i.$$

Also, we use the convention that $0 \cdot (\pm\infty) = 0$.

1. In the case of VERTEX COVER, we have opt = min and choose

   $$P_G^{\mathrm{ver}}(\vec{x}) = \sum_{\{v_i,v_j\} \in E} \infty \cdot (1 - x_i)(1 - x_j),$$

   where this sum brings $P_G(\vec{x})$ to infinity whenever there is an uncovered edge. In addition, $P_G(\vec{x}) \leq k$ then guarantees a vertex cover set of size at most k. Clearly, P_G is polynomial time computable.

2. Similarly, in the case of INDEPENDENT SET, let opt = max and choose

   $$P_G^{\mathrm{ver}}(\vec{x}) = \sum_{\{v_i,v_j\} \in E} (-\infty) \cdot x_i \cdot x_j.$$

3. DOMINATING SET is another example for a select&verify graph problem. Here, for $G = (V, E)$, we have

   $$P_G^{\mathrm{ver}}(\vec{x}) = \sum_{v_i \in V} (\infty \cdot (1 - x_i) \cdot \prod_{\{v_i,v_j\} \in E} (1 - x_j)),$$

   where this sum brings $P_G(\vec{x})$ to infinity whenever there is a non-dominated vertex which is not in the selected dominating set. In addition, $P_G(\vec{x}) \leq k$ then guarantees a dominating set of size at most k.

4. Similar observations do hold for many other graph problems and, in particular, weighted variants of these.[8] As a source of problems, consider the variants of DOMINATING SET listed in [140, 228, 364].

Moreover, graph problems where a small (or large) *edge set* is sought for can often be reformulated into vertex set optimization problems by introducing an additional artificial vertex on each edge of the original graph. In this way, EDGE DOMINATING SET can be handled. Similarly, planar graph problems

[8] In the weighted case, one typically chooses a "selecting function" of the form $P_G^{\mathrm{sel}} = \sum_{v_i \in V} \alpha_i x_i$, where α_i is the weight of the vertex v_i.

where a small (or large) *face set* is looked for are expressible as select&verify problems of the dual graphs.

We will also need a notion of select&verify problems where the "selecting function" and the "verifying function" operate on a subgraph of the given graph.

Definition 7.57 Let $P_\cdot = P_\cdot^{\rm sel} + P_\cdot^{\rm ver}$ be the function of a select&verify problem. For an n-vertex graph G and subgraphs $G^{\rm ver} = (V^{\rm ver}, E^{\rm ver})$, $G^{\rm sel} = (V^{\rm sel}, E^{\rm sel}) \subseteq G$, we let

$$P_{G^{\rm ver}}(\vec{x} \mid G^{\rm sel}) := P^{\rm ver}_{G^{\rm ver}}(\pi_{V^{\rm ver}}(\vec{x})) + P^{\rm sel}_{G^{\rm sel}}(\pi_{V^{\rm sel}}(\vec{x})),$$

where $\pi_{V'}$ is the projection of the vector $\vec{x} \in \{0,1\}^n$ to the variables corresponding to the vertices in V'.

Glueability. We are going to solve graph problems, slicing the given graph into small pieces with the help of small separators. The separators will serve as boundaries between the different graph parts into which the graph is split. For each possible assignment of the vertices in the separators, we want to—independently—solve the corresponding problems on the graph parts and then reconstruct a solution for the whole graph by "gluing" together the solutions for the graph parts. We need to assign *colors* to the separator vertices in the course of the algorithm. Hence, our algorithm has to be designed in such a manner that it can also cope with colored graphs. In general (e.g., in the case of DOMINATING SET), it is not sufficient to simply use the two colors 1 (for encoding "in the selected set") and 0 (for "not in the selected set").

Let us introduce some auxiliary notions. Let $G = (V, E)$ be an undirected graph and let C_0, C_1 be finite, disjoint sets. A *C_0-C_1-coloring* of G is a function $\chi : V \to C_0 + C_1 + \{\#\}$.[9] For $V' \subseteq V$, a function $\chi : V' \to C_0 \uplus C_1$ can naturally be extended to a C_0-C_1-coloring of G by setting $\chi(v) = \#$ for all $v \in V \setminus V'$.

Consider a vector $\vec{x} \in \{0,1\}^{|V|}$. Let χ be a C_0-C_1-coloring of G. Then, $\vec{x}$ is *consistent* with χ, written $\vec{x} \sim \chi$, if, for $i = 0, 1$ and $j = 1, \ldots, |V|$, $\chi(v_j) \in C_i \;\Rightarrow\; x_j = i, .$

If χ is a C_0-C_1-coloring of G and if χ' is a C'_0-C'_1-coloring of G, then χ is *preserved* by χ', written $\chi \rightsquigarrow \chi'$, if $\forall v \in V\, \forall i \in \{0,1\}\, (\chi(v) \in C_i \Rightarrow \chi'(v) \in C'_i)$.

In the next section, when doing the divide and conquer approach with a given separator, we will deal with colorings on two different color sets: one

[9]The symbol # will be used for the undefined (i.e., not yet defined) color.

color set $C^{\text{int}} := C_0^{\text{int}} + C_1^{\text{int}} + \{\#\}$ of *internal* colors that will be used for the assignments of colors to the separator vertices and a color set $C^{\text{ext}} := C_0^{\text{ext}} + C_1^{\text{ext}} + \{\#\}$ of *external* colors that will be used for handing down the information in the divide-step of the algorithm. The idea is that, in each recursive step, we will be confronted with a graph "pre-colored" with external colors. Every function $\oplus$ that assigns to a pair $(\chi^{\text{ext}}, \chi^{\text{int}})$ with $\chi^{\text{ext}} : V \to C^{\text{ext}}$, $\chi^{\text{int}} : V \to C^{\text{int}}$, $\chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}$, a $(C_0^{\text{ext}}\text{-}C_1^{\text{ext}})$-coloring $\chi^{\text{ext}} \oplus \chi^{\text{int}}$ is called a *recoloring* if $\chi^{\text{int}} \rightsquigarrow \chi^{\text{ext}} \oplus \chi^{\text{int}}$.

From the point of view of recursion, χ^{ext} is the pre-coloring which a certain recursion instance "receives" from the calling instance and χ^{int} represents a coloring which this instance assigns to a certain part of the graph. The coloring $\chi^{\text{ext}} \oplus \chi^{\text{int}}$ is handed down in the recursion.

We now introduce the central notion of "glueable" select&verify problems. This formalizes those problems that can be solved with separator based divide and conquer techniques as described above.

Definition 7.58 A select&verify problem $\mathcal{G}$ given by $(P_{\cdot}, \text{opt})$ is *glueable with σ colors* if there exist

- a color set $C^{\text{int}} := C_0^{\text{int}} + C_1^{\text{int}} + \{\#\}$ of internal colors with $|C_0^{\text{int}} + C_1^{\text{int}}| = \sigma$;
- a color set $C^{\text{ext}} := C_0^{\text{ext}} + C_1^{\text{ext}} + \{\#\}$ of external colors;
- a polynomial time computable function $h : (\mathbb{R}_{\geq 0} \cup \{\pm\infty\})^3 \to \mathbb{R}_{\geq 0} \cup \{\pm\infty\}$;

and if, for every n-vertex graph $G = (V, E)$ and subgraphs $G^{\text{ver}}, G^{\text{sel}} \subseteq G$ with a separation (A_1, S, A_2) of G^{ver}, we find

- recolorings $\oplus_X$ for each $X \in \{A_1, S, A_2\}$, and
- for each internal coloring $\chi^{\text{int}} : S \to C_0^{\text{int}} \uplus C_1^{\text{int}}$,
  - subgraphs $G_{A_i}^{\text{ver}}(\chi^{\text{int}})$ of G^{ver} with $G^{\text{ver}}[A_i] \subseteq G_{A_i}^{\text{ver}}(\chi^{\text{int}}) \subseteq G^{\text{ver}}[\delta A_i]$ for $i = 1, 2$, and
  - subgraphs $G_S^{\text{ver}}(\chi^{\text{int}})$ of G^{ver} with $G_S^{\text{ver}}(\chi^{\text{int}}) \subseteq G^{\text{ver}}[S]$

such that, for each external coloring $\chi^{\text{ext}} : V \to C^{\text{ext}}$,

$$\begin{aligned} &\text{opt}_{\substack{\vec{x} \in \{0,1\}^n \\ \vec{x} \sim \chi^{\text{ext}}}} P_{G^{\text{ver}}}(\vec{x} \mid G^{\text{sel}}) \qquad (7.1) \\ &= \ \text{opt}_{\substack{\chi^{\text{int}} : S \to C_0^{\text{int}} \uplus C_1^{\text{int}} \\ \chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}}} h\big(\text{Eval}_{A_1}(\chi^{\text{int}}), \text{Eval}_S(\chi^{\text{int}}), \text{Eval}_{A_2}(\chi^{\text{int}})\big). \end{aligned}$$

Here, $\mathrm{Eval}_X(\cdot)$ for $X \in \{A_1, S, A_2\}$ is of the form

$$\mathrm{Eval}_X(\chi^{\mathrm{int}}) \quad = \quad \mathrm{opt}_{\substack{\vec{x}\in\{0,1\}^n \\ \vec{x}\sim(\chi^{\mathrm{ext}}\oplus_X\chi^{\mathrm{int}})}} P_{G_X^{\mathrm{ver}}(\chi^{\mathrm{int}})}(\vec{x} \mid G^{\mathrm{ver}}[X] \cap G^{\mathrm{sel}}). \tag{7.2}$$

Lemma 7.59 VERTEX COVER *and* INDEPENDENT SET *are glueable with 2 colors and* DOMINATING SET *is glueable with 4 colors.*

Proof. For VERTEX COVER(see Example 7.56.1), we use the color sets $C_i^\ell := \{i^\ell\}$ for $\ell \in \{\mathrm{int}, \mathrm{ext}\}$ and $i = 0, 1$. The function h is $h(x, y, z) = x + y + z$. The subgraphs $G_X^{\mathrm{ver}}(\chi^{\mathrm{int}})$ for $X \in \{A_1, S, A_2\}$ and $\chi^{\mathrm{int}} : S \to C_0^{\mathrm{int}} \uplus C_1^{\mathrm{int}}$ are $G_X^{\mathrm{ver}}(\chi^{\mathrm{int}}) := G^{\mathrm{ver}}[X]$. In this way, the subroutine $\mathrm{Eval}_S(\chi^{\mathrm{int}})$ checks whether the coloring χ^{int} yields a vertex cover on $G^{\mathrm{ver}}[S]$ and the subroutines $\mathrm{Eval}_{A_i}(\chi^{\mathrm{int}})$ compute the minimum size vertex cover on $G^{\mathrm{ver}}[A_i]$. However, we still need to make sure that all edges going from A_i to S are covered. If a vertex in S is assigned a 1^{int} by χ^{int}, the incident edges are already covered. In the case of a 0^{int}-assignment for a vertex $v \in S$, we can color all neighbors in $N(v) \cap A_i$ to belong to the vertex cover. This is done by the following recolorings $\oplus_{A_i}$. Define

$$(\chi^{\mathrm{ext}} \oplus_{A_i} \chi^{\mathrm{int}})(v) = \begin{cases} 0^{\mathrm{ext}} & \text{if } \chi^{\mathrm{int}}(v) = 0^{\mathrm{int}}, \\ 1^{\mathrm{ext}} & \text{if } \chi^{\mathrm{int}}(v) = 1^{\mathrm{int}} \text{ or} \\ & \text{if } \exists w \in N(v) \text{ with } \chi^{\mathrm{int}}(w) = 0^{\mathrm{int}}, \\ \#, & \text{otherwise.} \end{cases}$$

By this recoloring definition, an edge between a separator vertex and a vertex in A_i which is not covered by the separator vertex (due to the currently considered internal covering) will be covered by the vertex in A_i. Our above reasoning shows that—with these settings—Equation (7.1) in Definition 7.58 is satisfied.

INDEPENDENT SET (see Example 7.56.2)) is shown to be glueable with 2 colors by a similar idea.

To show that DOMINATING SET (see Example 7.56.3)) is glueable with 4 colors, we use the following color sets

$$\begin{array}{ll} C_0^{\mathrm{int}} := \{0_{A_1}^{\mathrm{int}}, 0_{A_2}^{\mathrm{int}}, 0_S^{\mathrm{int}}\}, & C_1^{\mathrm{int}} := \{1^{\mathrm{int}}\}, \\ C_0^{\mathrm{ext}} := \{0^{\mathrm{ext}}\}, & C_1^{\mathrm{ext}} := \{1^{\mathrm{ext}}\}. \end{array}$$

The semantics of these colors is as follows. Assigning the color 0_X^{int}, for $X \in \{A_1, A_2, S\}$, to vertices in a current separation $V = A_1 \uplus S \uplus A_2$ means that the vertex is not in the dominating set and will be dominated by a vertex in X. Clearly, 1^{int} will mean that the vertex belongs to the dominating set. The external colors simply hand down the information whether a vertex belongs to the dominating set, represented by 1^{ext}, or whether it is not in the dominating set *and* still needs to be dominated, represented by 0^{ext}.[10] The function h simply is addition, i.e., $h(x,y,z) = x+y+z$. When handing down the information to the subproblems, for a given internal coloring $\chi^{\text{int}} : S \to C_0^{\text{int}} \uplus C_1^{\text{int}}$, we define

$$\begin{array}{rcl} G_{A_i}^{\text{ver}}(\chi^{\text{int}}) & := & G^{\text{ver}}[A_i \cup (\chi^{\text{int}})^{-1}(\{1^{\text{int}}, 0_{A_i}^{\text{int}}\})] \quad \text{and} \\ G_S^{\text{ver}}(\chi^{\text{int}}) & := & G^{\text{ver}}[(\chi^{\text{int}})^{-1}(\{1^{\text{int}}, 0_S^{\text{int}}\})]. \end{array}$$

The recolorings $\oplus_X$ for $X \in \{A_1, S, A_2\}$ are chosen to be

$$(\chi^{\text{ext}} \oplus_X \chi^{\text{int}})(v) = \begin{cases} 0^{\text{ext}} & \text{if } \chi^{\text{int}}(v) \in C_0^{\text{int}}, \\ 1^{\text{ext}} & \text{if } \chi^{\text{int}}(v) = 1^{\text{int}}, \\ \#, & \text{otherwise.} \end{cases}$$

Let us explain in a few lines why—with these settings—Equation (7.1) in Definition 7.58 is satisfied. If an internal coloring χ^{int} assigns color 0_X^{int} ($X \in \{A_1, S, A_2\}$) to a vertex in S, then this vertex needs to be dominated by a neighbor in X. This will be checked in $\text{Eval}_X(\chi^{\text{int}})$ using the graph $G_X^{\text{ver}}(\chi^{\text{int}})$. To this end, vertices assigned the color 0_X^{int} (i.e., the set $(\chi^{\text{int}})^{-1}(\{0_X^{\text{int}}\})$) are included in $G_X^{\text{ver}}(\chi^{\text{int}})$. The vertices assigned color 1^{int} (i.e., $(\chi^{\text{int}})^{-1}(\{1^{\text{int}}\})$) also need to be handed down to the subroutines, since such a vertex may already dominate vertices in X. The recolorings merge the given external coloring χ^{ext} with the current internal coloring χ^{int} in a way that already assigned colors from C_i^{int} or C_i^{ext} ($i = 0,1$) become i^{ext}. The terms $\text{Eval}_{A_i}(\chi^{\text{int}})$ then compute (for each internal coloring χ^{int}) the size of a minimum dominating set in A_i under the constraint that some vertices in δA_i still need to be dominated (namely, the vertices in $\delta A_i \cap (\chi^{\text{ext}} \oplus_{A_i} \chi^{\text{int}})^{-1}(0^{\text{ext}})$) and some vertices in δA_i can already be assumed to be in the dominating set (namely, the vertices in $\delta A_i \cap (\chi^{\text{ext}} \oplus_{A_i} \chi^{\text{int}})^{-1}(1^{\text{ext}})$). The term $\text{Eval}_S(\chi^{\text{int}})$ checks the correctness of the internal coloring χ^{int} of S. ▮

Note that, from the point of view of divide-and-conquer algorithms, three colors are enough for DOMINATING SET, since the color 1^{int} already determines the color 0_S^{int}.
We illustrate the ideas of the dominating set algorithm by using an example.

Example 7.60 Consider DOMINATING SET for the separated graph in Fig. 7.15. Beginning with the external coloring $\chi^{\text{ext}} \equiv \#$ and $G^{\text{ver}} = G^{\text{sel}} = G$, we need

[10] A vertex that is not in the dominating set but is already guaranteed to be dominated, e.g., by a vertex in the current separator, will never be handed down, since these vertices are of no use in the sequel of the recursion.

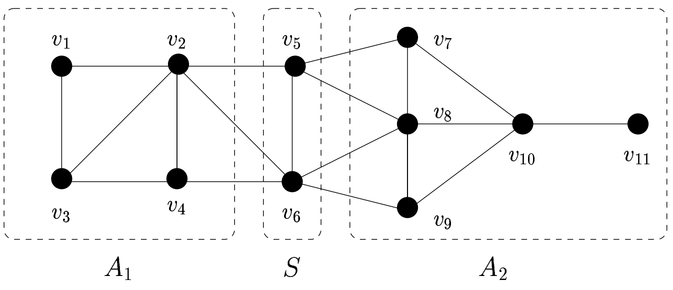


Figure 7.15: A partitioned graph.

to go over all $4^2 = 16$ internal colorings $\chi^{\text{int}} : S \to \{0^{\text{int}}_{A_1}, 0^{\text{int}}_{S}, 0^{\text{int}}_{A_2}, 1^{\text{int}}\}$ (which trivially satisfy $\chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}$). As an example, we choose χ^{int} with $v_5 \mapsto 0^{\text{int}}_{A_1}$, $v_6 \mapsto 0^{\text{int}}_{A_2}$. In this case, we get $G^{\text{ver}}_{A_1}[\chi^{\text{int}}] = G^{\text{ver}}[\{v_1, \dots, v_5\}]$, $G^{\text{ver}}_{S}[\chi^{\text{int}}] = \emptyset$, and $G^{\text{ver}}_{A_2}[\chi^{\text{int}}] = G^{\text{ver}}[\{v_6, \dots, v_{11}\}]$. Recursively, we will use these graphs for the verifying function and the graphs $G^{\text{ver}}[A_1]$, $G^{\text{ver}}[S]$, and $G^{\text{ver}}[A_2]$ for the selecting function. The external colorings that will be handed down to the subproblems after the recoloring look as follows: On the graph $G^{\text{ver}}_{A_1}[\chi^{\text{int}}]$, we have $\chi_1(v_i) := (\chi^{\text{ext}} \oplus_{A_1} \chi^{\text{int}})(v_i) = \#$ for $i = 1, \dots, 4$, and $\chi_1(v_5) = 0^{\text{ext}}$. On the graph $G^{\text{ver}}_{A_2}[\chi^{\text{int}}]$, we have $\chi_2(v_i) := (\chi^{\text{ext}} \oplus_{A_2} \chi^{\text{int}})(v_i) = \#$ for $i = 7, \dots, 11$, and $\chi_2(v_6) = 0^{\text{ext}}$. Clearly,

$$\text{Eval}_{A_1}(\chi^{\text{int}}) = 2, \quad \text{Eval}_{S}(\chi^{\text{int}}) = 0, \quad \text{and} \quad \text{Eval}_{A_2}(\chi^{\text{int}}) = 2.$$

The minimum in Equation (7.2) of Definition 7.58 for $X = A_1$ is obtained, e.g., by choosing the vertices v_1 and v_2 (note that the latter needs to be chosen, since $\chi_1(v_5) = 0^{\text{ext}}$, meaning that v_5 is forced to be dominated in this term). The minimum for A_2 is obtained, e.g., by choosing the vertices v_8 and v_{10} (again, $\chi_2(v_6) = 0^{\text{ext}}$ forces either v_8 or v_9 to be in the dominating set). Hence,

$$h(\text{Eval}_{A_1}(\chi^{\text{int}}), \text{Eval}_{S}(\chi^{\text{int}}), \text{Eval}_{A_2}(\chi^{\text{int}})) = 2 + 0 + 2 = 4.$$

We obtain an optimal result, e.g., for the choice of the internal coloring χ^{int} with $\chi^{\text{int}}(v_5) = \chi^{\text{int}}(v_6) = 0^{\text{int}}_{A_2}$. Here, we get $\text{Eval}_{A_1}(\chi^{\text{int}}) = 1$, $\text{Eval}_{S}(\chi^{\text{int}}) = 0$, and $\text{Eval}_{A_2}(\chi^{\text{int}}) = 2$, for the possible choices of v_3, v_8, v_{10} as dominating set vertices.

Remark 7.61 *In the case that recurrences are based on separator theorems yielding ℓ-separators, let us call a problem ℓ-*glueable with σ_ℓ colors *if σ_ℓ colors are to be distinguished in the recursion. For example, an extension of our previous lemma shows that* DOMINATING SET *is ℓ-glueable with $\ell + 2$*

colors, where $|C_0^{int}| = \ell + 1$. There need not be, however, a dependence of the number of colors on the number of graph parts: both VERTEX COVER *and* INDEPENDENT SET *are ℓ-glueable with two colors.*

Besides the problems stated in the preceding Lemma 7.59, many more select&verify problems are glueable, for example, those which are listed in [363, 364, 365]. In particular, weighted versions and variations of the problems discussed in Lemma 7.59 are glueable.

7.5.3 Fixed-parameter divide-and-conquer algorithms

In this section, we provide the basic framework for deriving fixed-parameter algorithms based on the concepts we introduced so far.

Using glueability for divide-and-conquer

Fix a graph class $\mathbb{G}$ for which a $\sqrt{\cdot}$-separator theorem with constants α and β (cf. Definition 7.48) is known. We consider a glueable select&verify graph problem $\mathcal{G}$ defined by $(P_\cdot, \text{opt})$. The evaluation of the term $\text{opt}_{\vec{x}\in\{0,1\}^n} P_G(\vec{x})$ (cf. Definition 7.55) can be done recursively according to the following strategy.
The sizes of the subproblems, i.e., the sizes of the graphs $G_{A_i}^{\text{ver}}(\chi^{\text{int}})$ which are used in the recursion, play a crucial role in the analysis of the running time of this algorithm. A particularly nice situation is given by the following problems.

Definition 7.62 A glueable select&verify problem is called a *slim problem* if the subgraphs $G_{A_i}^{\text{ver}}(\chi^{\text{int}})$ are only by a constant number of vertices larger than $G^{\text{ver}}[A_i]$, i.e., if there exists an $\eta \geq 0$ such that $|V(G_{A_i}^{\text{ver}}(\chi^{\text{int}}))| \leq |A_i| + \eta$ for all internal colorings $\chi^{\text{int}} : S \to C^{\text{int}}$.

Note that the proof of Lemma 7.59 shows that both VERTEX COVER and INDEPENDENT SET are slim with $\eta = 0$, whereas DOMINATING SET is not.

The following proposition gives the running time of the above algorithm in terms of the parameters of the separator theorem used and the select&verify problem considered. In order to assess the time required for the above given divide-and-conquer algorithm, we use the following abbreviations for the running times of certain subroutines:

- $T_S(n)$ denotes the time to find a "sufficiently small" separator in an n-vertex graph from class $\mathbb{G}$.

---

1. Start the computation by evaluating

$$\mathrm{opt}_{\vec{x}\in\{0,1\}^n} P_G(\vec{x}) \;=\; \mathrm{opt}_{\vec{x}\in\{0,1\}^n,\,\vec{x}\sim{\chi^{\mathrm{ext}}}_0} P_{G^{\mathrm{ver}}}(\vec{x} \mid G^{\mathrm{sel}}),$$

   where "${\chi^{\mathrm{ext}}}_0 \equiv \#$" is the everywhere undefined external coloring and $G^{\mathrm{ver}} = G^{\mathrm{sel}} = G$ (also cf. Definition 7.57).

2. When $\mathrm{opt}_{\vec{x}\in\{0,1\}^n,\,\vec{x}\sim\chi^{\mathrm{ext}}} P_{G^{\mathrm{ver}}}(\vec{x} \mid G^{\mathrm{sel}})$ needs to be calculated for some subgraphs $G^{\mathrm{sel}}, G^{\mathrm{ver}} \subseteq G$, and an external coloring $\chi^{\mathrm{ext}} : V(G) \to C_0^{\mathrm{ext}} \uplus C_1^{\mathrm{ext}} \uplus \{\#\}$, we do the following:

   (a) If G^{ver} has size greater than some constant c, then find a $\sqrt{\cdot}$-separator S for G^{ver} with $V(G^{\mathrm{ver}}) = A_1 \uplus S \uplus A_2$.

   (b) Define $\Phi := \{\chi^{\mathrm{int}} : S \to C_0^{\mathrm{int}} \uplus C_1^{\mathrm{int}} \mid \chi^{\mathrm{ext}} \rightsquigarrow \chi^{\mathrm{int}}\}$.
   For all internal colorings $\chi^{\mathrm{int}} \in \Phi$ do:

      i. Determine $\mathrm{Eval}_{A_i}(\chi^{\mathrm{int}})$ recursively for $i = 1, 2$.

      ii. Determine $\mathrm{Eval}_S(\chi^{\mathrm{int}})$.

   (c) Return $\mathrm{opt}_{\chi^{\mathrm{int}}\in\Phi}\, h(\mathrm{Eval}_{A_1}(\chi^{\mathrm{int}}), \mathrm{Eval}_S(\chi^{\mathrm{int}}), \mathrm{Eval}_{A_2}(\chi^{\mathrm{int}}))$.

- $T_M(n)$ denotes the time to construct the modified graphs $G_X^{\mathrm{ver}}(\chi^{\mathrm{int}}) \in \mathbb{G}$ and the modified colorings $(\chi^{\mathrm{ext}} \oplus_X \chi^{\mathrm{int}})$ (for $X \in \{A_1, S, A_2\}$ and each internal coloring $\chi^{\mathrm{int}} \in \Phi$ from an n-vertex graph from class $\mathbb{G}$.
- $T_E(m)$ is the time to evaluate $\mathrm{Eval}_S(\chi^{\mathrm{int}})$ for any $\chi^{\mathrm{int}} \in \Phi$ in a separator S of size m.
- $T_G(n)$ is the time for gluing the results obtained by two sub-problems each of size $\mathcal{O}(n)$.

In the following, we assume that all these functions are polynomials.

Proposition 7.63 *Let $\mathbb{G}$ be a graph class for which a $\sqrt{\cdot}$-separator theorem with constants α and β is known and let $\mathcal{G}$ be a select&verify problem defined by $(P_\cdot, \mathrm{opt})$ that is glueable with σ colors. Then, for every $G \in \mathbb{G}$, $\mathrm{opt}_{\vec{x}\in\{0,1\}^n} P_G(\vec{x})$ can be computed in time*

$$c(\alpha', \beta, \sigma)^{\sqrt{n}} q(n), \quad \text{where} \quad c(\alpha', \beta, \sigma) = \sigma^{\beta/(1-\sqrt{\alpha'})}.$$

Here, $\alpha' = \alpha + \varepsilon$ for any $\varepsilon \in (0, 1-\alpha)$ and $q(\cdot)$ is some polynomial; the running time analysis only holds for $n \geq n_0(\varepsilon)$.

If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators (and $\mathbb{G}$ is the class of planar graphs), then the running time for the computation is $c(\alpha,\beta,\sigma)^{\sqrt{n}}q(n)$, which then holds for all n.

The (rather lengthy) proof is contained in [14].

Remark 7.64 *A similar proposition holds for graph classes on which an ℓ-separator theorem is known with constants α and β. It might turn out that such separator theorems have better ratio $\beta/(1-\sqrt{\alpha})$, which, in turn, would directly improve the running time in Proposition 7.63.*

How (linear) problem kernels help

If the considered parameterized problem has a problem kernel of size dk, we can use the considerations we have made up to this point in order to obtain fixed-parameter algorithms whose exponential term is of the form $c^{\sqrt{k}}$ for some constant c. More generally, a problem kernel of size $p(k)$ yields exponential terms of the form $c^{\sqrt{p(k)}}$.

Theorem 7.65 *Assume the following:*

- *Let $\mathbb{G}$ be a graph class for which a $\sqrt{\cdot}$-separator theorem with constants α and β is known,*
- *let $\mathcal{G}$ be a select&verify problem defined by $(P_\cdot, \mathrm{opt})$ glueable with σ colors, and*
- *suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ on $\mathbb{G}$ computable in time $T_K(n,k)$.*

Then, there is an algorithm to decide $(G,k)\in\mathcal{G}$, for a graph $G\in\mathbb{G}$, in time

$$c(\alpha',\beta,\sigma)^{\sqrt{p(k)}}q(k)+T_K(n,k), \quad \textit{where} \quad c(\alpha',\beta,\sigma)=\sigma^{\beta/(1-\sqrt{\alpha'})}, \tag{7.3}$$

and $\alpha'=\alpha+\varepsilon$ for any $\varepsilon\in(0,1-\alpha)$, holding only for $k\geq k_0(\varepsilon)$, where $q(\cdot)$ is some polynomial.

If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators (on the class $\mathbb{G}$ of planar graphs), then the running time for the computation is

$$c(\alpha,\beta,\sigma)^{\sqrt{p(k)}}q(k)+T_K(n,k),$$

which then holds for all k.

In particular, Theorem 7.65 means that, for glueable select&verify problems for planar graphs that admit a *linear* problem kernel of size dk, we can get an algorithm of running time

$$O\big(c(\alpha,\beta,\sigma,d)^{\sqrt{k}}q(k)+T_K(n,k)\big),\quad \text{where}\quad c(\alpha,\beta,\sigma,d)=\sigma^{\sqrt{d}\beta/(1-\sqrt{\alpha})}.$$

Obviously, the choice of the separator theorem has a decisive impact on the constants of the corresponding algorithms. In particular, our running time analysis shows that the ratio $r(\alpha,\beta):=\beta/(1-\sqrt{\alpha})$ has a direct and significant influence on the running time. In Table 7.1, this ratio is computed for the various $\sqrt{\cdot}$-separator theorems. In the following example we use these ratios explicitly.

Example 7.66 In the case of VERTEX COVER on planar graphs, we can take $d=2$, $\alpha=2/3$, and $\beta=\sqrt{2/3}=\sqrt{4/3}$ (see [128]) with the ratio $r(\alpha,\beta)\approx 10.74$. In this way, we obtain an algorithm with running time $\mathcal{O}(2^{\sqrt{2}\cdot 10.74\cdot\sqrt{k}}+nk)$. Neglecting polynomial terms, we have such obtained a $c^{\sqrt{k}}$-algorithm with $c\approx 2^{15.19}\approx 37381$. By way of contrast, taking $d=2$, $\alpha=3/4$, and $\beta=\sqrt{2\pi/\sqrt{3}}\cdot(1+\sqrt{3})/\sqrt{8}\approx 1.84$ (see [353]) with $r(\alpha,\beta)\approx 13.73$, we get an algorithm with running time $\mathcal{O}(2^{\sqrt{2}\cdot 13.73\cdot\sqrt{k}}+nk)$. This means, we have a $c^{\sqrt{k}}$-algorithm with $c\approx 2^{19.42}\approx 701459$.

The constants obtained by this first approach are admittedly bad. However, by a careful analysis of how separator theorems are obtained in the literature, we can show in [14]:

Theorem 7.67 *Let $\mathcal{G}$ be a select&verify problem on planar graphs defined by $(P_{\cdot},\mathrm{opt})$ which is glueable with σ colors, and suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ computable in time $T_K(n,k)$.*

Then, there is an algorithm to decide $(G,k)\in\mathcal{G}$, for an n-vertex planar graph G, in time

$$c(\alpha',\sigma)^{\sqrt{p(k)}}q(k)+T_K(n,k),\quad \textit{where}\quad c(\alpha',\sigma)\approx\sigma^{1.80665/(1-\sqrt{\alpha'})},$$

and $\alpha'=2/3+\varepsilon$ for any $\varepsilon\in(0,1/3)$, holding only for $k\geq k_0(\varepsilon)$, where $q(\cdot)$ is some polynomial.

If $\mathcal{G}$ is slim, then the running time for the computation is

$$c(2/3,\sigma)^{\sqrt{p(k)}}q(k)+T_K(n,k),$$

which then holds for all k.

Example 7.68 For PLANAR VERTEX COVER, we obtain by the previous theorem an $\mathcal{O}^*(f(k))$-algorithm with $f(k) \leq 2^{13.9234\sqrt{k}} \approx 15537^{\sqrt{k}}$, which obviously beats the figures in Example 7.66.

To further improve on our constants, it is possible to analyze a planar $\sqrt{\cdot}$-separator theorem yielding 3-separators (due to Djidjev [130]) in essentially the same way as sketched above, leading to the following theorem:

Theorem 7.69 *Let $\mathcal{G}$ be a select&verify problem on planar graphs defined by $(P_\cdot, \mathrm{opt})$ which is 3-glueable with σ colors, and suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ computable in time $T_K(n,k)$.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for an n-vertex planar graph G, in time

$$c(\alpha', \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k), \quad \textit{where} \quad c(\alpha',\sigma) \approx \sigma^{2.7056/(1-\sqrt{\alpha'})},$$

and $\alpha' = 1/2 + \varepsilon$ for any $\varepsilon \in (0, 1/2)$, holding only for $k \geq k_0(\varepsilon)$, where $q(\cdot)$ is some polynomial.

If $\mathcal{G}$ is slim, then the running time for the computation is

$$c(1/2, \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k),$$

which then holds for all k.

Example 7.70 For VERTEX COVER on planar graphs, we obtain in this way an $\mathcal{O}^*(f(k))$-algorithm with $f(k) \leq 2^{13.0639\sqrt{k}} \approx 8564^{\sqrt{k}}$, which again beats the figure derived in Example 7.68.

Observe that Theorem 7.69 is not always yielding a better algorithm than Theorem 7.67, since possibly more colors are needed in the recursion for 3-glueability than for (2-)glueability, see Remark 7.61.

Remark 7.71 *We discuss the importance of cycle separator theorems or slim graph problems. Assume that none of these two conditions is met in a given situation. Then, the claimed bound from Equation (7.3) of Theorem 7.65 is only true for some $\alpha' = \alpha + \varepsilon$ with $\varepsilon \in (0, 1-\alpha)$. Now, there is a certain trade-off in the choice of ε:*

1. *The factor $\beta/(1-\sqrt{\alpha'})$ in the exponent of $c(\alpha', \beta, \sigma)$ tends to infinity if α' tends to one, i.e., if ε is as large as possible.*

2. *The analysis of Theorem 7.65 is only valid if $p(k) \geq (\beta/\varepsilon)^2$.*

Keeping in mind that typical values of $p(k)$ are not very large in practical cases, the second point means that, since β is fixed, ε should be comparatively large, otherwise, β/ε would be greater than $\sqrt{p(k)}$. This gives us very bad constants in the analysis due to the first point.

As explained in the following example, Theorem 7.65 is not only interesting in the case of planar graph problems.

Example 7.72 Since VERTEX COVER is a slim problem which has a linear size kernel, Theorem 7.65 yields a $c^{\sqrt{gk}}$-algorithm for $\mathbb{G}_g$, where $\mathbb{G}_g$ denotes the class of graphs of genus bounded by g; see [131], where the existence of a separator of size $\mathcal{O}(\sqrt{gn})$ for n-vertex graphs from $\mathbb{G}_g$ was proven. For the same reason, we get a $c^{\sqrt{gk}}$-algorithm for INDEPENDENT SET on $\mathbb{G}_g$. Similarly, for the class of ℓ-map graphs, we obtain $c^{\sqrt{\ell k}}$-algorithms for VERTEX COVER and for INDEPENDENT SET based on [92].

Note that these are the first examples of fixed-parameter algorithms with sublinear exponents for bounded genus graphs and for ℓ-map graphs. In this sense, the techniques discussed in this paper apply to a wider range of graphs compared to the approach discussed in the next section. The "Layerwise Separation Property" we provide there makes sense for planar graphs only, although it might be extensible to map graphs.

7.6 The beauty of small kernels: layerwise separation

This approach differs strongly from research directions, where running times of algorithms are improved in a very problem-specific manner (e.g., by extremely sophisticated case-distinctions). For example, once one can show that a problem has the so-called "Layerwise Separation Property," one can run a general algorithm which quickly computes a tree decomposition of guaranteed small width (independent of the concrete problem). In summary, the heart of our approach can roughly be sketched as follows: If...

1. ...one can show that a graph problem carries some nice properties (e.g., the Layerwise Separation Property) and

2. ...one can determine some corresponding "problem-parameters" for these properties (e.g., the width and the size-factor of the Layerwise Separation Property);

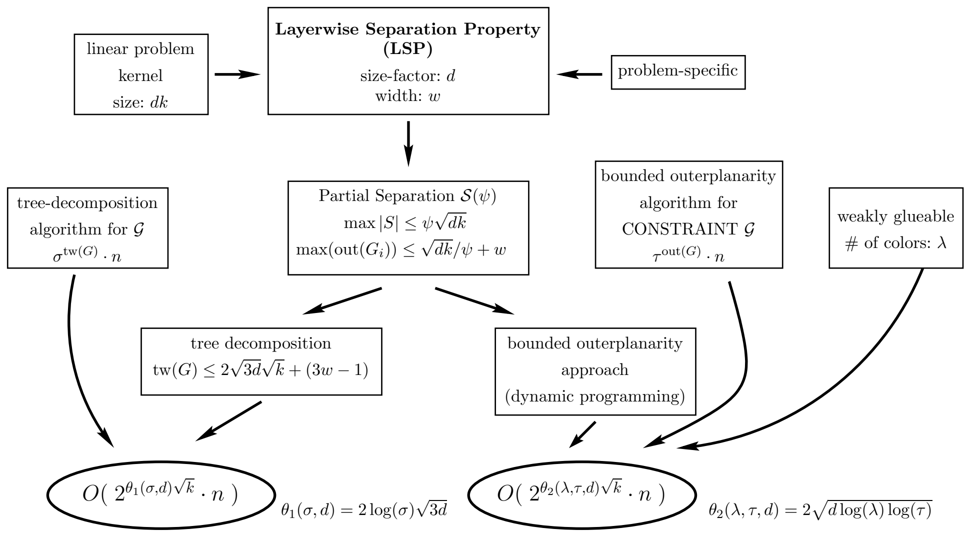

linear problem kernel size: dk
Layerwise Separation Property (LSP)
size-factor: d
width: w
problem-specific
tree-decomposition algorithm for $\mathcal{G}$
$\sigma^{\mathrm{tw}(G)} \cdot n$
Partial Separation $\mathcal{S}(\psi)$
$\max |S| \leq \psi\sqrt{dk}$
$\max(\mathrm{out}(G_i)) \leq \sqrt{dk}/\psi + w$
bounded outerplanarity algorithm for CONSTRAINT $\mathcal{G}$
$\tau^{\mathrm{out}(G)} \cdot n$
weakly glueable
# of colors: λ
tree decomposition
$\mathrm{tw}(G) \leq 2\sqrt{3d}\sqrt{k} + (3w-1)$
bounded outerplanarity approach (dynamic programming)
$O(\ 2^{\theta_1(\sigma,d)\sqrt{k}} \cdot n\ )$
$\theta_1(\sigma, d) = 2\log(\sigma)\sqrt{3d}$
$O(\ 2^{\theta_2(\lambda,\tau,d)\sqrt{k}} \cdot n\ )$
$\theta_2(\lambda, \tau, d) = 2\sqrt{d\log(\lambda)\log(\tau)}$

Figure 7.16: Road map of our methodology for planar graph problems.

then one gets an algorithm of running time $\mathcal{O}(c^{\sqrt{k}}n^{\mathcal{O}(1)})$, where we give concrete formulas on how to evaluate the constant c as a function of these problem-parameters.

In a first phase, one separates the graph in a particular way ("layerwise"). The key property of a graph problem to allow such an approach will be the so-called "Layerwise Separation Property." The details herefore are presented in Section 7.6.1. It will be shown that such a property holds for quite a large class of graph problems including those which admit a linear problem kernel. This property assures that the planar graph can be separated nicely.

In a second phase, the problem is solved on the layerwisely separated graph. We present two independent ways to achieve this in Section 7.6.2. Either, using the separators to set up a tree decomposition of width $\mathcal{O}(\sqrt{k})$ and solving the problem using this tree decomposition; or using a combination of a trivial approach on the separators and some algorithms working on graphs of bounded outerplanarity (see [32]) for the partitioned rest graphs. Figure 7.16 gives a general overview of our methodology presented in the following two sections. As noted before, in the second phase (Section 7.6.2) we will describe two independent ways to solve the underlying graph problem on the layerwisely separated graph. Both the tree decomposition as well as the bounded outerplanarity approach do have their pros and cons, which is why we present both of them. As to the tree decomposition approach, its advantage is its greater generality (up to the tree decomposition it is the same for all graph problems). In particular, it is definitely easier to implement in practice, also due to its universality and mathematical elegance.

By way of contrast, as to the bounded outerplanarity approach, in some cases we obtain better (theoretical worst-case) time complexity bounds for our algorithms in comparison with the tree decomposition approach. Moreover, the space consumption is significantly smaller, because the tree decomposition approach works in its dynamic programming part with possibly large tables. To achieve this, however, we need more complicated formalism and more constraints concerning the underlying graph problems.

Recall the notion of a *separation* (A_1, S, A_2) of G; the graphs $G[A_i]$ are called the *graph chunks* of the separation. Given a separation (A_1, S, A_2), we use the shorthands $\delta A_i := A_i \uplus S$ for $i = 1, 2$. We will also consider *X-Y-separators* with $X, Y \subseteq V$: such a separator cuts every path from X to Y in G.

7.6.1 Phase 1: Layerwise separation

In this section, we exploit the layer-structure of a plane graph in order to gain a "nice" separation of the graph. It is important that a YES-instance (G, k) (where G is a plane graph) of the graph problem $\mathcal{G}$ admits a so-called "layerwise separation" of small size. By this, we mean, roughly speaking, a separation of the plane graph G (i.e., a collection of separators for G), such that each separator is contained in the union of constantly many subsequent layers (see conditions 1 and 2 of the following definition). For (fixed-parameter) algorithmic purposes, it will be important that the corresponding separators are "small" (see condition 3 of the definition).

Definition 7.73 Let $(G = (V, E), \phi)$ be a plane graph (where ϕ is the embedding) of outerplanarity $r := \text{out}(G, \phi)$, and let $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$ be its layer decomposition. A *layerwise separation of width w and size s* of (G, ϕ) is a sequence $(S_1, \ldots, S_r)$ of subsets of V, with the properties that, for $i = 1, \ldots, r$:[11]

1. $S_i \subseteq \bigcup_{j=i}^{i+(w-1)} L_j$,
2. S_i is an L_{i-1}-L_{i+w} separator, and
3. $\sum_{j=1}^{r} |S_j| \leq s$.

The crucial property that makes the algorithms developed in this paper work is what we call the "Layerwise Separation Property."

[11]By default, we let $S_i := \emptyset$ for all $i < 1$ and $i > r$.

Definition 7.74 A parameterized problem $\mathcal{G}$ for planar graphs is said to have the *Layerwise Separation Property* (abbreviated by: LSP) of *separation width* w and *size-factor* d if for each $(G, k) \in \mathcal{G}$ and every planar embedding ϕ of G, the plane graph (G, ϕ) admits a layerwise separation of width w and size dk.

How can layerwise separations be obtained?

The LSP can be shown directly for many parameterized graph problems.

Example 7.75

1. Consider PLANAR VERTEX COVER. Here, we get constants $w = 2$ and $d = 2$. In fact, for $(G, k) \in$ VERTEX COVER (and any planar embedding ϕ of G) with a "witnessing" vertex cover V' of size k, the sets $S_i := (L_i \cup L_{i+1}) \cap V'$ form a layerwise separation, given the layer decomposition $\mathcal{L}(G, \phi) = (L_1, \dots, L_r)$.
2. We have already seen how the non-trivial fact is proven that for PLANAR DOMINATING SET this property holds, yielding constants $w = 3$ and $d = 51$.

Fenau and Juedes ([179], also described below) have shown that all problems describable by formulas from Planar $\mathcal{TMIN}_1$ (as defined in [256]) satisfy LSP. This class includes PLANAR RED-BLUE DOMINATING SET, FACE COVER, and PLANAR EDGE DOMINATING SET.

A large class of parameterized graph problems for which the LSP holds is given whenever there exists a reduction to a linear problem kernel.

Lemma 7.76 *Let $\mathcal{G}$ be a parameterized problem for planar graphs that admits a problem kernel of size dk. Then, the parameterized problem $\mathcal{G}'$ where each instance is replaced by its problem kernel has the LSP of width 1 and size-factor d.*

Proof. Let $(G', k') \in \mathcal{G}'$ with $k' \leq dk$ be the problem kernel of $(G, k) \in \mathcal{G}$, and let $\mathcal{L}(G', \phi') = (L'_1, \dots, L'_{r'})$ be the layer decomposition of (G', ϕ') (where ϕ' is any embedding). Let $r' = \text{out}(G', \phi')$. Observe that $r' \leq \frac{dk}{3}$ since each layer has to consist of at least 3 vertices. Then, the sequence $S_i := L'_i$ for $i = 1, \dots, r'$ is a layerwise separation of width 1 and size dk of (G', ϕ'). ∎

Chap. 4 gives us therefore already a collection of problems that satisfy LSP. In particular, we have:

Example 7.77

1. PLANAR VERTEX COVER has the LSP of width 1 and size-factor 2 (which is even better than what was shown in Example 7.75).

2. PLANAR INDEPENDENT SET has the LSP of width 1 and size-factor 4 on the set of reduced instances.

3. PLANAR DOMINATING SEThas the LSP of width 1 and size-factor 67. Observe that these bounds are worse than the one mentioned in Example 7.75, since it is the size-factor that mainly influences the running time of our algorithms.

What are layerwise separations good for?

From a layerwise separation of small size (say bounded by $\mathcal{O}(k)$), we are able to choose a set of separators such that their size is bounded by $\mathcal{O}(\sqrt{k})$ and—at the same time—the subgraphs into which these separators cut the original graph have outerplanarity bounded by $\mathcal{O}(\sqrt{k})$. In order to formalize such a choice of separators from a layerwise separation, we give the following definition.

Definition 7.78 Let $(G = (V, E), \phi)$ be a plane graph with layer decomposition $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$. A *partial layerwise separation of width* w is a sequence $\mathcal{S} = (S_1, \ldots, S_q)$ such that there exist $i_0 = 1 < i_1 < \ldots < i_q < r = i_{q+1}$ such that for $i = 1, \ldots, q$:[12]

1. $S_j \subseteq \bigcup_{\ell=i_j}^{i_j+(w-1)} L_\ell$,

2. $i_j + w \leq i_{j+1}$ (so, the sets in $\mathcal{S}$ are pairwise disjoint) and

3. S_j is a L_{i_j-1}-L_{i_j+w} separator.

The sequence $\mathcal{C}_\mathcal{S} = (G_0, \ldots, G_q)$ with

$$G_j := G[(\bigcup_{\ell=i_j}^{i_{j+1}+(w-1)} L_\ell) - (S_j \cup S_{j+1})], \quad j = 0, \ldots, q,$$

is called the *sequence of graph chunks* obtained by $\mathcal{S}$.

With this definition at hand, we can state the key result needed to establish the algorithms that will be presented in Section 7.6.2. The proof techniques applied show some similarity to Baker [32].

Proposition 7.79 *Let $(G = (V, E), \phi)$ be a plane graph that admits a layerwise separation of width w and size dk. Then, for every $\psi \in \mathbb{R}_{\geq 0}$, there exists a partial layerwise separation $\mathcal{S}(\psi)$ of width w such that*

1. $\sum_{S \in \mathcal{S}(\psi)} |S| \leq \psi\sqrt{dk}$ and[13]

[12]Again, by default, we set $S_i := \emptyset$ for $i < 1$ and $i > q$.

[13]Taking $\sum$ instead of max here is a proposal of Kanj and Perkovič, compare [15, 252].

2. $\operatorname{out}(H) \le \frac{\sqrt{dk}}{\psi} + w$ *for each graph chunk H in $\mathcal{C}_{\mathcal{S}(\psi)}$.*

Moreover, there is an algorithm with running time $\mathcal{O}(\sqrt{k}n)$ which, for a given ψ,

- *recognizes whether (G, ϕ) admits a layerwise separation of width w and size dk and, if so, computes $\mathcal{S}(\psi)$;*
- *computes a partial layerwise separation of width w that fulfills the conditions above.*

Proof. For $m = 1, \dots, w$, consider the integer sequences $I_m = (m + jw)_{j=0}^{\lfloor r/w \rfloor - 1}$ and the corresponding sequences of separators $\mathcal{S}_m = (S_i)_{i \in I_m}$. Note that each $\mathcal{S}_m$ is a sequence of pairwise disjoint separators. Since $(S_1, \dots, S_r)$ is a layerwise separation of size dk, this implies that there exists a $1 \le m' \le w$ with

$$\sum_{i \in I_{m'}} |S_i| \le \frac{dk}{w}. \tag{7.4}$$

For a given ψ, let $s := \psi\sqrt{dk}$. Define $\mathcal{S}(\psi)$ to be the subsequence of $\mathcal{S}_{m'}$ such that $|S| \le s$ for all $S \in \mathcal{S}(\psi)$, and $|S| > s$ for all $S \in \mathcal{S}_{m'} - \mathcal{S}(\psi)$. This yields condition 1. As to condition 2, suppose that $\mathcal{S}(\psi) = (S_{i_1}, \dots, S_{i_q})$. How many layers are two separators S_{i_j} and $S_{i_{j+1}}$ apart? Herefore, note that the number of separators in $\mathcal{S}_{m'}$ that appear between S_{i_j} and $S_{i_{j+1}}$ is $(i_{j+1} - i_j)/w$. Since all of these separators have size greater than s, their number has to be bounded by dk/ws, see Equation (7.4). Therefore, we get $i_{j+1} - i_j \le \sqrt{dk}/\psi$ for all $j = 1, \dots, q-1$. Hence, the chunks $G[(\bigcup_{\ell=i_j}^{i_{j+1}+w-1} L_\ell) - (S_{i_j} \cup S_{i_{j+1}})]$ have outerplanarity at most $\sqrt{dk}/\psi + w$.

The algorithm that computes a partial layerwise separation $\tilde{\mathcal{S}}$ proceeds as follows: For given ψ, compute $s := \psi\sqrt{dk}$. Then, for $j = 1, \dots, r - w$, one checks whether the graph $\tilde{G}_j(v_s, v_t)$ admits a v_s-v_t–separator $\tilde{S}_j$ of size at most s. Here, $\tilde{G}_j(v_s, v_t)$ is the graph $G[\bigcup_{\ell=j}^{j+(w-1)} L_\ell]$ with two further vertices v_s and v_t and edges from v_s to all vertices in L_j and from v_t to all vertices in L_{j+w-1}. The separator $\tilde{S}_j$ can be computed in time $\mathcal{O}(s \cdot n)$ using techniques based on maximum flow (see [250] for details).

Let $\tilde{\mathcal{S}} = (\tilde{S}_1, \dots, \tilde{S}_q)$ be the sequence of all separators of size at most s found in this manner.[14] Suppose that $\tilde{S}_j \subseteq \bigcup_{\ell=i_j}^{i_j+(w-1)} L_\ell$ for some indices $1 \le i_1 < \ldots < i_q \le r$. Note that, by the arguments given above, no two

[14]Possibly, the separators $\tilde{S}_j$ in $\tilde{\mathcal{S}}$ found by the algorithm may differ from the separators in $\mathcal{S}(\psi)$.

such separators can be more than $\sqrt{dk}/\psi$ layers apart. Hence, if there was a j_0 such that $i_{j_0+1} - i_{j_0} > \sqrt{dk}/\psi$, the algorithms exits and returns "no." Otherwise, $\tilde{\mathcal{S}}$ is a partial layerwise separation of width w. ∎

In what follows, the positive real number ψ of Proposition 7.79 is also called *trade-off parameter.* This is because it allows us to optimize the trade-off between outerplanarity and separator size.

Proposition 7.79 will help construct a tree decomposition of treewidth $\mathrm{tw}(G) = \mathcal{O}(\sqrt{k})$, assuming that a layerwise separation of some constant width and size dk exists. Hence, for graph problems fulfilling this assumption and, moreover, allowing a $\sigma^{\mathrm{tw}(G)}n$ time algorithm for constant σ when the graph is given together with a tree decomposition, we obtain a solving algorithm with running time $c^{\sqrt{k}}n$. This aspect will be outlined in Subsection 7.6.2.

7.6.2 Phase 2: Algorithms on layerwisely separated graphs

After Phase 1, we are left with a set of disjoint (layerwise) separators of size $\mathcal{O}(\sqrt{k})$ separating the graph in components, each of which having outerplanarity bounded by $\mathcal{O}(\sqrt{k})$. We now present two different ways how to obtain, in a second phase, a $c^{\sqrt{k}}$-algorithm that makes use of this separation. In both cases, there is the trade-off parameter ψ from Proposition 7.79 that can be used to optimize the running time of the resulting algorithms.

Using tree decompositions

We use the concept of tree decompositions as described above. We show how the existence of a layerwise separation of small size helps to constructively obtain a tree decomposition of small width.

Theorem 7.80 *Let (G, ϕ) be a plane graph that admits a layerwise separation of width w and size dk. Then, we have* $\mathrm{tw}(G) \leq 2\sqrt{3dk} + (3w - 1)$. *Such a tree decomposition can be computed in time $\mathcal{O}(k^{3/2}n)$.*

Proof. By Proposition 7.79, for each $\psi \in \mathbb{R}_{\geq 0}$, there exists a partial layerwise separation $\mathcal{S}(\psi) = (S_1, \ldots, S_q)$ of width w with corresponding graph chunks $\mathcal{C}_{\mathcal{S}(\psi)} = (G_0, \ldots, G_q)$, such that $\sum_{S \in \mathcal{S}(\psi)} |S| \leq \psi\sqrt{dk}$ and $\mathrm{out}(G_i) \leq \frac{\sqrt{dk}}{\psi} + w$ for all $i = 0, \ldots, q$. The algorithm that constructs a tree decomposition $\mathfrak{X}_\psi$ is given as follows:

- Construct a tree decomposition $\mathcal{X}_i$ of width at most $3\,\mathrm{out}(G_i) - 1$ for each of the graphs G_i (using the algorithm from Theorem 7.23).

- Add S_i and S_{i+1} to every bag in $\mathcal{X}_i$ ($i = 0, \dots, q$).
- Let T_i be the tree of $\mathcal{X}_i$. Then, successively add an arbitrary connection between the trees T_i and T_{i+1} in order to obtain a tree T.

Using Proposition 7.35, we see that the tree T, together with the constructed bags, gives a tree decomposition of G. Clearly, its width $\mathrm{tw}(\mathfrak{X}_\psi)$ is upper-bounded by

$$\begin{aligned}\mathrm{tw}(\mathfrak{X}_\psi) &\leq \sum_{S \in \mathcal{S}(\psi)} |S| + \max_{i=0,\dots,q} \mathrm{tw}(G_i) \\ &\leq \sum_{S \in \mathcal{S}(\psi)} |S| + 3(\max_{i=0,\dots,q} \mathrm{out}(G_i)) - 1 \\ &\leq (\psi + 3/\psi)\sqrt{dk} + (3w - 1).\end{aligned}$$

This upper bound is minimized for $\psi = \sqrt{3}$. ∎

By [20, Proposition 4.5], a graph G that has no K_h-minor has treewidth bounded by $h^{3/2}\sqrt{n}$. In particular, this implies that a planar graph has treewidth bounded by $11.2\sqrt{n}$. In the case of the existence of linear problem kernels for a given graph problem $\mathcal{G}$, this method might be used in order to obtain $c^{\sqrt{k}}$-algorithms. From our results, we can derive upper bounds of the treewidth of a planar graph in terms of several graph specific numbers. As the reader may verify, these problem-specific treewidth bounds tend to outperform the numbers obtainable via [20, Proposition 4.5]. For example, Theorem 7.80 and Example 7.75 imply the following inequalities for a planar graph G, relating the treewidth with the vertex cover and dominating set number:

$$\begin{aligned}\mathrm{tw}(G) &\leq 2\sqrt{6\,vc(G)} + 5, \quad \text{and} \\ \mathrm{tw}(G) &\leq 6\sqrt{17\,ds(G)} + 8.\end{aligned}$$

Note that for general graphs, no relation of the form

$$\mathrm{tw}(G) \leq f(ds(G)) \tag{7.5}$$

(for any function f) holds; consider, e.g., the clique K_n with n vertices, where $\mathrm{tw}(K_n) = n - 1$, but $ds(K_n) = 1$. Fomin and Thilikos have recently shown [191] that Eq. (7.5) holds for a graph G iff G has bounded *local treewidth.* For VC, only the linear relation

$$\mathrm{tw}(G) \leq vc(G)$$

can be easily shown: Note that the complement of a vertex cover set C forms an independent set I in G. Hence, we can easily construct even a path decomposition by choosing $|I|$ bags and making each bag consist of all vertices of C and exactly one element from I each time. This estimate is sharp (which becomes clear by, again, considering the graph K_n, where $vc(K_n) = n - 1$).

Theorem 7.80 yields a $c^{\sqrt{k}}$-algorithm for certain graph problems:

Theorem 7.81 *Let $\mathcal{G}$ be a parameterized problem for planar graphs. Suppose that*

1. *$\mathcal{G}$ has the LSP of width w and size-factor d, and*

2. *there exists a time $\sigma^\ell n$ algorithm that decides $(G, k) \in \mathcal{G}$, if G is given together with a tree decomposition of width ℓ.*

Then, there is an algorithm to decide $(G, k) \in \mathcal{G}$ in time $\mathcal{O}(\sigma^{3w-1} \cdot 2^{\theta_1(\sigma,d)\sqrt{k}} n)$, where $\theta_1(\sigma, d) = 2(\log \sigma)\sqrt{3d}$.

Proof. Given an instance (G, k), in linear time we can compute some planar embedding ϕ of G (for details see [94]). In time $\mathcal{O}(\sqrt{k}n)$ (see Proposition 7.79), we can check whether the plane graph (G, ϕ) admits a layerwise separation of width w and size dk.

If so, the algorithm of Theorem 7.80 computes a tree decomposition of width at most $2\sqrt{3dk} + (3w - 1)$, and we can decide $(G, k) \in \mathcal{G}$ by using the given tree decomposition algorithm in time $\mathcal{O}(\sigma^{2\sqrt{3dk}+(3w-1)} n)$.

If (G, ϕ) does not admit such a layerwise separation, we know that $(G, k) \notin \mathcal{G}$, by definition of the LSP. ∎

Example 7.82 Going back to our running examples, it is well-known that VERTEX COVER and INDEPENDENT SETadmit such a tree decomposition based algorithm with $\sigma = 2$ and, in the case of DOMINATING SET, with $\sigma = 4$, as detailed above.

1. For PLANAR VERTEX COVER, by Example 7.77.1 Theorem 7.81 guarantees an $\mathcal{O}(2^{2\sqrt{6k}} n)$ algorithm for this problem.

2. For PLANAR INDEPENDENT SET, Example 7.77.2 and Theorem 7.81 yield an $\mathcal{O}(2^{4\sqrt{3k}} n)$ algorithm.

3. By Example 7.75.2, Theorem 7.81 improves on the result from [8] (that are reproduced above; there, we got an $\mathcal{O}(4^{6\sqrt{34k}} n) \approx \mathcal{O}(2^{69.98\sqrt{k}} n)$ algorithm), namely, getting an $\mathcal{O}(4^{6\sqrt{17k}} n) \approx \mathcal{O}(2^{49.48\sqrt{k}} n)$ algorithm for PLANAR DOMINATING SET.

In this subsection, we have seen that, for plane graphs, the notion of the LSP gives us a sufficient condition to upperbound the treewidth of the YES-instance graphs of a problem. Moreover, this property led to *fast* computations of the corresponding tree decompositions. All in all, we came up with algorithms of running time $\mathcal{O}(c^{\sqrt{k}}n)$ for a wide class of problems.

The next subsection aims to show similar results in a different context.

Using bounded outerplanarity

We now turn our attention to certain parameterized graph problems for which we know that a solving algorithm of linear running time on the class of graphs of bounded outerplanarity exists. This issue was addressed in [32]; several examples can be found therein. In this subsection, examine how, in this context, the notion of select&verify problems and the LSP will lead to $c^{\sqrt{k}}$-algorithms. Since this will be a purely separator-based approach, we will restrict ourselves to parameterized graph problems that can be solved easily on separated graphs. We will introduce the notion of weakly glueable select&verify problems in a first paragraph and present the design of $c^{\sqrt{k}}$-algorithms for these problems afterwards (see Paragraph 7.6.2).

Weakly glueable graph problems
The notion of weak glueability that is introduced here is much related to the notion of glueability we introduced in the previous section. We need one more auxiliary notion for colorings.

Definition 7.83 For two 0-1-colorings $\chi_1, \chi_2 : V \to \{0, 1, \#\}$ with $\chi_1^{-1}(\{0,1\}) \cap \chi_2^{-1}(\{0,1\}) = \emptyset$, the *sum* $\chi_1 + \chi_2$ is defined by

$$(\chi_1 + \chi_2)(v) = \begin{cases} \chi_1(v) & \text{if } \chi_1(v) \neq \#, \\ \chi_2(v) & \text{if } \chi_2(v) \neq \#, \\ \# & \text{otherwise.} \end{cases}$$

Definition 7.84 A select&verify problem $\mathcal{G}$ given by $(P_\cdot, \text{opt})$ is *weakly glueable with λ colors* if there exist

- a color set $C := C_0 + C_1 + \{\#\}$ with $|C_0 + C_1| = \lambda$, and
- a polynomial time computable function $h : (\mathbb{R}_{\geq 0} \cup \{\pm\infty\})^3 \to \mathbb{R}_{\geq 0} \cup \{\pm\infty\}$;

and, for every n-vertex graph $G = (V, E)$ and subgraphs $G^{\text{sel}}, G^{\text{ver}} \subseteq G$ with a separation (A_1, S, A_2) of G^{ver}, we find, for each coloring $\chi^{\text{int}} : S \to C_0 \uplus C_1$,

- subgraphs $G(A_i, \chi^{\text{int}})$ of G^{ver} with $G^{\text{ver}}[A_i] \subseteq G(A_i, \chi^{\text{int}}) \subseteq G^{\text{ver}}[\delta A_i]$ for $i = 1, 2$,
- subgraphs $G(S, \chi^{\text{int}})$ of G^{ver} with $G(S, \chi^{\text{int}}) \subseteq G^{\text{ver}}[S]$

such that, for each 0-1-coloring $\chi^{\text{ext}} : V \to \{0, 1, \#\}$ with $\chi^{\text{ext}} |_S \equiv \#$, we have

$$\begin{aligned} &\text{opt}_{\substack{\vec{x} \in \{0,1\}^n \\ \vec{x} \sim \chi^{\text{ext}}}} P_{G^{\text{ver}}}(\vec{x} \mid G^{\text{sel}}) \\ &= \text{opt}_{\chi^{\text{int}} : S \to C_0 \uplus C_1} h\big(\text{Eval}_{A_1}(\chi^{\text{int}}), \text{Eval}_S(\chi^{\text{int}}), \text{Eval}_{A_2}(\chi^{\text{int}})\big). \end{aligned} \tag{7.6}$$

Here, $\text{Eval}_X(\cdot)$ for $X \in \{A_1, S, A_2\}$ is of the form

$$\text{Eval}_X(\chi^{\text{int}}) \quad = \quad \text{opt}_{\substack{\vec{x} \in \{0,1\}^n \\ \vec{x} \sim (\chi^{\text{ext}} + \hat{\chi^{\text{int}}})}} P_{G(X, \chi^{\text{int}})}(\vec{x} \mid G[X] \cap G^{\text{sel}}).$$

Example 7.85 We give some examples of weakly glueable problems, where —for the time being—we restrict ourselves to the case where $G^{\text{ver}} = G^{\text{sel}} = G$. The subtlety of allowing different subgraphs G^{ver}, G^{sel} in the definition above is due to technical reasons that become clear later. All examples generalize in a straight-forward way to this case.

1. VERTEX COVER is weakly glueable with $\lambda = 2$ colors. We use the color sets $C_i := \{i\}$ for $i = 0, 1$. The function h is $h(x, y, z) = x + y + z$. The subgraphs $G(X, \chi^{\text{int}})$ for $X \in \{A_1, S, A_2\}$ and $\chi^{\text{int}} : S \to C_0 \uplus C_1$ are

$$\begin{aligned} G(A_i, \chi^{\text{int}}) &:= G[A_i \cup {\chi^{\text{int}}}^{-1}(0)] \quad \text{for } i = 1, 2, \quad \text{and} \\ G(S, \chi^{\text{int}}) &:= G[S]. \end{aligned}$$

   The subroutine $\text{Eval}_S(\chi^{\text{int}})$ checks if the coloring χ^{int} yields a vertex cover on $G[S]$ and the subroutines $\text{Eval}_{A_i}(\chi^{\text{int}})$ compute the minimum size vertex cover on $G[A_i]$ under the constraint that all neighbors in A_i of a vertex in ${\chi^{\text{int}}}^{-1}(0)$ are covered. Obviously, Eq. (7.6) is thus satisfied.

2. Similarly, INDEPENDENT SET is weakly glueable with 2 colors.

3. DOMINATING SET is weakly glueable with $\lambda = 4$ colors, using $C_0 := \{0_{A_1}, 0_{A_2}, 0_S\}$ and $C_1 := \{1\}$. The semantics of these colors is as follows. Assigning the color 0_X, for $X \in \{A_1, S, A_2\}$, to vertices in a separation (A_1, S, A_2) means that the vertex is not in the dominating set and will be dominated by a vertex from X. Color 1 means that the vertex

belongs to the dominating set. We set $h(x,y,z) = x+y+z$. For a given coloring $\chi^{\mathrm{int}} : S \to C_0 \uplus C_1$, we define

$$\begin{aligned} G(A_i, \chi^{\mathrm{int}}) &:= G[A_i \cup {\chi^{\mathrm{int}}}^{-1}(\{1, 0_{A_i}\})] \\ G(S, \chi^{\mathrm{int}}) &:= G[{\chi^{\mathrm{int}}}^{-1}(\{1, 0_S\})]. \end{aligned}$$

In this way, color information is passed to the subproblems. $\mathrm{Eval}_S(\chi^{\mathrm{int}})$ checks whether the assignments of the color 0_S are correct (i.e., whether all vertices assigned 0_S are dominated by a vertex from S). Also, Eval_{A_i} returns the size of a minimum dominating set in A_i under the constraint that some vertices in δA_i still need to be dominated (namely, the vertices in ${\chi^{\mathrm{int}}}^{-1}(0_{A_i})$) and some vertices in δA_i can already be assumed to be in the dominating set (namely, the vertices in ${\chi^{\mathrm{int}}}^{-1}(1)$). With these settings, Eq. (7.6) is satisfied.

We want to mention in passing that—besides the problems given here—many more select&verify problems are weakly glueable. In particular this is true for the weighted versions and variations of the above mentioned problems.

The algorithm

Similar to Theorem 7.81, which is based on tree decompositions, we construct a partial layerwise separation $\mathcal{S}(\psi)$ with optimally adapted trade-off parameter ψ to guarantee an efficient dynamic programming algorithm. However, for our purposes here, we need to be able to deal with "precolored" graphs.

Definition 7.86 Let $\mathcal{G}$ be a select&verify graph problem defined by $(P_\cdot, \mathrm{opt})$. The problem CONSTRAINT $\mathcal{G}$ then is to determine, for an n-vertex graph $G = (V,E)$, two subgraphs $G^{\mathrm{sel}}, G^{\mathrm{ver}} \subseteq G$, and a given 0-1-coloring $\chi : V \to \{0,1,\#\}$, the value

$$\mathrm{opt}_{\substack{\vec{x}\in\{0,1\}^n \\ \vec{x}\sim\chi}} P_{G^{\mathrm{ver}}}(\vec{x} \mid G^{\mathrm{sel}}).$$

Theorem 7.87 *Let $\mathcal{G}$ be a select&verify problem for planar graphs. Suppose that*

1. *$\mathcal{G}$ has the LSP of width w and size-factor d,*
2. *$\mathcal{G}$ is weakly glueable with λ colors, and*
3. *there exists an algorithm that solves the problem* CONSTRAINT *$\mathcal{G}$ for a given graph G in time $\tau^{\mathrm{out}(G)}n$.*

Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$ in time $\mathcal{O}(\tau^w \cdot 2^{\theta_2(\lambda,\tau,d)\sqrt{k}}n)$, where $\theta_2(\lambda,\tau,d) = 2\sqrt{d\log(\lambda)\log(\tau)}$.

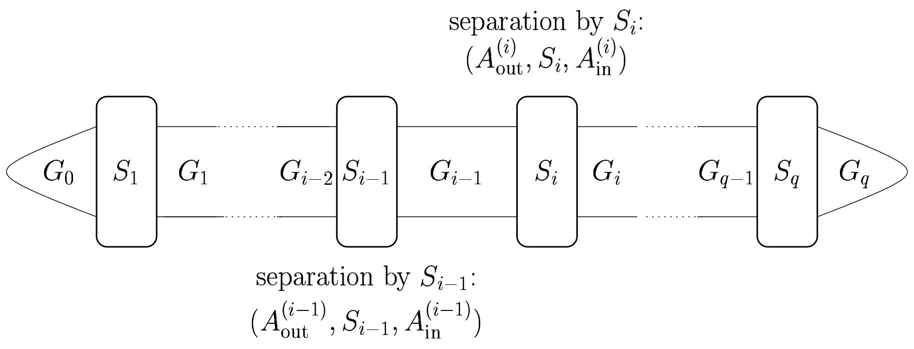


Figure 7.17: Dynamic programming on layerwisely separated graph.

Proof. Let us first sketch the overall structure of the algorithm.

1. Compute some planar embedding ϕ of G, and find a "suitable" partial layerwise separation $(S_1, \ldots, S_q)$ for the plane graph (G, ϕ). A coarse sketch of the such obtained graph structure is depicted in Figure 7.17.

2. By using dynamic programming techniques, an optimal solution is found by sweeping over the graph from left to right, as illustrated in Figure 7.17. More detailed, we do the following:

   (a) For all possible "colorings" of S_1, find an optimal solution of CONSTRAINT $\mathcal{G}$ on G_0 (plus suitably chosen precolored vertices from S_1); store the obtained optima in a (large) table belonging to S_1.

   (b) For $j := 2$ to q do:

   - For all possible "colorings" of S_{j-1} and of S_j, find an optimal solution of CONSTRAINT $\mathcal{G}$ on G_{j-1} (plus suitably chosen precolored vertices from S_{j-1} as well as of S_j).
   - Store the obtained optima (for the subgraph of G with vertices from G_0 through G_{j-1} and S_1 through S_j) in a table belonging to S_j.
   - (For reasons of space efficiency, you might now forget about the table belonging to S_{j-1}.)

   (c) For all possible "colorings" of S_q, find an optimal solution of CONSTRAINT $\mathcal{G}$ on G_q (plus suitably chosen precolored vertices from S_q); store the obtained optima in a (large) table belonging to S_q.

   (d) From the table pertaining to S_q, the desired optimum is found.

We are now giving formal details of the sketched algorithms, thereby proving its correctness. Suppose $\mathcal{G}$ is defined by $(P_\cdot, \mathrm{opt})$.

Step 1: Given an instance (G, k), in linear time we can compute some planar embedding ϕ of G (see [94]). Compute a partial layerwise separation $\mathcal{S}(\psi)$ (ψ will be determined later) for (G, ϕ), using Proposition 7.79 and the assumption 1 (LSP). Suppose $\mathcal{S}(\psi) = (S_1, \dots, S_q)$ and let $\mathcal{C}_{\mathcal{S}(\psi)} = (G_0, \dots, G_q)$ denote the corresponding graph-chunks cut out by $\mathcal{S}(\psi)$.

For every separator S_i, we get a separation of the form

$$(A^{(i)}_{\mathrm{out}}, S_i, A^{(i)}_{\mathrm{in}}),$$

where $A^{(i)}_{\mathrm{out}}$, and $A^{(i)}_{\mathrm{in}}$, respectively, are the vertices of the graph chunks of lower order layers, and higher order layers, respectively. By default, we let $S_0 = S_{q+1} = \emptyset$ such that the corresponding separations are $(\emptyset, S_0, V)$ and $(V, S_{q+1}, \emptyset)$, respectively. The layerwise separation and the separations $(A^{(i)}_{\mathrm{out}}, S_i, A^{(i)}_{\mathrm{in}})$ are illustrated in Figure 7.17.

Step 2: Determine the value

$$\mathrm{opt}_{\vec{x} \in \{0,1\}^n} P_G(\vec{x}).$$

This can be done by a dynamic programming approach that makes use of the weak glueability of $\mathcal{G}$ as follows.

We successively compute, for $i = 1, \dots, q+1$, the values

$$M^{(i)}(\mu^{(i)}) \quad := \quad \mathrm{opt}_{\vec{x} \sim \widehat{\mu^{(i)}}} P_{H_i^{\mathrm{ver}}(\mu^{(i)})}(\vec{x} \mid H_i^{\mathrm{sel}}) \tag{7.7}$$

for all C_0-C_1-colorings $\mu^{(i)} : S_i \to C_0 + C_1$, where

$$H_i^{\mathrm{ver}}(\mu^{(i)}) \; := \; G(A^{(i)}_{\mathrm{out}}, \mu^{(i)}) \quad \text{and} \quad H_i^{\mathrm{sel}} \; := \; G[A^{(i)}_{\mathrm{out}}].$$

Note that we have

$$\mathrm{opt}_{\vec{x} \in \{0,1\}^n} P_G(\vec{x}) = M^{(q+1)}(\mu),$$

for the empty map $\mu : S_{q+1} = \emptyset \to C_0 + C_1$, because $H^{\mathrm{sel}}_{q+1} = G[A^{(q+1)}_{\mathrm{out}}] = G[V] = G$ and $H^{\mathrm{ver}}_{q+1}(\mu) = G$ (since $H^{\mathrm{sel}}_{q+1} \subseteq H^{\mathrm{ver}}_{q+1}(\mu)$).

The computation of $M^{(i)}(\mu^{(i)})$ as defined in (7.7) can be done iteratively. To do so, note that $H_i^{\mathrm{ver}}(\mu^{(i)})$ is separated by S_{i-1} in

$$(B^{(i-1)}_{\mathrm{out}}, S_{i-1}, B^{(i-1)}_{\mathrm{in}}),$$

where $B_{\text{out}}^{(i-1)} = A_{\text{out}}^{(i-1)}$ and $B_{\text{in}}^{(i-1)} = A_{\text{in}}^{(i-1)} \cap V(H_i^{\text{ver}}(\mu^{(i)}))$. Hence, by definition of weak glueability we have

$$\begin{aligned} M^{(i)}(\mu^{(i)}) &= \text{opt}_{\mu^{(i-1)}:S_{i-1}\to C_0+C_1} \\ &\qquad h(\text{Eval}_{B_{\text{out}}^{(i-1)}}(\mu^{(i-1)}), \text{Eval}_{S_{i-1}}(\mu^{(i-1)}), \text{Eval}_{B_{\text{in}}^{(i-1)}}(\mu^{(i-1)})) \end{aligned} \tag{7.8}$$

where

$$\text{Eval}_X(\mu^{(i-1)}) = \text{opt}_{\vec{x}\sim(\widehat{\mu^{(i-1)}}+\widehat{\mu^{(i)}})} P_{G(X,\mu^{(i-1)})}(\vec{x} \mid G[X] \cap H_i^{\text{sel}}). \tag{7.9}$$

for $X \in \{B_{\text{out}}^{(i-1)}, S_{i-1}, B_{\text{in}}^{(i-1)}\}$. Here, recall that $\widehat{\mu^{(i)}}$ denotes the 0-1-coloring corresponding to $\mu^{(i)}$. In particular, for the different choices of X, we get the following.

- For $X = B_{\text{out}}^{(i-1)}$, equation (7.9) becomes

$$\begin{aligned} \text{Eval}_{B_{\text{out}}^{(i-1)}}(\mu^{(i-1)}) &= \text{opt}_{\vec{x}\sim(\widehat{\mu^{(i-1)}}+\widehat{\mu^{(i)}})} P_{H_{i-1}^{\text{ver}}(\mu^{(i-1)})}(\vec{x} \mid H_{i-1}^{\text{sel}} \cap H_i^{\text{sel}}) \\ &= M^{(i-1)}(\mu^{(i-1)}), \end{aligned} \tag{7.10}$$

  where the last equation holds, since $H_{i-1}^{\text{sel}} \subseteq H_i^{\text{sel}}$ and since $\widehat{\mu^{(i)}} \equiv \#$ restricted to $G(B_{\text{out}}^{(i-1)}, \mu^{(i-1)}) = H_{i-1}^{\text{ver}}(\mu^{(i-1)})$.

  Hence, the value of $\text{Eval}_{B_{\text{out}}^{(i-1)}}(\mu^{(i-1)})$ is given by the previous step of the iteration.

- For $X = S_{i-1}$, equation (7.9) becomes

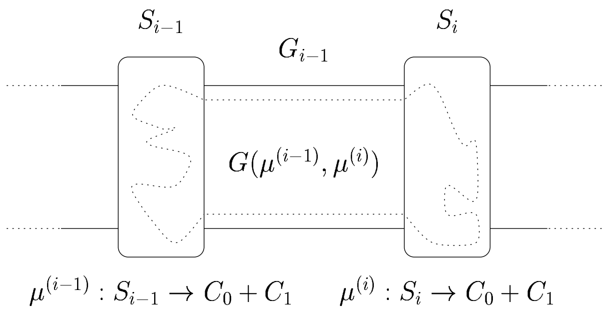
$$\begin{aligned} \text{Eval}_{S_{i-1}}(\mu^{(i-1)}) &= \text{opt}_{\vec{x}\sim\widehat{\mu^{(i-1)}}} P_{G(S_{i-1},\mu^{(i-1)})}(\vec{x} \mid G[S_{i-1}]) \\ &= P_{G(S_{i-1},\mu^{(i-1)})}(\vec{x}_{\widehat{\mu^{(i-1)}}} \mid G[S_{i-1}]),, \end{aligned} \tag{7.11}$$

  where $\vec{x}_{\widehat{\mu^{(i-1)}}} \in \{0,1\}^n$ is an arbitrary vector such that $\vec{x}_{\widehat{\mu^{(i-1)}}} \sim \widehat{\mu^{(i-1)}}$. The first equation above holds, since $G(S_{i-1}, \mu^{(i-1)}) \subseteq G[S_{i-1}] \subseteq H_i^{\text{sel}}$ and $\widehat{\mu^{(i)}} \equiv \#$ restricted to $G(S_{i-1}, \mu^{(i-1)})$. The second equation is true since the 0-1-coloring $\widehat{\mu^{(i-1)}}$ assigns color 0 or color 1 to all vertices in S_{i-1}, and since $G(S_{i-1}, \mu^{(i-1)}) \subseteq G[S_{i-1}]$.

  Hence, the value $\text{Eval}_{S_{i-1}}(\mu^{(i-1)})$ can be computed by a simple evaluation of the function P for the given vector $\vec{x}_{\widehat{\mu^{(i-1)}}}$.

- For $X = B_{\text{in}}^{(i-1)}$, equation (7.9) becomes

$$\begin{aligned} \text{Eval}_{B_{\text{in}}^{(i-1)}}(\mu^{(i-1)}) &= \text{opt}_{\vec{x}\sim(\widehat{\mu^{(i-1)}}+\widehat{\mu^{(i)}})} P_{G(B_{\text{in}}^{(i-1)},\mu^{(i-1)})}(\vec{x} \mid G[B_{\text{in}}^{(i-1)}] \cap H_i^{\text{sel}}) \\ &= \text{opt}_{\vec{x}\sim(\widehat{\mu^{(i-1)}}+\widehat{\mu^{(i)}})} P_{G(\mu^{(i)},\mu^{(i-1)})}(\vec{x} \mid G_{i-1}), \end{aligned} \tag{7.12}$$



S_{i-1}
S_i
G_{i-1}
$G(\mu^{(i-1)},\mu^{(i)})$
$\mu^{(i-1)}: S_{i-1} \to C_0 + C_1$
$\mu^{(i)}: S_i \to C_0 + C_1$

Figure 7.18: The graph $G(\mu^{(i)},\mu^{(i-1)})$.

where $G(\mu^{(i)},\mu^{(i-1)}) := G(A_{\text{in}}^{(i-1)},\mu^{(i-1)}) \cap G(A_{\text{out}}^{(i)},\mu^{(i)})$. This graph is illustrated in Figure 7.18. In the evaluation above we used that $G[B_{\text{in}}^{(i-1)}] \cap H_i^{\text{sel}} = G_{i-1}$ and that $G(B_{\text{in}}^{(i-1)},\mu^{(i-1)}) = G(\mu^{(i)},\mu^{(i-1)})$. Hence, the value $\text{Eval}_{B_{\text{in}}^{(i-1)}}(\mu^{(i-1)})$ can be computed using the $\tau^{\text{out}(G)}$ time algorithm for CONSTRAINT $\mathcal{G}$.

Hence, plugging formulas (7.10), (7.11), and (7.12) in expression (7.8), we obtain

$$M^{(i)}(\mu^{(i)}) = \text{opt}_{\mu^{(i-1)}:S_{i-1}\to C_0+C_1}\ h\left(\begin{array}{c} M^{(i-1)}(\mu^{(i-1)}) \\ P_{G(S_{i-1},\mu^{(i-1)})}(\vec{x}_{\widehat{\mu^{(i-1)}}} \mid G[S_{i-1}]) \\ \text{opt}_{\vec{x}\sim\widehat{\mu^{(i-1)}+\mu^{(i)}}}\ P_{G(\mu^{(i)},\mu^{(i-1)})}(\vec{x} \mid G_{i-1}) \end{array}\right). \tag{7.13}$$

This evaluation is done successively for all $i = 1, \ldots, q+1$. By induction, one sees that

$$\text{opt}_{\vec{x}\in\{0,1\}^n}\, P_G(\vec{x}) = M^{(q+1)}(\mu)$$

can be computed in this way.

Computation time: For fixed coloring $\mu^{(i)}$, computing $M^{(i)}(\mu^{(i)})$ according to equation (7.13) costs time

$$\lambda^{|S_{i-1}|} \cdot \tau^{\text{out}(G[S_{i-1}\cup V_{G_{i-1}}\cup S_i])}.$$

The first factor reflects the cost of computing the opt over all $\mu^{(i-1)} : S_{i-1} \to C_0{+}C_1$. The second factor arises by the evaluations on the graphs $G(S_{i-1},\mu^{(i-1)})$, $G(\mu^{(i-1)},\mu^{(i)}) \subseteq G[S_{i-1} \cup V(G_{i-1}) \cup S_i]$, where we use assumption 3 of the theorem. Thus, the running time for evaluating $M^{(i)}(\mu^{(i)})$ for all $\mu^{(i)} : S_i \to C_0 + C_1$ is bounded by

$$\lambda^{|S_i|} \cdot \lambda^{|S_{i-1}|} \cdot \tau^{\text{out}(G[S_{i-1}\cup V_{G_{i-1}}\cup S_i])}.$$

Hence, the total running time of the algorithm is bounded by $2^{\theta(\psi)}n$, where

$$\begin{aligned}\theta(\psi) &\leq \log(\lambda)\sum_{i=1,\ldots,q}|S_i| \quad + \quad \log(\tau)\max_{i=0,\ldots,q}\operatorname{out}(G[S_i \cup V_{G_i} \cup S_{i+1}])\\ &\leq \log(\lambda)\psi\sqrt{dk} \quad + \quad \log(\tau)(\frac{\sqrt{dk}}{\psi}+w)\\ &= (\log(\lambda)\psi + \frac{\log\tau}{\psi})\sqrt{dk} \quad + \quad \log(\tau)w\end{aligned}$$

This upper bound is minimized for $\psi = \sqrt{\log(\tau)/\log(\lambda)}$, which gives us the claimed value $\theta_2(\lambda,\tau,d) = 2\sqrt{d\log(\lambda)\log(\tau)}$ and the constant τ^w. ∎

It remains to say for which problems there exists a solving algorithm of the problem CONSTRAINT $\mathcal{G}$ for a given graph G in time $\tau^{\operatorname{out}(G)}n$.

In the case of PLANAR VERTEX COVER, as well as in the case of PLANAR INDEPENDENT SET, it is quite simple to handle a "precoloring" $\chi^{\text{ext}} : V \to \{0,1,\#\}$ for a graph $G=(V,E)$.

More formally, given an admissible[15] coloring χ^{ext}, one likes to transform an instance (G,k) to an instance (G',k'), such that $(G,k)\in\mathcal{G}$ for some witnessing vector $\vec{x}$ with $\vec{x}\sim\chi^{\text{ext}}$ iff $(G',k')\in\mathcal{G}$ for some witnessing vector $\vec{x'}$ (without any constraint).

For PLANAR VERTEX COVER, this can, e.g., be achieved by the following transformation:

$$\begin{aligned}G' &= G[V-(C\cup N({\chi^{\text{ext}}}^{-1}(0)))] \quad \text{and}\\ k' &= k - |{\chi^{\text{ext}}}^{-1}(1)| - |\{v\in V-C \mid \exists u\in{\chi^{\text{ext}}}^{-1}(0)\cap N(v)\}|,\end{aligned}$$

here $C := {\chi^{\text{ext}}}^{-1}(\{0,1\})$ denotes the vertices that are already assigned a color. A vertex $v\in C$, which is—by the coloring χ^{ext}— assigned not to be in the vertex cover, i.e., $\chi^{\text{ext}}(v)=0$, forces its neighbors to be in a vertex cover. Hence, the set $N({\chi^{\text{ext}}}^{-1}(0))$ needs to be in any vertex cover (given by $\vec{x}$) that fulfills the constraint $\vec{x}\sim\chi^{\text{ext}}$. This justifies the definition of G'. The parameter k becomes smaller by the number of vertices which are already assigned to be in the vertex cover, i.e., ${\chi^{\text{ext}}}^{-1}(1)$, and the number of vertices that are forced to be in the vertex cover by χ^{ext}, i.e., by the number of vertices in $V-C$ that have a neighbor in ${\chi^{\text{ext}}}^{-1}(0)$. We can apply the non-constraint algorithm to (G',k'), with $\operatorname{out}(G')\leq\operatorname{out}(G)$. A similar observation helps deal with PLANAR INDEPENDENT SET.

[15]Here, *admissible* means that there exists a vector $\vec{x}\in\{0,1\}^n$ with $\vec{x}\sim\chi^{\text{ext}}$, such that $P_G^{\text{ver}}(\vec{x})=0$.

Example 7.88 1. For PLANAR VERTEX COVER, we have $d = 2$, $w = 1$ (see Example 7.77.1), $\lambda = 2$ (see Example 7.85.1), and $\tau = 8$ (see the result of Baker [32] which can be adapted to the constraint case by the considerations above) and, hence, the approach in Theorem 7.87 yields an $\mathcal{O}(2^{2\sqrt{6k}}n)$ time algorithm.

2. Similarly, Examples 7.77.2, and 7.85.2 give $d = 4$, $w = 1$ and $\lambda = 2$ for PLANAR INDEPENDENT SET. Since $\tau = 8$ (see [32] and our considerations for the constraint case), we obtain a $\mathcal{O}(2^{4\sqrt{3k}}n)$ time algorithm.

3. Kanj and Perkovič have recently shown [253] that CONSTRAINT DOMINATING SET can be solved in $\mathcal{O}(27^{\text{out}(G)}n)$ time, this way improving on Example 7.82. Together with Example 7.75.1, Theorem 7.87 gives an $\mathcal{O}(3^{6\sqrt{17k}}n) \approx \mathcal{O}(2^{39.21\sqrt{k}}n)$ time algorithm for PLANAR DOMINATING SET.

Which of the two approaches (presented in Subsections 7.6.2 and 7.6.2, respectively) for the algorithms on layerwisely separated graphs should be preferred? This question is discussed in details in [15]. The bottom-line is that it mainly depends whether there are particular outerplanarity-based algorithms that are superior to the treewidth-based algorithms (when using the $r \leq 3k$ bound described above that relates a graph of outerplanarity r with its treewidth k).

7.6.3 The approach of Fomin and Thilikos

In a very deep paper, Fomin and Thilikos [190] (see [193] for the report version) were able to prove the following result:

Theorem 7.89 *Every planar n-vertex graph has treewidth at most* $3.182\sqrt{n}$.

The drawback of that approach is currently that the way how to actually obtain a tree decomposition for a planar graph with n vertices is not very clear from [190], so that it is not that easy to implement.

However, observe that also the approach we detailed above (using treewidth instead of branchwidth), yields a result analogous to Thm. 7.89, although with somewhat weaker constants. More precisely, Theorem 7.80 can be used, since we can basically set $k = n$ and $d = w = 1$. With this observation, we obtain a bound of $2\sqrt{3n} \leq 3.4642\sqrt{n}$ on the treewidth for any planar n-vertex graph.

Theorem 7.89 immediately entails $\mathcal{O}^*(c^{\sqrt{k}})$-algorithms for many planar problems where we know of linear problem kernels. For example, we would get an $\mathcal{O}(2^{2\cdot 3.182\sqrt{67k}}) = \mathcal{O}^*(2^{52.092\sqrt{k}})$-algorithm for PLANAR DOMINATING SET.

7.7 Further improvements for PLANAR DOMINATING SET

First of all, let us summarize the preprocessing observations that we have made up to this point on PLANAR DOMINATING SET in Alg. 67. This algorithm prelude can be performed in linear time.

Algorithm 67 Algorithmic Prelude

Input(s): A plane graph G and parameter $k \in \mathbb{N}$.

```
Compute the ring decomposition $\mathcal{F}_G$.
Compute a minimum dominating set $D$ of $\mathcal{F}_G$ (see Lemma 7.30).
if $|D| > k$ then
  return NO
else
  continue
end if
```

As it is well-known, given some planar graph G, we could also build an embedding of G in the plane in linear time. The planar embedding algorithm could also be included as part of the prelude, if necessary.

After executing the prelude, algorithms for PLANAR DOMINATING SET proceed as follows. If G has a dominating set of size k, then Proposition 7.27 tells us that G is $\leq 3k$ outerplanar. So, assume that G is $\leq 3k$ outerplanar.

In the following, we will show how to obtain a sequence S—called a *layerwise separator*—of "small" separators $S_3, \dots, S_{r-2}$, where S_i contains vertices from L_{i-1}, L_i and L_{i+1} that separate layer L_{i-1} from L_{i+2}, such that

$$\parallel S \parallel = |S_3| + \cdots + |S_{r-2}| \leq h(k)$$

for some linear-bounded function h. This was also the basic trick in what we explained when referring to [8]. Once we have such a "small" layerwise separator we can essentially split G into disconnected components, each of which is $\mathcal{O}(\sqrt{k})$-outerplanar.

Notice that it must be the case that one of the families

$$F_i = \bigcup_{j=i \bmod 3, 3 \leq j \leq r-2} S_j,$$

$i = 1, 2, 3$ satisfies $|F_i| \leq h(k)/3$. Without loss of generality, assume that $|F_1| \leq h(k)/3$. Now, F_1 can be partitioned into $g(k)$ mutually disjoint subfamilies

$$F_1^m = \bigcup_{j_i = 1+m+ig(k)} S_{j_i},$$

$m = 0, \ldots, g(k)-1$, $i = 0, \ldots, \lfloor r/g(k) \rfloor$. At least one F_1^m satisfies $|F_1^m| \leq h(k)/(3g(k))$. Notice that the separator $S^m = F_1^m$ splits the graph into components that are at most $3g(k)+3$ outerplanar.

As detailed above, to find the minimum dominating set in G, it suffices to search through all possible three possible *domination choices* for each vertex v in the partial layerwise separator $S^m = (S_{j_0}, \ldots, S_{j_q})$. Schematically, we can think of the "rest" of the vertices of G being "inbetween" some S_{j_i} and $S_{j_{i+1}}$. The possible domination choices are "v is in the dominating set," "v is dominated by component to the left," or "v is dominated by a vertex $u \in S_m$ or a component to the right." Since $|S^m| \leq h(k)/3g(k)$, there are $3^{h(k)/3g(k)}$ possible domination choices for S^m. Using a setting of domination choices for S^m, it is possible to split the graph into $\leq 3g(k)+5$-outerplanar components that are precolored with the colors $\{\text{free}, \text{already dominated}, \text{not dominated}\}$. (These components may contain contain vertices from S^m.) Making use of Theorem 7.23, we can solve the "precolored" PLANAR DOMINATING SET problem on the remaining graph components in time $\mathcal{O}(4^{9g(k)})$ time. Kanj and Perkovič [253] described a dynamic programming algorithm for precolored PLANAR DOMINATING SET on r-outerplanar graphs running in time $\mathcal{O}(27^r)$. This allows us to compute the size of the minimum dominating set in time $\mathcal{O}(3^{9g(k)})$ for each precolored component. Using the notation introduced so far, we arrive at the following result.

Lemma 7.90 PLANAR DOMINATING SET *can be solved in time*

$$\mathcal{O}(h(k)/g(k) 3^{h(k)/(3g(k))+9g(k)} + n^3),$$

where k upperbounds the number of vertices in the dominating set.

More details are described in Algorithm 68.

Since $h(k)$, as determined in the next section, will be bounded by dk for some d, $g(k)$ would then be best chosen as $c\sqrt{k}$ such that $dk/(3c\sqrt{k}) = 9c\sqrt{k}$, i.e., $c = \sqrt{d}/(3\sqrt{3})$, because $3^{dk/(3c\sqrt{k})}$ is the asymptotics of the time spent to test all possible separator settings, and $3^{3*3c\sqrt{k}}$ is the time to solve the problem on the remaining graph chunks. Hence, $9c\sqrt{k} = 9\sqrt{dk}/(3\sqrt{3}) = \sqrt{3dk}$. This gives us an $\mathcal{O}(\sqrt{k}3^{2\sqrt{3dk}} + n^3) = \mathcal{O}(\sqrt{k}2^{2\log_2(3)\sqrt{3dk}} + n^3)$ algorithm.

Algorithm 68 A Generic Subexponential Algorithm for Planar DS

Input(s): A plane graph G and a parameter $k \in \mathbb{N}$

Output(s): YES iff G has a dominating set of size $\leq k$

```
Perform Algorithm 67 and kernelization (see Chap. 4)
Try to find a "small" partial layerwise separator $S^m$ with $|S^m| \leq h(k)/(3g(k))$.
if such a small separator cannot be found then
  return NO
end if
for all $3^{h(k)/3g(k)}$ settings of domination choices for the vertices in $S^m$ do
  for all precolored graph chunks of $G$ do
    Compute a minimal dominating set for the precolored graph chunk
  end for
  if minimum dominating set of $G$ using this choice is $\leq k$ then
    return YES
  end if
end for
return NO
```

For convenience, we list a table for different values of $f(d) = 2\log_2(3)\sqrt{3dk}$ rounded up, which we use in the following sections.

$d =$	9	15	16	20	39	45	51
$f(d) =$	16.48	21.27	21.97	24.56	34.29	36.84	39.21

These numbers correspond to the constant factor in the exponent of the running times for different estimates of $h(k)$. Observe that the last column corresponds to the $h(k)$-bound worked out in [8], combined with the outerplanarity algorithm from [253].

To allow for some "geometric arguments" when tweaking the analysis of basically Alg. 68, we need to slightly adapt our earlier notions.

If C is an embedded cycle and $x \in C$ some vertex, then we can speak about the left and right neighbor of x on C, if we imagine ourselves to be placed in the center of the cycle. Similarly, if y is some point outside of C which is connected by edges to vertices $x_0, \dots, x_{m-1}$ (and no others) on C (in that order from left to right, when "standing" in the center of C, viewing in direction of y), then there uniquely exists some j such that the region described by the edges $\{y, x_j\}$, $\{y, x_{(j+1) \bmod m}\}$ and the left-to-right path from $x_{(j+1) \bmod m}$ to x_j contains all edges from y to any x_i on C. We call $\{y, x_j, x_{(j+1) \bmod m}\}$ the *upper triple* associated to y and C (also in the boundary case when $m = 0$ and $m = 1$, when the triple will be $\{y\}$ and

$\{y, x_0\}$, respectively). Likewise, if y is a point in the interior of C and z is another reference "point" within C, then there are points x, x' on C adjacent to y such that all other vertices x'' on C which are adjacent to y are on only one of the two sub-paths into which C is cut by x and x', while z is contained in the other "half." We will call $\{y, x, x'\}$ the *lower triple* associated to y and z.

As we will understand below, it is possible to restrict oneself to the consideration of simple r-outerplanar graphs.

Our aim is to obtain a set of vertices S_i from layers L_{i-1}, L_i and L_{i+1} that separate the vertices of L_{i-1} from the vertices of L_{i+2}. In a fashion similar to [8], we achieve this by deleting "upper", "lower", and also kind of "middle" triples of vertices in $D \cap B_i$ and $D \cap J_i$. Let $b_i = |D \cap B_i|$ and $j_i = |D \cap J_i|$.

Lemma 7.91 *There exists a set S_i that separates L_{i-1} from L_{i+2} of size $|S_i| \leq 3b_{i-1} + 3j_i + 3b_i + 3j_{i+1} + 3b_{i+1}$.*

This lemma is proven quite analogous to the reasoning we detailed above.

Since $\sum_{i=3}^{r-2} |S_i| \leq 9k = h(k)$, for simple r-outerplanar graphs, we get an $2^{16.4715\sqrt{k}}$-algorithm (which is of course much better than what we obtained above). This is appears to be a lower bound on the running-time that we can obtain by the current approach.

Let us try to transfer our reasoning to the general case. Recall the notions $B(i, j)$ introduced along the definition of a ring decomposition.

As a **first preparatory step** for layer L_i, let us think about how much it would "cost" to actually separate each $B(i, j)$ from the other $B(i, j')$ such that we can be sure that any (remaining) dominating set vertex on $B(i, j)$ or in the interior of $B(i, j)$ only dominates vertices on $B(i, j)$ or in the interior of $B(i, j)$ or from J_i. Furthermore, "indirect" influences of different rings by interconnection patterns will be ruled out. Moreover, this reasoning will imply that later on we can safely speak about dominating vertices "private" to $B(i, j)$. The vertices "taken out" in this preparatory step will be put into the separator we are going to construct.

For the general case, we will need to sufficiently separate the boundary cycles in each layer so that we can employ the previous arguments for simple r-outerplanar graphs. In particular, we say that a set S^i *properly separates* L_i *according to* D if each $x \in (D \cap L_i) \setminus S^i$ dominates only vertices in one $B(i, j) \setminus S^i$ for some $1 \leq j \leq c_i$. The following results are crucial to our construction of S^i.

Lemma 7.92 *If $x \in L_i \setminus B(i, j)$ and P_1 and P_2 are two paths leading from x to some points z_1 and z_2 on $B(i, j)$, using only vertices from J_i on the way, then either $z_1 = z_2$ or z_1 and z_2 are neighbors.*

Lemma 7.93 *It suffices to take out at most four vertices to separate two linked rings of the same layer.*

As we show next, we can construct such an S^i by applying the previous two lemmas, using only $4c_i$ vertices. The proof and construction are contained in the journal version of [179].

Lemma 7.94 *There exists an S^i with $|S^i| \leq 4c_i$ that properly separates L_i according to any dominating D.*

In the example of Fig. 7.8, we would have to take out v_1, v_4 and v_8 to prevent $B(v_3)$ from influencing $B(v_{10})$, and if we assume that there is an edge between v_2 and v_9, then these two vertices would be taken out.

We now describe a **second preparatory step** for layer L_i, sketched in the following lemma, which will simplify the "triple" arguments in what follows.

Lemma 7.95 *Having performed the first preparatory step for layer L_i, there are at most $2c_i$ vertices on $B_{i-1} \cup B_i$ whose removal leaves a graph such that any path from any $x \in B_{i-1}$ down-to B_i must get to some ring $B(i,j)$ specific to x, i.e., there is a partial mapping $x \mapsto j$.*

The proof of Lemma 7.95 is omitted here (we refer to the yet unpublished journal version of [179]); however, we give some intuition concerning the approach. By removing S^i from G, we partition L_i into regions that can only reach a single $B(i,j)$. The planarity of G forces an ordering of these regions w.r.t. B_{i-1}. Hence, it is possible to delete at most 2 shared fringe vertices from B_{i-1} per $B(i,j)$ to force all paths starting at $x \in B_{i-1}$ to go to $B(i,j)$.

We now assume that we have performed both preparatory steps *for layers L_i and L_{i+1}*. This way, we take out at most $6c_i + 6c_{i+1}$ vertices.

After these preparatory steps, the graph will look "locally" like a simple r-outerplanar graph. Therefore, the analysis of that previously considered case basically transfers. Altogether, $|S_i| \leq 3b_{i-1}+3j_i+3b_i+3d_{i+1}+6c_i+6c_{i+1}$.

Theorem 7.96 *If G is a plane graph with a dominating set of size at most k, then the overall sum of the sizes of the layerwise separators can be upperbounded by $h(k) = 9k + 12c(k)$, where $c(k)$ upperbounds $\sum_{i=1}^{r} c_i$.*

We can now adapt some trade-off computations of Kanj and Perkovič [252] that show that $c(k)$ is negligibly small; in other words, the case of simple r-outerplanar graphs is the "dominating case."

Kanj and Perkovič made the following observation: There is no need to cut off branches in $\mathcal{T}'_G$ starting in a node n whose distance $\delta(n)$ to its farest descendant is only $\delta(n) \leq D(k) = \mathcal{O}(\sqrt{k})$, since, in the worst case, only $D(k)$ layers would be added to each graph chunk.

Using some simplified arguments, we show that their computations carry over to our new analysis, as well. Actually, the above analysis was "tuned" in order to exploit their technique best possible.

For a given bounding function D, call a node n in $\mathcal{T}'_G$ *deep*, *medium*, and *shallow*, respectively, if $\delta(n) > D(k)$, $\delta(n) = D(k)$, or $\delta(n) < D(k)$, respectively.

Let μ denote the number of medium nodes in $\mathcal{T}'_G$. Any medium node n contains $D(k)$ layers in $\text{int}(n)$ and therefore at least $D(k)/3$ vertices from the dominating set D. Given medium nodes n and n' in $\mathcal{T}_G$ (and hence rings in G), then $\text{int}(n) \cap \text{int}(n') = \emptyset$. If $\mu > 3k/D(k)$, then the number of dominating set vertices in $\bigcup\{\text{int}(n) : n \text{ is medium}\}$ would exceed k. Hence, $\mu \leq 3k/D(k)$.

A node is called *m-complex* if it has at least m non-shallow children. Considering the number of inner nodes in m-ary trees with μ leaves, it is readily seen that there are no more than $\lfloor(\mu-1)/(m-1)\rfloor$ many m-complex nodes. Denoting the number of non-shallow children of of n by $c(n)$, it follows that the sum of all $c(n)$, taken over complex nodes, is no larger than $\lfloor m(\mu-1)/(m-1)\rfloor$.

In the following, let us specialize towards $m = 2$.[16] Terming a non-2-complex node *simple* generalizes the earlier introduced terminology of simple r-outerplanar graphs. Our reasoning entails:

Lemma 7.97 $\sum\{c(n) \mid n \textit{ is complex}\} \leq 2\mu \leq 6k/D(k)$.

Let us now reconsider the considerations of the previous section under the observation that we only try to separate deep nodes (rings). This leads immediately to our main theorem.

Theorem 7.98 *The* PLANAR DOMINATING SET *problem can be solved in time*

$$\mathcal{O}(\sqrt{k}3^{6\sqrt{3k}+o(\sqrt{k})} + n^3) \approx \mathcal{O}(\sqrt{k}2^{6\log_2(3)\sqrt{3k}} + n^3) \approx \mathcal{O}(2^{16.4715\sqrt{k}} + n^3).$$

Observe that similar improvements can be obtained for the other domination-type problems mentioned in [8]. More specifically, for PLANAR INDEPENDENT DOMINATING SET, we get the same constants as stated in Theorem 7.98, since the treewidth- and outerplanarity-based algorithms have the same running

[16]In [252], the case $m = 3$ was considered instead.

times. But, if we like to treat say the problem of finding a total dominating set, i.e., a set D, $|D| \leq k$, such that *each vertex x* is dominated by some other vertex $y \in D$, then the constants get worse, since the treewidth-based algorithm as outlined in [8] is worse, and similar changes have to be implemented in the outerplanarity-based algorithm of Kanj and Perkovič [253].

Moreover, our results also improve on the constants derived in [123] for DOMINATING SET on $K_{3,3}$-minor-free or K_5-minor-free graphs. Here, the following theorem is important.

Theorem 7.99 *If G is a planar graph which has a dominating set of size k, then G has treewidth of at most $6\sqrt{3k} + o(\sqrt{k}) = \mathcal{O}(10.40\sqrt{k})$.*

Observe that this largely improves the upperbound $6\sqrt{34k}$ proved in [8] and also nearly matches the treewidth bound $9.55 \approx (4.5)^{1.5}$ derived in [192] via completely different arguments.

Corollary 7.100 *If the n-vertex graph G is $K_{3,3}$-minor-free or K_5-minor-free, then in time $\mathcal{O}(4^{9.55\sqrt{k}}n)$ it can be determined if G has a dominating set of size at most k or not.*

Observe that the strategy outlined in the last two sections is not only applicable to dominating set, but rather to any planar graph problem where the sum of the sizes of the layerwise separators can be estimated as a linear function in k and in the number of rings. For many problems, we can really restrict our attention to the case of simple r-outerplanar graphs, since the estimates involving the number of rings and the optimization computations do not rely on the structure of the concrete problem in any way. We exemplify this approach in the next section.

Let us briefly mention that also related problems can be solved as sketched above. Recall ROMAN DOMINATION for a moment. From Lemma 7.9 and the treewidth bound of Fomin and Thilikos [193] we can immediately conclude:

Corollary 7.101 PLANAR ROMAN DOMINATION *can be solved in time*

$$\mathcal{O}^*(5^{9.55\sqrt{k}}) = \mathcal{O}^*(2^{22.165\sqrt{k}}).$$

7.8 PLANAR RED-BLUE DOMINATING SET **and related problems**

The techniques outlined in the previous sections apply to other problems on planar graphs. For PLANAR RED-BLUE DOMINATING SET, it suffices to build an appropriate separator S_i to separate layers L_{i-1} from layer L_{i+3}.

Lemma 7.102 *There exists a separator S_i that separates layer L_{i-1} from layer L_{i+3} in G of size*

$$|S_i| \leq 3b_{i-1} + 3j_i + 7b_i + 4j_{i+1} + 5b_{i+1} + 5j_{i+2} + 5b_{i+2} + 6c_i + 6c_{i+1} + 6c_{i+2}.$$

Proof. (Sketch) Let us sketch how to prevent an "adversary path" from layer L_{i-1} to sneak through to layer L_{i+3} in this case. Observe that the "geometrical arguments" used up to now basically transfer to this case, as well, while the "domination-type" arguments needs to be adapted. The construction and more detailed proof can be found in the journal version of [179].

Our aim is again to construct a separator S_i which contains the following ingredients, basically combining two "neighbored" separators as constructed in the usual DOMINATING SET problem:

1. Do the two "preparatory steps" for layers L_i, L_{i+1} and L_{i+2}. This totals up to at most $\underline{6c_i + 6c_{i+1} + 6c_{i+2}}$ vertices going into S_i.

2. Both on the rings of L_i and of L_{i+1}, remove all dominating set vertices and their two neighbors, putting at most $\underline{3b_i + 3b_{i+1}}$ vertices into S_i.

3. For each $d \in D \cap (B_{i-1} \cup J_i)$, take out the "upper triples." This adds another at most $\underline{3b_{i-1} + 3j_i}$ vertices into S_i.

4. For each $d \in D \cap J_{i+1}$, take out the (at most) four vertices which cut off the region $\overline{R(d)}$; namely, there are (according to Lemma 7.92 at most two "connection points" of $\overline{R(d)}$ on B_{i+1} and there are at most two "outermost" connection points of $\overline{R(d)}$ on B_i (by the definition of $\overline{R(d)}$ and due to previous surgery). This adds $\underline{4j_{i+1}}$ vertices to S_i.

5. For each $d \in D \cap B_{i+1}$: if d has no neighbors in J_i but neighbors on B_i, cut out the "lower triple" associated to this situation; if d has a neighbor x in J_i, cutting out the "quadruple" as described in the previous step is what we do. In each case, due to step 2, only at most two vertices have to be removed additionally, giving another (at most) $\underline{2b_{i+1}}$ vertices into S_i.

6. For each $d \in D \cap B_i$: go "to the left" of d on B_i (and symmetrically, go "to the right") looking for red vertices on B_i which share a blue vertex in L_{i+1} with d. Geometrically, this gives us a series of nested triangles. We take out the endpoints of the "largest" (on both sides). Due to step 2, this adds at most $\underline{4b_i}$ vertices into S_i.

7. For each $d \in D \cap (J_{i+2} \cup B_{i+2})$, first remove the lower triple $\{d, z_1, z_2\}$ associated to d and some (other) z on B_{i+2}. This lower triple describes a region enclosed by the edges $\{d, z_1\}$ and $\{d, z_2\}$ as well as a path P on B_{i+1} between z_1 and z_2 such that z is not enclosed in this region. Let y_1 and y_2 be the two neighbors of z_1 and z_2, respectively, which are not contained in the region. Let u_1 and u_2 denote the two "outermost" vertices on B_i such that there is a path from P to u_i using only intermediate vertices from J_i (if any). Then, put $\{d, z_1, z_2, u_2, u_2\}$ into S_i, given another $\underline{5j_{i+1} + 5b_{i+1}}$ vertices.

This totals up to

$$|S_i| \leq (3b_{i-1} + 3j_i) + (7b_i + 4j_{i+1}) + (5b_{i+1} + 5j_{i+2}) + 5b_{i+2} + 6c_i + 6c_{i+1} + 6c_{i+2}$$

▮

Adapting the notion of "partial layerwise separator" to allow for considering only every fourth layer, there exist a partial layerwise separator S with $||S|| \leq h(k) = 20k + 18c(k)$. Since the trade-off computations only involve combinatorics on ring decomposition trees, they readily carry over to this case, basically making the term $c(k)$ disappear when it comes to the running time of algorithms.

Taking either the $27^l n$-algorithm (l denoting the number of layers of the graph, i.e., its outerplanarity) as given in [253] or the 3^{tw}-algorithm (tw denotes the treewidth of the graph; observe that $tw < 3out$) exhibited in [8] and finding a c such that the overall size of the partial layerwise separators $20k/(4c\sqrt{k})$ matches the treewidth of the remaining graph chunks $12c\sqrt{k}$, i.e., $c = \frac{\sqrt{5}}{\sqrt{12}} = \frac{\sqrt{5}}{2\sqrt{3}}$. This gives the following result.

Theorem 7.103 PLANAR RED-BLUE DOMINATING SET *can be solved in time*

$$\mathcal{O}(3^{2 \cdot 10 \cdot \frac{\sqrt{3}}{\sqrt{5}} + o(\sqrt{k})} n) = \mathcal{O}(2^{24.551\sqrt{k}} n).$$

Moreover, a planar graph which has a k-red/blue dominating set has treewidth of at most $20\frac{\sqrt{3}}{\sqrt{5}} + o(\sqrt{k}) \approx 15.5\sqrt{k}$.

Using the results of [123], this readily transfers to the case of $K_{3,3}$-minor-free graphs and of K_5-minor-free graphs.

Theorem 7.103 has several important applications, as we show next.

Our results concerning PLANAR RED-BLUE DOMINATING SET can be used to show that each problem in a subset of the class Planar $\mathcal{TMIN}$ has a $c^{\sqrt{k}}$ parameterized algorithm. Planar $\mathcal{TMIN}$ is a syntactic class that was defined by Khanna and Motwani [256], along with two other classes, to characterize those problem that admit PTAS. The precise definition of Planar $\mathcal{TMIN}$ and it subclasses requires a brief explanation.

Given a collection of variables X, a *minterm* is simply a conjunction (AND) of literals over X. A literal l is *negative* if it is the negation of some variable $x_i \in X$, i.e., $l = \neg x_i$. Otherwise, a literal is *positive.* A minterm m is positive if all of the literals in m are positive. Likewise, a minterm m is negative if all of the literals in m are negative. A first order formula (FOF) is a disjunction (OR) of minterms. A FOF is positive if all of its minterms are positive. Similarly, A FOF is negative if all of its minterms are negative. The *width* of a FOF is the number of minterms in the formula. The *size* of a minterm is the number of literals in the minterm.

The class $\mathcal{TMIN}$ [256] is the class of all $\mathcal{NP}$-optimization problems that can be written (rewritten) as follows. Given a collection C of positive FOFs over n variables, find a minimum weighted truth assignment T that satisfies all FOFs in C. Given a collection C of FOFs over n variables X, the *incidence graph* of C is the bipartite graph G_C with edges between FOFs in C and the set of variables X such that there is an edge between a formula and a variable if and only the variable appears in a minterm in the formula. The class Planar $\mathcal{TMIN}$ is the class $\mathcal{TMIN}$ restricted to problems with planar incidence graphs.

Cai, Fellows, Juedes and Rosamond (unpublished) have shown that there exist problems in Planar $\mathcal{TMIN}$ that are W[1]-hard, and hence not all problems in Planar $\mathcal{TMIN}$ have parameterized tractable algorithms unless $\mathcal{FPT} = W[1]$. In contrast, we show here that all the problems in subclass of Planar $\mathcal{TMIN}$, known as Planar $\mathcal{TMIN}_1$, have fast parameterized algorithms.

Planar $\mathcal{TMIN}_1$ is the subclass of Planar $\mathcal{TMIN}$ where all minterms are restricted to be of size 1. It is easy to see that PLANAR VERTEX COVER is in Planar $\mathcal{TMIN}_1$ since the minimum vertex cover in a planar graph G can be described as the minimum weighted truth assignment satisfying

$$\bigwedge_{(u,v)\in E} (x_u \vee x_v).$$

Notice that each term $x_u \vee x_v$ is FOF, and the the incidence graph of this collection of FOFs is planar.

Theorem 7.103 leads immediately to the following result.

Theorem 7.104 *Each problem Π in Planar $\mathcal{TMIN}_1$ can be solved in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{k})}p(n))$ for some polynomial $p(\cdot)$.*

The main application of this general result is that faster algorithms for PLANAR RED-BLUE DOMINATING SET can be used to achieve faster algorithms for a variety of problems in Planar $\mathcal{TMIN}_1$. In particular, this leads to the fastest-known parameterized algorithms for FACE COVER.

Corollary 7.105 FACE COVER *can be solved in time* $\mathcal{O}(2^{24.551\sqrt{k}}n)$.

Proof. Place variables on each face and formulae describing the face-incidence at each vertex of the given graph. ∎

This largely improves the best-known algorithm running in time $\mathcal{O}(3^{36\sqrt{34k}}n)$ according to [8]. Moreover, Alber *et al.* discuss the related DISK DIMENSION problem. The DISK DIMENSION problem treated by Bienstock and Monma [47] generalizes FACE COVER in two ways: firstly, they do not start with a fixed embedding of the planar input graph and, secondly, they have an additional input of their problem, namely a set D of designated vertices, where only these need to be covered. In [8], it is admitted that "both of these generalizations seem to be hard to treat within our framework." As regarding the first generalization that basically involves minimizing over all planar embeddings, we don't know how to tackle this with the present approach, either. However, having a set D of designated vertices in the input is amenable to our approach, since it would only change the set of blue (formulae) vertices in the translation into Planar $\mathcal{TMIN}_1$. Recall our discussion of FACE COVER and ANNOTATED FACE COVER from Chap. 5.

Corollary 7.106 *The* PLANAR EDGE DOMINATING SET *problem can be solved in time* $\mathcal{O}(2^{24.551\sqrt{k}}n)$.

Proof. Place variables on each edge and formulae describing edge-adjacencies at each vertex of the given plane graph. ∎

7.9 Other related graph classes

Up to now, we focused on algorithms for planar graphs. There are, however, related graph classes that have been also investigated from the point of view of parameterized algorithmics. We shortly report on these results here. Details can be found in the mentioned papers.

Disk graphs. Disk graphs are a subclass of intersection graphs described by disks in the plane, i.e., the vertices of the corresponding graph are the disks, and an edge between two vertices indicates that the corresponding disks overlap. A collection D of disks is λ-*precision* if all centers are pairwisely at least λ apart.

Alber and Fiala [16] consider the following parameterized problem, whether for a given set D of n disks (of bounded radius ratio, i.e., the maximum radius among all disks divided by the minimum radius among all disks is bounded by some α) in the Euclidean plane there exists a set of k non-intersecting disks.

For this problem, they expose an algorithm running in time $n^{\mathcal{O}(\sqrt{k})}$. For λ-precision disk graphs of bounded radius ratio, they show that the problem is fixed parameter tractable with a running time $\mathcal{O}(c^{\sqrt{k}} + p(n))$ for some constant c and a polynomial p. The results are based on problem kernelization (also see Section 9.2) and a new "geometric (-separator) theorem" which holds for all disk graphs of bounded radius ratio.

So, more precisely, they showed membership in $\mathcal{FPT}$ for the following problem, where the considered graph class is predetermined by the choice of α and λ:

Problem name: INDEPENDENT SET ON DISKS GRAPHS (DIS)
Given: A disk graph $G = (V, E)$ whose disk model has radii between 1 and α and is λ-precision
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Bounded genus graphs. The notion of bounded genus generalizes the notion of a planar graph in a straightforward manner: no longer graphs that are embeddable on a sphere are looked at (this corresponds to graphs of genus zero) but graphs that are embeddable on the surface of a sphere that is decorated with γ handles; this graph parameter γ upperbounds the *genus* of the graph under consideration.

We already mentioned in Sec. 7.5 that for these graph classes, $\mathcal{FPT}$-algorithms of running time $\mathcal{O}^*(c_\gamma^{\sqrt{k}})$ can be constructed. Alternatively, Ellis, Fan and Fellows have shown in [154] that DOMINATING SET ON BOUNDED GENUS GRAPHS can be solved in time $\mathcal{O}((4\gamma + 40)^k n^2)$ by a search tree algorithm.

There is also an algorithmic theory for solving basically all problems mentioned in this chapter (amongst others) in time $\mathcal{O}^*(c^{\sqrt{k}})$ that was recently developed by Demaine, Fomin, Hajiaghayi and Thilikos [116].

***H*-minor free graphs.** The theory we mentioned [116] also applies to H-minor free graphs. Related recent papers include [112, 121, 122, 216, 185].

For the particular case of $K_{3,3}$ and K_5 as forbidden minors, we mentioned special results in the previous section.

Chapter 8

Further approaches

In this chapter, we discuss further approaches to the phenomenon of parameterized problems, including some non-standard applications. Moreover, this chapter can be seen as containing some ideas how to further develop areas in which the parameterized paradigm can be applied.

In Section 8.1, we present one approach that has been proven to be useful to develop efficient parameterized algorithms: dynamic programming on subsets. We proceed with a section (Sec. 8.2) on enumeration, a technique that is also useful for a "rapid prototyping" approach to decision problems. Related to enumerating all solutions is the issue of counting them, which is dealt with in Section 8.3.

Section 8.4 briefly reviews some other techniques that are useful for proving $\mathcal{FPT}$ results.

Another issue is of course how to practically employ the methodologies presented in this Habilitationsschrift. A short section is devoted to this issue, dealing both with sequential and parallel computations.

8.1 Dynamic programming on subsets

Fomin, Kratsch and Woeginger [187] recently came up with an efficient parameterized algorithm for the following problem, which is in fact an alternative parameterization of HITTING SET:

Problem name: MINIMUM HITTING SET, PARAMETERIZED BY # EDGES (HSE)
Given: A hypergraph $G = (V, E)$
Parameter: $|E|$
Output: Find a minimum *hitting set* $C \subseteq V$

The algorithm uses a technique known as *dynamic programming on subsets*. To this end, given a hypergraph $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$, the algorithm maintains a 2-dimensional array F that contains, for $E' \subseteq E$ and for $j = 1, \dots, n$, in $F[E', j]$ the minimum cardinality of a subset C of $V_j := \{v_1, \dots, v_j\}$ that covers E' (C is also called a *hitting set for* E' (*relative* to V_j)); if no such cover exists, set $F[E', j] = \infty$. More details on how to construct the entries for F are contained in Alg. 69. There, also the basic reasoning for the inductive step in the correctness proof of that algorithm is given.

Let us mention that the famous Dreyfus-Wagner algorithm for solving the Steiner tree in graphs problem is based on a similar recursion and exhibits basically the same running time, see [326]. This is not completely surprising, given the relatively close connections between red-blue dominating set and Steiner tree in graphs.

Algorithm 69 A dynamic programming algorithm for minimum hitting set, parameterized by # edges, called HSE

Input(s): a hypergraph $G = (V, E)$, $V = \{v_1, \dots, v_n\}$
Output(s): a hitting set $C \subset V$ of minimal cardinality

```
for all $E' \subseteq E$ do
  $F[E', 1] := 1$
  for all $e \in E'$ do
    if $v_1 \notin e$ then
      $F[E', 1] := \infty$
    end if
  end for
end for
for $j = 2, \dots, n$ do
  for all $E' \subseteq E$ do
    Let $E'' := \{e \in E' \mid v_j \in e\}$.
    $F[E', j] := \min\{F[E', j-1], F[E' \setminus E'', j-1] + 1\}$
    {Two cases arise: either $v_j$ is not belonging to a minimum hitting
    set for $E'$, then, $F[E', j] = F[E', j-1]$; or $v_j$ belongs to a minimum
    hitting set $C$ for $E'$, but then, $C \setminus \{v_j\}$ is a minimum hitting set for
    $E' \setminus E''$ relative to $V_{j-1}$, so that $F[E', j] = F[E' \setminus E'', j-1] + 1$.}
  end for
end for
```

Theorem 8.1 minimum hitting set, parameterized by # edges *can be solved in time* $\mathcal{O}^*(2^{|E|})$ *for a hypergraph instance* $G = (V, E)$.

When dealing with sets, it is however often sufficient to do a much simpler thing, namely to test all possible assignments of certain numerical values to the set under scrutiny. A nice non-trivial application of this simple idea was recently published by B. Reed, K. Smith and A. Vetta [330]: they considered the following problem:

Problem name: BIPARTIZATION (BP)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *bipartization set* $C \subseteq V$ with $|C| \leq k$ whose removal produces a bipartite graph?

It was open for more than 10 years if this problem belongs to $\mathcal{FPT}$ or not. Their algorithm is based on a couple of nice ideas that might be useful for similar situations, as well. The importance of the result for the area of parameterized algorithmics is also described in [243]. But let us first present the algorithm in Alg. 70.

The main point to the correctness of this greedy algorithm is of course the subroutine BP-improve presented in Alg. 71 that contains the exponential part of the overall algorithm. To describe this algorithm, we introduce the following additional auxiliary notations.

Given a set $X \subset Y$, denote by $X[i]$ the set $\{x_i \mid x \in X\}$, where we assume that $X[i] \cap Y \neq \emptyset$.

If $G = (V, E)$ is a graph with bipartization set $C \subseteq V$, then $G - C$ has (by definition) two independent sets S_1, S_2 such that $S_1 \cup S_2 = V \setminus C$. $BP(G, C)$ denotes the *bipartization variant* of G with respect to C, where the graph $BP(G, C) = (V', E')$ is given as follows:

- $V' = (V \setminus C) \cup C[1] \cup C[2]$.
- E' contains the following edges:
  1. all edges from $G - C$;
  2. if $e \in E$ joins some $y \in S_i$ to some $c \in C$, then we put a corresponding edge e' into E' that joins y to c_{3-i};
  3. if $e \in E$ joins two vertices $x, y \in C$, then we put a corresponding edge e' into E' that either joins x_1 to y_2 or that joins x_2 to y_1.

Given $v' \in V'$, call v *corresponding* to v' if either $v' \in V \setminus C$ and $v = v'$ or $v \in C$ and $v' = v_i \in C[i]$ for $i = 1, 2$.

The bipartization variant $BP(G, C)$ in indeed bipartite, since $S_1 \cup C[1]$ and $S_2 \cup C[2]$ are both independent sets.

Algorithm 70 A greedy algorithm for BIPARTIZATION, called GBP

```
Input(s): a graph $G = (V, E)$, a positive integer $k$
Output(s): if possible: a subset $C \subset V$, $|C| \leq k$ whose removal produces a
  bipartite graph or
  NO if no such set exists.
  if $V = 0$ AND $k \geq 0$ then
    return $\emptyset$
  else
    pick a vertex $v$
    $C :=$ GBP$(G - v, k)$
    if $C =$NO then
      return NO
    else if $|C| < k$ then
      return $C \cup \{v\}$
    else
      $C :=$ BP-improve$(G, C \cup \{v\}, k)$
      if $C =$NO then
        return NO
      else
        return $C$
      end if
    end if
  end if
```

Given a subset $Y \subset C$, a partition of $Y[1] \cup Y[2]$ into two sets Y_A and Y_B is a *valid partition* if, for all $y \in Y$, either $y_1 \in Y_A$ and $y_2 \in Y_B$ or $y_1 \in Y_B$ and $y_2 \in Y_A$.

The correctness of Alg. 71 relies on a characterization of minimum bipartization sets in terms of valid partitions that is contained in [330]; we refer the interested reader to that paper. Note that (as stated in Alg. 71), the bipartization improvement subroutine can be implemented to run in time $\mathcal{O}^*(3^k)$, which is slightly better than claimed in the paper [330]. [1]

So, we can state the following results:

Theorem 8.2 BIPARTIZATION IMPROVEMENT *can be solved in time* $\mathcal{O}(3^k mn)$, *where* n *and* m *are the number of vertices and edges, respectively, of the input graph.*

[1]This improvement was (to our knowledge) first announced in the invited talk R. Niedermeier gave at MFCS'04.

Algorithm 71 A search algorithm to improve solutions for BIPARTIZATION, called improve-BP

```
Input(s): a graph $G = (V, E)$, a bipartization set $C$, a positive integer $k$
  such that $|C| = k + 1$
Output(s): if possible: a subset $C \subset V$, $|C| \le k$ whose removal produces a
  bipartite set or
  NO if no such set exists.

  Let $G' = (V', E') = BP(G, C)$.
  found:=NO
  for all mappings $\alpha : C \to \{0, 1, 2\}$ AND not found do
    Let $Y := \{c \in C \mid \alpha(c) \neq 0\}$.
    Let $Y_A := \{c_i \mid c \in C, \alpha(c) = i\}$.
    Let $Y_B := \{c_i \mid c \in C, \alpha(c) = 3 - i\}$.
    Let NOT$(C) := C[1] \cup C[2] \setminus (Y_A \cup Y_B)$.
    {By construction, $Y_A$ and $Y_B$ form a valid partition.}
    if there are less than $|Y|$ vertex disjoint paths from $Y_A$ to $Y_B$ in $G'' :=$
    $G' -$ NOT$(C)$ then
      found:=YES
      Let $W'$ be a cutset that separates $Y_A$ from $Y_B$ in $G''$, with $|W'| < |Y|$.
      Let $W$ be the set of vertices in $G$ that correspond to vertices in $W'$.
      {By definition, $|W| \le |W'|$.}
      {$W \cup (C \setminus Y)$ is a bipartization set of $G$}
    end if
  end for
  if found then
    return $W$
  else
    return NO
  end if
```

Corollary 8.3 BIPARTIZATION *can be solved in time $\mathcal{O}(3^k mn)$, where n and m are the number of vertices and edges, respectively, of the input graph.*

We explain the work of this algorithm by a small example.

Example 8.4 Let $G = (V, E)$ with $V = \{1, 2, 3, 4, 5\}$. G consists of two triangles: $(1, 2, 3, 1)$ and $(1, 4, 5, 1)$. We want to know if there exists a bipartization set of size 1. Assume that the ith recursion of the main algorithm, GBP is called with the graph $G[\{1, \dots, i\}]$. For example, GBP$(G[\{1, 2\}, 1)$ returns the empty set. Therefore, GBP$(G[\{1, 2, 3\}, 1)$ returns $\{3\}$. Similarly, GBP$(G[\{1, 2, 3, 4\}, 1)$ returns $\{3\}$. Hence, GBP$(G[\{1, 2, 3, 4, 5\}, 1)$ first constructs $\{3, 5\}$ as bipartization set. The subroutine BP-improve will however find that this is not a minimum bipartization set. For example, with $Y_A = \{3_1, 5_2\}$ and $Y_B = \{3_2, 5_1\}$, $\{1\}$ appears to be a small cutset in G''. Hence, $\{1\}$ is a smaller bipartization set in G'.

Let us summarize some of the ideas that might turn out to be fruitful in other circumstances, as well.

- The actual recursion in the main routine is "trivial" and does not make any branches at all.

  More precisely: The recursion will produce a solution either of size k or of size $k + 1$ or it returns that no solution exists. If a solution of size k is returned, this is o.k. If the size is $k+1$, a subroutine is triggered that checks if the solution is minimal. If the solution is minimal, the overall procedure returns that no solution exists. If the solution is not minimal, a smaller solution is generated and is used in the further recursion.

  The parameterized complexity behavior is hidden in the subroutine.

- The general idea is therefore to produce a solution that might be slightly larger than wanted but then verify minimality.

- The mentioned subroutine works by trivial assignment of three values to the set under scrutiny and testing all possibilities. This also immediately gives the claimed time bound.

- Reed *et al.*'s algorithm seem to be the first example of an *improvement problem* to be observed in $\mathcal{FPT}$. More precisely, it can be seen that Alg. 71 puts the following problem in $\mathcal{FPT}$:

  Problem name: BIPARTIZATION IMPROVEMENT
  Given: A graph $G = (V, E)$, a bipartization set $C \subseteq V$ with $|C| = k + 1$
  Parameter: a positive integer k
  Output: Is there a bipartization set $C' \subseteq V$ with $|C'| \leq k$?

  It is clear that corresponding problems can be also studied in other circumstances. While it is trivial that, whenever the "original version" of a problem is in $\mathcal{FPT}$, then its "improvement version" is in $\mathcal{FPT}$, as well, Reed *et al.*'s algorithm is—to our knowledge—the first example of how to use a converse relation. It would be interesting to see examples of W[1]-hard problems whose improvement version lies in $\mathcal{FPT}$.

The problems BIPARTIZATION and BIPARTIZATION, EDGE VARIANT are known to be interrelated. Choi, Nakajima and Rim showed [98] that the problems are equivalent if the degree of the input graphs is limited to three. More generally, S. Wernicke [376, Theorem 6.7] provided a parameterized reduction that shows how to solve BIPARTIZATION, EDGE VARIANT with the help of BIPARTIZATION. That reduction transforms an instance (G, k) of BIPARTIZATION, EDGE VARIANT into an instance (G', k') of BIPARTIZATION such that $k' = k$, i.e., it is even parameter-preserving. We can therefore state as a corollary:

Corollary 8.5 BIPARTIZATION, EDGE VARIANT *is in $\mathcal{FPT}$. More specifically, an instance (G, k) of* BIPARTIZATION, EDGE VARIANT *can be solved in time $\mathcal{O}^*(3^k)$.*

Lemma 6.20 allows us then to state:

Corollary 8.6 BIPARTIZATION, REPLACING EDGES BY 2-PATHS VARIANT *is in $\mathcal{FPT}$.*

Note, however, that the links between BP and BPEDGE are not perfect: while BP, restricted to planar graphs, is still $\mathcal{NP}$-hard, see [98], BPEDGE (or equivalently MAXIMUM CUT) then becomes solvable in polynomial time, see [223].

Let us return again to FACILITY LOCATION, which we have previously looked into in Chap. 4.[2] The idea of *dynamic programming on subsets* gives us a better overall running time.

Theorem 8.7 FACILITY LOCATION *can be solved in time $\mathcal{O}(k^{3k+1}2^k + |M|)$.*

Proof. We start by kernelization as described in Lemma 4.10. In a preprocessing phase, we compute the cost incurred by a certain set of customers when being served by a single facility, storing the results in a table "one-serve" with 2^k entries. Since there are at most k^{3k} many facilities and k customers, we get a running time of $\mathcal{O}(k^{3k}2^k)$. Then, we can compute the minimal costs of serving a certain group of customers by some facilities by dynamic programming, combining two subsets at a time. If we have stored the results of the preprocessing in table "one-serve", each step in the dynamic programming will take only constant time, so that we arrive at the following formula for dynamic programming:

$$c(A) = \min\{\min_{\emptyset \neq B \subsetneq A} c(B) + c(A \setminus B), \text{one-serve}(A)\}$$

[2]What follows is part of unpublished work with M. Fellows.

Disregarding preprocessing time, this amounts in $\mathcal{O}(4^k)$ operations; the preprocessing actually worsens this in the way explained. ∎

Remark 8.8 *If we neglect the kernelization, we arrive at an $\mathcal{O}(4^k|M|)$ algorithm for* FACILITY LOCATION.

We like to remark that FACILITY LOCATION shows up in really many places. For example, in a project called XTRACT, M. Garofalakis *et al.* [203] proposed to apply the Mimimum Description Length (MDL) principle to select the most appropriate Document Type Definition (DTD) for a specific element of an XML document with unknown or insufficiently known DTD. Since the related combinatorial problem seems to be hard,[3] they proposed taking approximation algorithms for FACILITY LOCATION to practically solve the involved MDL-based optimization step. Their straightforward translation can be also read as a parameterized reduction, so that in principle also our exact algorithm could be incorporated in that project.

Fellows *et al.* also used dynamic programming on subsets to obtain better running times for algorithms dealing with packings and matchings of geometric objects and graphs [168].

8.2 Parameterized Enumeration

8.2.1 General notions and motivation

The problem of enumerating all minimal hitting sets is an important example in the area of Artificial Intelligence, where this problem is known as the *transversal hypergraph problem*, see [151, 152, 153, 331]. Further references and links to problems in computational biology can be found in [108]. In fact, we already saw a couple of applications of this technique in Chap. 6. The following exposition, however, rather follows the ideas from [171], the notions being slightly modified, to reconciliate them with [108].

We call an optimization problem *fixed-parameter enumerable* iff all optimal solutions of size k can be listed in time $\mathcal{O}(f(k)p(n))$.

In [171], we rather studied the following problems from a parameterized standpoint for an optimization problem:

- generate *all* feasible solutions of size k,
- generate *all optimum* solutions of size k, and
- generate *representative* solutions.

[3]In fact, we recently showed $\mathcal{NP}$-hardness of that problem [173].

Then—in contrast what we define here—, the first of these versions was called fixed-parameter enumerable. In [171], an example was given for VERTEX COVER that shows that even this "simple" problem is not enumerable in this sense: the graph with n vertices and no edges has $\mathcal{O}(n^k)$ many covers of size k. For minimization problems, the tasks of listing all minimal solutions of size k and of listing all minimum solutions of size k are parameterized interreducible, as one easily sees.

We will return to what is meant be the third enumeration task (to generate representative solutions), later in this section.

8.2.2 Enumerating hitting sets

It is rather straightforward to devise a variant of Mehlhorn's simple search tree algorithm(s) to list all minimal vertex covers up to a size of k, see Alg. 72.

Algorithm 72 A simple enumerating search tree algorithm, called VC-enum

Input(s): a graph $G = (V, E)$, a positive integer k, a subset C containing the vertices already assumed to be in the cover; furthermore, global read access to the original graph and write access to a global list L of covers (to be constructed) is required.

Output(s): if possible, returns a minimal vertex cover $C \subseteq V$, $|C| \leq k$ (and puts it onto a global list L), or
NO if no vertex cover of size at most k exists.

```
if $k \leq 0$ and $E \neq \emptyset$ then
  return NO
else if $k \geq 0$ and $E = \emptyset$ then
  for all $x \in C$ do
    if $C \setminus x$ is a cover of the original graph then
      return NO
    end if
  end for
  {$C$ is a minimal vertex cover with no more vertices than required by the original bound}
  put $C$ onto the list $L$ and return
else
  Choose edge $e = \{x, y\} \in E$
  VC-enum($G - x, k - 1, C \cup \{x\}$)
  VC-enum($G - N(x), k - \deg(x), C \cup N(x)$)
end if
```

The correctness of Alg. 72 is based on the following characterization of minimal vertex covers:

Lemma 8.9 *C is a minimal vertex cover of a graph $G = (V, E)$ if and only if*

- *C is a cover of G, and*
- *for each $x \in V$, $N[x]$ is not completely contained in C.*

Proof. If C is a vertex cover of G that is not minimal, then there is a strict subset C' of C that is already a cover of G. Assume (w.l.o.g.) that $C = C' \cup \{x\}$. Since x can be deleted from C without destroying the cover property, all neighbors of x must be in C. Hence, $N[x] \subseteq C$.

Conversely, if C is a vertex cover of G with some vertex $x \in C$ such that $N[x] \subseteq C$, then $C \setminus \{x\}$ is also a cover of G, so that C is not minimal. ∎

By providing read access to the original instance (G, k) and by initializing a list L of covers by the empty list, Alg. 72 can be used to list all minimal vertex covers, thanks to the following proposition.[4]

Proposition 8.10 *Alg. 72 lists all minimal vertex covers of size up to k within the list L.*

Proof. Due to the minimality test in Alg. 72, it is clear that Alg. 72 only lists minimal vertex covers of size up to k within the list L.

The question is if it is possible not to list some minimal vertex covers by ignoring non-minimal covers. Let C be such a non-minimal cover. Due to Lemma 8.9, there is a vertex x such that $N[x] \subseteq C$. How is is possible that x together with all its neighbors is put into a cover along the construction performed by the algorithm?

Assume first that during the course of the algorithm we branch at x.

In the branch that x is not put into the cover, $N(x)$ is put into the cover. This implicitly makes x an isolated vertex, since all edges incident with x are covered and can be hence removed. Therefore, x will never be considered anymore by the algorithm, since only edges are picked for branching. Therefore, in this branch, x will never be put into the cover, so that $N[x]$ won't be completely contained in any cover produced this way. So, the non-minimal cover C can not turn up this way.

In the branch that puts x into the cover, it might happen that later on all neighbors of x are put into the cover. Hence, with hindsight, having put x

[4]P. Damaschke proposes a similar algorithm for listing all minimal vertex covers in [108]. Prop. 8.10 basically justifies his statement "One easily verifies that any minimal vertex cover appears...".

into the cover turns out to be superfluous. But the case that all neighbors of x are put into the cover is also treated by the algorithm, see the considerations in the preceding paragraph. Hence, it is sound not to further consider C (as done by the algorithm).

Finally, we have to discuss the possibility that $N[x]$ is put into C without ever branching on x. Since we are considering all uncovered edges within the course of the algorithm, this means that at least for one edge $\{x, y\}$ we have branched on y.

In the branch that does not put y into the cover, we cannot have put all of $N[x]$ into C by a reasoning similar to the one above when we considered the case the x is not put into the cover during branching. So, the only interesting remaining case is that we put y into the cover. This reasoning is true for all neighbors y' of x, so that "at worst" finally all neighbors of x are put into the cover. But then, all edges incident with x are covered, so that x won't be considered any more, and hence $N[x]$ won't be completely contained in C. ∎

From the previous proposition and due to the trivial binary branching in the search tree, we can deduce:

Theorem 8.11 *Alg. 72 lists all minimal vertex covers of size up to k in time $\mathcal{O}^*(2^k)$.*

In fact, if we knew how to deal with small-degree vertices, then Alg. 72 could actually run faster. Unfortunately, for the mere task of listing all minimal vertex covers, we can do no better, as seen by the following example.

Remark 8.12 *Essentially, there is no better minimal vertex cover enumeration algorithm than the one given in Alg. 72, since the graph*

$$(\{1, \dots, k\} \times \{1, 2\}, \{\{(i, 1), (i, 2)\} \mid 1 \leq i \leq k\})$$

has 2^k many different minimum (!) vertex covers.

This simple example is interesting, since it shows, in addition, that there is no minimal vertex cover enumeration algorithm for planar vertex cover having running time of the form $c^{\sqrt{k}}n$, as it has been found for the decision problem for example in [15].

Remark 8.13 *We only mention that the reduction rules 1 and 2 presented for* VERTEX COVER *in Chap. 2 are also valid for the purpose of parameterized enumeration; this idea is detailed in [108]. More precisely, if C_1 is the set of vertices put into* any *vertex cover (of size at most k) due to Rule 2, then each minimal solution C_2 that is found by the search-tree method explained* *above (when applied to the kernel) can be transformed into a minimal solution $C_1 \cup C_2$ of the original graph instance. Details are left to the reader.*

Note that also (variants of) the Nemhauser-Trotter theorem can be used to find sets of vertices C_1' that must be contained in any minimum *vertex cover, see Rem. 4.37. In practice, these can be used as well to further restrict the size of the search space if looking for an enumeration of all minimum vertex covers up to a given size, see [96, 95] for more details. Observe that this corresponds to the kind of enumeration problem proposed in [171].*

There is one more idea from [108] we like to mention here: the idea of enumerating *compact representations* of vertex cover solutions. Then, the example from Rem. 8.12 could be easily dealt with when allowing special representations for matchings. Damaschke developed a rather intricate algorithm to enumerate compact representations of all minimal vertex covers up to size k. If we relax this task a bit and allow possible enumerations of additional minimal vertex covers of size larger than k in some situations, we can actually find an easier algorithm with better running time, if we understand the following as a compact representation;[5] observe that a compact representation formally represents a collection of covers:

1. $\emptyset$ is an expression denoting a compact representation that denotes no sets at all, i.e., $C(\emptyset) = \{\emptyset\}$.

2. If a is a vertex, then a is an (atomic) compact representation of the cover collection $C(a)$ only containing the cover $\{a\}$, i.e., $C(a) = \{\{a\}\}$.

3. If a is a vertex, then $\hat{a}$ is an (atomic) compact representation of the cover collection $C(\hat{a})$ only containing the covers $\{a\}$ and $N_1(a)$, where $N_1(a)$ collects all neighbors of a of degree one, i.e., $C(\hat{a}) = \{\{a\}, N_1(a)\}$.

4. If A and B are compact representations that represent cover collections $C(A)$ and $C(B)$, resp., then $A + B$ represents the cover collections

$$C(A + B) = \{X \cup Y \mid X \in C(A), Y \in C(B)\}.$$

   In the special case that $A = \emptyset$, then $A + B = B$.
   In the special case that $B = \emptyset$, then $A + B = A$.

5. If A and B are compact representations that represent cover collections $C(A)$ and $C(B)$, then $A \cup B$ represents the cover collection $C(A \cup B) = C(A) \cup C(B)$.

[5]The following results haven't appeared elsewhere.

6. Nothing else are compact representations.

Example 8.14 For example, the minimal vertex covers of the graph

$$(\{1,\dots,k\}\times\{1,2\},\{\{(i,1),(i,2)\}\mid 1\leq i\leq k\})$$

can be written as

$$\widehat{(1,1)}+\widehat{(2,1)}+\cdots+\widehat{(k,1)}.$$

For instance, if $k=3$,

$$\begin{aligned}&\widehat{(1,1)}+\widehat{(2,1)}+\widehat{(3,1)}\\ &=\ \{\{(1,1)\},\{(1,2)\}\}+\{\{(2,1)\},\{(2,2)\}\}+\{\{(3,1)\},\{(3,2)\}\}\\ &=\ \{\{(1,i),(2,j),(3,\ell)\}\mid 1\leq i,j,\ell\leq 2\}\end{aligned}$$

In the course of Alg. 73, we will construct a compact representation C of minimal vertex covers of the input graph. Accompanying a graph $G=(V,E)$, we have a representation function ρ that maps a vertex x either onto x or onto $\hat{x}$. Initially, ρ will map each vertex x onto x. Similarly, the initial cover collection C will equal $\{\emptyset\}$.

Then, we will use modified reduction rules:

Reduction rule 68 *If (G,k) is the graph instance with representation function ρ and C the compact representation of the partial covers found so far and if x is an isolated vertex, then delete x from G. If $\rho(x)=\hat{x}$, then modify $C:=C+\rho(x)$ and decrement k.*

Reduction rule 69 *Let x be a vertex in G of degree one. If $N_1(x)\neq\emptyset$, then delete $N_1(x)$ from G and modify ρ such that $\rho(x)=\hat{x}$.*

Let us illustrate the work of the reduction rules with the help of Example 8.14:

Example 8.15 We continue considering Example 8.14 with $k=3$. Initially, $\rho((i,j))=(i,j)$, and $C=\{\emptyset\}$.

Let Rule 69 apply to $(1,1)$, i.e., now $\rho((1,1))=\widehat{(1,1)}$, and $N_1((1,1))=\{(1,2)\}$ got deleted from the instance. Then, Rule 68 may trigger with respect to $(1,1)$, which is now an isolate. Since $\rho((1,1))=\widehat{(1,1)}$, the previously initiated $C=\{\emptyset\}$ is modified to become $C+\widehat{(1,1)}=\{\{(1,1)\},\{(1,2)\}\}$.

Then, Rule 69, followed by Rule 68, might apply to $(1,1)$, further modifying C to become

$$C+\widehat{(1,2)}=\{\{(1,1),(2,1)\},\{(1,1),(2,2)\},\{(1,2),(2,1)\},\{(1,2),(2,2)\}\}.$$

Another sequence of applications of the reduction rules, this time to $(3, 2)$ (e.g.), leaves us with the correct 8-element minimal cover collection, as already computed in Example 8.14.

Note that (as can be seen by the example), $\hat{x}$ will be interpreted with respect to the original graph, not the one modified due to Rule 69.

Algorithm 73 A search tree algorithm for enumerating compact representations of vertex covers, called VC-enum-compact

```
Input(s): a graph $G = (V, E)$, a representation function $\rho$, a positive integer
  $k$, a compact representation $C$ of the partial cover found so far.
Output(s): If possible: Collects into $C$ compact representations of all min-
  imal vertex covers of size up to $k$.

  Apply reduction rules 68 and 69, this way possibly modifying the instance.
  if $k \leq 0$ and $E \neq \emptyset$ then
    return NO
  else if $k \geq 0$ and $E = \emptyset$ then
    Choose a vertex $x$ of maximum degree in $G$
    {$\deg(x) \geq 2$ due to the reduction rules}
    {For simplicity, $\rho$ also denotes the appropriate restriction of $\rho$ to a certain
    new vertex set.}
    $C_1$ :=VC-enum-compact$(G - x, \rho, k - 1, C + x)$
    $C_2$ :=VC-enum-compact$(G - N(x), \rho, k - \deg(x), C + x_1 + \cdots + x_r)$, where
    $N(x) = \{x_1, \ldots, x_r\}$.
    if $C_1 =$ NO then
      return $C_2$
    else if $C_2 =$ NO then
      return $C_1$
    else
      return $C_1 \cup C_2$
    end if
  end if
```

Theorem 8.16 *Alg. 73 lists representations of all minimal vertex covers of size up to k (and possible some more minimal cover representations) in time* $\mathcal{O}^*(1.6182^k)$.

We will leave detailed proofs for the correctness of this algorithm to the reader. Observe, however, that the representation function ρ only comes into play via the reduction rules. The purpose of this function is to ensure that actually only minimal vertex covers are created. The problem shows up if, one after the other, the neighbors of a certain vertex x are considered to be put into a partial cover. This will first turn x into a vertex of degree one and then into an isolate. In the decision version of VERTEX COVER, we could simply discard x in that case. However, if x happened to have a neighbor of degree one in former stages of the algorithm, $\rho(x) = \hat{x}$. Then, either x or that neighbor should be in a minimal vertex cover set, provided all other neighbors of x are already in the cover. To ensure that no non-minimal covers are created, it must be prevented that x together with all of its neighbors comes into any cover that is described.

As regards running time, it is crucial to observe that any maximum degree vertex ever selected for branching will be at least two. Namely, assume a graph having maximum degree of one. Then, $N_1(x) \neq \emptyset$ is satisfied for any vertex of degree one (assuming simple input graphs as we usually do). Hence, Rule 69, followed by Rule 68, will trigger, finally eliminating all vertices from the instance.

It is relatively straightforward to generalize the above algorithm for enumerating all minimal vertex covers to an algorithm that enumerates all hitting sets; if the size of the hyperedges of the given hypergraph is bounded by some constant d, such an algorithm would run in time $\mathcal{O}^*(d^k)$, cf. [108]. A possible application of such an algorithm in the context of plan generation is discussed in Sec. 6.3.

In actual fact, the task of enumerating all minimal hitting sets has been already undertaken by Reiter [331] (also refer to the corrections in [214] and further discussions concerning the applicability of the diagnosis approach in [259]; the work of de Kleer and Williams is also pretty much related [260]). Since the mentioned papers sketch a nice application scenario of computing all minimal hitting sets (and also for the variant of finding one particular minimum hitting set), let us present this application scenario in the following.

Reiter's general starting point is the notion of a *system*, that is seen as a pair (SD,COMPONENTS), where SD is the *system description* and COMPONENTS is the finite set of system components. The system description should use some logical language and may use the distinguished unary predicate AB($\cdot$), interpreted as *abnormal*. It takes any component as an argument. The system description is intended to describe the normal behavior of the components. For example, a sentence like "Normally, an adult human's heart rate is between 70 and 90 beats per minute." could be expressed as follows in some logical language:

$$\mathrm{ADULT}(x) \wedge \text{HEART-OF}(x,h) \wedge \neg\mathrm{AB}(h) \implies \mathrm{rate}(h) \geq 70 \wedge \mathrm{rate}(h) \leq 90.$$

Then, Reiter introduces the notion of an *observation*, which is a finite set OBS of first-order sentences describing the actual behavior of a concrete system. Then, a system (SD,$\{c_1, \ldots, c_n\}$) is *faulty* if

$$\mathrm{SD} \cup \{\neg\mathrm{AB}(c_1), \ldots, \neg\mathrm{AB}(c_n)\} \cup \mathrm{OBS}$$

is inconsistent. Then, appealing to the *principle of parsimony* (akin to Occam's razor), Reiter calls a set $\Delta \subseteq$ COMPONENTS a *diagnosis* for (SD,COMPONENTS,OBS) if Δ is a minimal set of components such that

$$\mathrm{SD} \cup \{\mathrm{AB}(c) \mid c \in \Delta\} \cup \{\neg\mathrm{AB}(c) \mid c \in \mathrm{COMPONENTS} \setminus \Delta\} \cup \mathrm{OBS}$$

is consistent. Conversely, a *conflict set* for (SD,COMPONENTS,OBS) is a set $C \subseteq$ COMPONENTS such that

$$\mathrm{SD} \cup \{\neg\mathrm{AB}(c) \mid c \in C\} \cup \mathrm{OBS}$$

is inconsistent. The connection between these notions is obtained by the following theorem of Reiter:

Theorem 8.17 *$\Delta \subseteq$ COMPONENTS is a diagnosis for*

(SD,COMPONENTS,OBS)

iff Δ is a minimal hitting set for the collection of (minimal) conflict sets for (SD,COMPONENTS,OBS), where conflict sets are interpreted as hyperedges of a hypergraph with vertex set COMPONENTS.

Example 8.18 For example, consider the system depicted in Fig. 8.1. Components of this system could be: the aerial antenna, the satellite dish, the two surveying cameras, the switch between different scenes, the TV set, a couple of wires and cables, etc. Now, we won't try to specify in any formal way a system description.

What could be the cause if we cannot observe any picture on the TV screen? In principle, the failure of any of the mentioned components might result in such an observation. So, in order to make a better diagnosis, we should be more specific about this observation, or we might wish to add more observations. For example, if we can express that we only observe no picture if the switch tries to transmit a signal from the aerial, this pretty much restricts the possible set of causes. More specifically, the surveying cameras would no longer be in a minimal conflict set.

Observe that the possible restriction to minimal conflict sets in the formulation of Theorem 8.17 (as also observed by Reiter) is valid through the

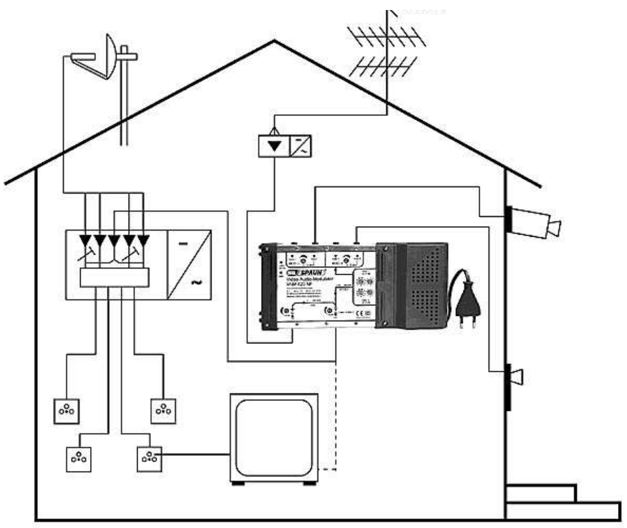


Figure 8.1: A sample system.

soundness of reduction rule 4 (edge domination) also in the case of the enumeration version of HITTING SET.

In Reiter's scenario, it makes also perfect sense to look for a minimum hitting set (resp., to enumerate all minimum hitting sets), and since a specific diagnosis also incurs a specific cost, the weighted hitting set problem shows up here, as well.

We basically left open the problem how to actually obtain a HITTING SET specification out of a system description (plus the observations). In practice, theorem provers are used as an intermediate step. This problem is explicitly addressed in [224]. In fact, explicitly Reiter does not restrict the logic to be used in the specification of the system. For example, in [355], the use of order-sorted logic is described; more details can be found in [196]. This can also include probabilistic reasoning, see [109, 263].

Let us again mention that Reiter suggests further pruning rules to improve on the computation of a hitting set tree (that basically represents all minimal hitting sets); in fact, there exists a whole sequence of paper that deals with this subject; in chronological order [333, 214, 379, 273]. It is worth mentioning that the first papers on the computation of hitting set trees use branching on edges, while [273] suggest the use of binary branching, as we also advocated in Chap. 5 for the decision problem variant. Also, alternatives to hitting set trees for computing all minimal hitting sets have been discussed in the diagnosis literature. For example, Lin and Jiang [273] report on good results using a translation of a hitting set instance into a propositional logic formula. Fijany *et al.* [181] propose the use of satisfiability and 0/1 integer programming techniques for this problem. They also describe a very good application for using a computer-assisted diagnosis approach, namely that of spacecrafts that should be able to autonomously diagnose faults. They also report on the alternative of using expert knowledge (for creating a good inference base) in this setting: the development of such a self-monitoring expert system took 20 work years for the Cassini spacecraft, and this is indeed a barely tolerable amount of time in a rapidly developing area of technology.

As a further example, observe that the theory of model-based diagnosis has also been used to solve the diagnosis problem in configuration knowledge bases, see [162]. The reader who likes to see a very concrete application might wish to study the RAPPER project [197], where the use of model-based diagnosis in photocopier service is studied.

Reiter's theory of model-based diagnosis has not only found application in the analysis (in the sense of diagnosis) of (faulty) technical systems. For example, Obst reports in [312] how model-based diagnosis can be used in spatial reasoning tasks as they show up in the Robocup (robot soccer) scenario, where in fact hypotheses about the environment (instead of a diagnosis of a faulty system) have to be generated. Further generalizations can be found in [24], where not also the diagnosis of a system but also its reliability is studied in a probabilistic framework. The relevance of model-based reasoning within the area of Artificial Intelligence can be also seen from Site [Chapter 11, Site 15].

8.2.3 Multiple parameters for CONSTRAINT BIPARTITE VERTEX COVER

We studied CONSTRAINT BIPARTITE VERTEX COVER in various places throughout this Habilitationsschrift. Let us here focus on using the technique of enumeration to place multi-parameter versions of CBVC within $\mathcal{FPT}$.

We first need some more notions for the 2-parameter version of CONSTRAINT BIPARTITE VERTEX COVER:

Let (S_1, S_2) be a pair of cover sets for our CBVC. By its *signature* we simply understand $(|S_1|, |S_2|)$. A signature $(|S_1|, |S_2|)$ (and the solution (S_1, S_2)) is called *minimal* if there is no solution (S_1', S_2') for our CBVC with $|S_1'| \leq |S_1|$ and $|S_2'| < |S_2|$ or $|S_1'| < |S_1|$ and $|S_2'| \leq |S_2|$.

Lemma 8.19 *If we consider k_1 and k_2 as fixed, then there are at most* $\min\{k_1, k_2\} + 1$ *pairwise uncomparable (minimal) signatures.*

This allows to state how to deal with graphs containing only vertices of degree at most two in a different way:

Lemma 8.20 *Let $G = (V_1, V_2, E)$ be a connected undirected bipartite graph with maximum vertex degree 2 and let $\ell = |E|$ be the number of edges in G.*

1. *If G is a cycle, then for $\ell' := \ell/2$ we have the minimal signatures $(0, \ell')$, $(\ell', 0)$ as well as $(2, \ell' - 1), (3, \ell' - 2), \dots, (\ell' - 1, 2)$ if $\ell > 4$.*
2. *Let G be a path.*
   (a) *If ℓ is odd, then for $\ell' := (\ell+1)/2$ we have the minimal signatures $(0, \ell'), (1, \ell' - 1), \dots, (\ell' - 1, 1), (\ell', 0)$.*
   (b) *If ℓ is even, then for $\ell' := \ell/2 + 1$ we have the minimal signatures $(0, \ell' - 1), (2, \ell' - 2), \dots, (\ell' - 1, 1), (\ell', 0)$ if $|V_1| > |V_2|$ and $(0, \ell'), (1, \ell' - 1), \dots, (\ell' - 2, 2), (\ell' - 1, 0)$ if $|V_1| < |V_2|$.*

Upon recalling the multi-parameter versions of CONSTRAINT BIPARTITE VERTEX COVER, it appears to be reasonable to think about separately solving the different blocks; however, if spares are shared, then it is not clear that a solution of the first block that has signature say $(2, 2)$ is better than a solution with signature $(1, 6)$, since saving one (shared) spare row for another block might be crucial to the solvability of the reconfiguration problem for the whole chip. Hence, we naturally arrive at the problem of enumerating *representative solutions* for CONSTRAINT BIPARTITE VERTEX COVER: namely, for each signature (x, y) with $x + y = k$, we store one minimal solution (where minimality is now a two-dimensional notion relating to the comparability of signature vectors) whose signature is smaller than or equal to (x, y).

In [180], a parameterized algorithm running in time less than $\mathcal{O}(1.4^{k_1+k_2}n)$ was developed for this decision problem. In fact, by analyzing the decision procedure developed in that paper one easily derives:

Corollary 8.21 *For the CBVC problem, generating one representative solution for each minimal signature can be done in time*

$$O(1.3999^{k_1+k_2}k_1k_2 + (k_1 + k_2)n),$$

where n is the number of vertices of the input graph and k_1 and k_2 are the two parameters.

Theorem 8.22 *Given a chip board with n elementary cells which is split into k_3 blocks each of which has at most k_1 neighboring spare rows and k_2*

neighboring spare columns, then a reconfiguration strategy can be found in time

$$\mathcal{O}(k_3(1.3999^{k_1+k_2}k_1k_2 + (k_1+k_2)n) + k_3(\min\{k_1,k_2\}+1)^{\sqrt{k_3}+1})$$

if it exists.

Proof. (Sketch) At first, we run the representative enumeration procedure from Corollary 8.21 for each block. Then, all possible combinations of signatures for all blocks are examined to see whether the decision problem is solvable. This second step can be implemented more efficiently by using dynamic programming techniques in a sweep-line fashion. From a graph-theoretic point of view, we exploit the fact that a grid graph (representing the local dependencies between the blocks on the chip) with k vertices has treewidth of at most $\sqrt{k}+1$, see [54] and also Chap. 7. ∎

In other words, the parameterized enumeration of representative solutions can be used in order to show that another (related) decision problem is fixed parameter tractable, considering k_1, k_2 and k_3 as parameters of the problem.

The third mentioned variation which is also incorporating linked spares seems to be harder, since knowing only one representative solution per signature is of not much help here. Even worse, also the generation of all minimal solutions (which can be done as in the case of vertex cover elaborated above) would not help, since possibly non-optimal solutions (considered "locally" for each block) would be a better choice. For example, consider the following chip with two blocks each containing three rows:

	1	2	3	4	5	6	7	8	9
1	?			?					
2									
3		?							
4				?					
5	?			?			?		
6	?			?					

For each of the two blocks, we have one spare row and, furthermore, there are two linked spare columns. If we use the linked spare columns in order to repair columns number 1 and 4, the array can be repaired by using the remaining two spare rows for row number 3 and row number 5. Only considering the first block, this solution is not minimal, since its signature $(1,2)$ is outperformed by taking, e.g., a spare row for row number 1 and one of the two linked spare columns for column number 2. However, then the second block would be not repairable with the remaining spares (one spare row and one spare column).

Only at the expense of a considerable exponential blow-up, we can show the following fixed parameter tractability result:

Theorem 8.23 *Given a chip board with n elementary cells which is split into k_3 blocks each of which has at most k_1 neighboring spare rows and k_2 neighboring spare columns and assuming that there are, furthermore, at most k_4 linked spare rows and k_5 linked spare columns on the whole board, then a reconfiguration strategy can be found in time*

$$\mathcal{O}(k_3((k_1+k_2+k_4+k_5)n+ \binom{k_3(k_1+k_4)}{k_4}\binom{k_3(k_2+k_5)}{k_5}[1.3999^{k_1+k_2}k_1k_2+(\min\{k_1,k_2\}+1)^{\sqrt{k_3}+1}]))$$

if it exists.

Proof. Such a board can be reconfigured as follows:

1. Kernelize each block assuming that there are at most k_1+k_4 spare rows and at most k_2+k_5 spare columns per block. The size of the problem kernel such obtained is $k_3(k_1+k_4)(k_2+k_5)$.

2. Consider all possible assignments of the k_4 linked spare rows to one of the $k_3(k_1+k_4)$ possibly faulty rows and all assignments of linked spare columns to possibly faulty columns and apply the algorithm sketched in the proof of the preceding theorem to each of the remaining "boards".

∎

Of course, the algorithm obtained in the previous theorem is only manageable for very small values of k_3, k_4 and k_5. Again, one might think about a weighted variant of the last considered problem (which is again solvable by considering the signatures as detailed above), since a solution using one linked spare is probably to be preferred over a solution using $\approx \sqrt{k_3}$ many individual spares.

Remark 8.24 *The example shown in this section proves that, from the point of view of applications, it might make perfect sense to consider problems with a certain number of parameters. Note that the philosophy behind the development of fixed parameter algorithms is that the involved parameters should be small in practice, and this is exactly what we expect for all five parameters occurring in Theorem 8.23.*

To conclude our discussion on variants of CONSTRAINT BIPARTITE VERTEX COVER, let us mention another issue discussed in [227]: namely, within our formalization of the reconfiguration problem, we assumed up to now that the k_1 spare rows and k_2 spare columns that are supplied on a chip are *not faulty*. Assuming some errors in the fabrication process, this assumption is of course not realistic.

In fact, the current state-of-the-art seem to be even more complicated (at first glance): Namely, if a row r and a column c are replaced by spare lines r_i and c_j, respectively, the question might arise what happens with the "central element" that is seemingly both replaced by r_i and c_j? To avoid dependencies on the sequence of replace operations (and also for technological reasons), it is assumed that this central element is replaced by the so-called *limbo* element that seemingly both belonged to r_i and to c_j (before reconfiguration). The problem now is that (in contrast to what we have seen before) we cannot anymore take any two of the spare rows (if two were required under the faultless spares assumption), since some of these spare might themselves be faulty, and even the limbo elements might be faulty, so that we have, in addition, to specify which of the spare rows we take. For a naive algorithm, this should mean that the trivial $2^{k_1+k_2}$ branching we saw before is now replaced by some $(k_1 + k_2)^{k_1+k_2}$-branching. Luckily, the situation is not so bad. Handa and Haruki proved the following theorem:

> A chip (array) M with k_1 spare rows and k_2 spare columns is reconfigurable (assuming that the spares themselves might be faulty) if and only if there is an associated chip array M' with k_1 spare rows and k_2 spare columns that is reconfigurable (assuming that the spares themselves are not faulty).

Hence, we can use the algorithms we developed so far in this scenario, as well.

8.2.4 EDGE DOMINATING SET

In this part, we show how to use the parameterized enumeration approach to solve some parameterized decision problems. These results haven't been published elsewhere.

Recall that an *edge dominating set* of a graph is a subset D of edges such that each edge is either in D or incident to an edge in D.

Problem name: EDGE DOMINATING SET (EDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $|D| \leq k$?

According to [70], a graph has a minimum edge dominating set that is also a maximal matching.

Algorithm 74 A search tree algorithm for EDS based on enumeration, called EDS

Input(s): a graph $G = (V, E)$, a positive integer k
Output(s): if possible: a subset $D \subset E$, $|D| \leq k$, that dominates all edges or
NO if no such set exists.

```
Create a list $L$ of minimal vertex covers $C$ of $G$ with $|C| \leq 2k$, using Alg. 72.
{This hides the search tree part.}
for all $C \in L$ do
  Create $G' = G[C]$.
  Find a maximum matching $M$ in $G'$.
  Let $V'$ be all vertices in $G'$ that are not covered by $M$.
  Let $D := M$.
  for all $v \in V'$ do
    Choose an edge $e \in E$ such that $v \in e$.
    Add $e$ to $D$.
  end for
  if $|D| \leq k$ then
    return $D$
  end if
end for
return NO
```

Theorem 8.25 EDGE DOMINATING SET *is in* $\mathcal{FPT}$.[6]

Proof. First, we observe that to every EDGE DOMINATING SET instance $G = (V, E)$ with a solution $D \subseteq E$ of size at most k there corresponds a vertex cover of size at most $2k$: simply take all vertices incident to the at most k cover edges, i.e., $C = \bigcup_{e \in D} e$.[7] Moreover, we can greedily find a minimal vertex cover $C' \subseteq C$.

Conversely, if $C \subseteq V$ is a minimal vertex cover of size at most $2k$ of $G = (V, E)$, we can construct an edge dominating set $D \subseteq E$ of G that satisfies the following requirements:

- $k \leq |D| \leq 2k$,
- $C \subseteq \bigcup_{e \in D} e$,

[6]A preliminary version of this result was obtained together with R. Niedermeier and U. Stege (unpublished).

[7]This observation was also the basis of a first, simple factor-4 approximation algorithm for MINIMUM EDGE DOMINATING SET presented in [70].

- D is a minimum set satisfying the first two requirements.

Consider first the isolates in $G[C]$: by minimality of C, each isolate x is necessary to cover (at least) one of the edges, so that x must be simulated by one (arbitrary) edge e_x with $x \in e_x$, and since x is an isolate, $y \notin e_x$ for each $y \in C \setminus \{x\}$. Hence, all the requirements are satisfied when restricting one's attention to isolates in $G[C]$.

If K is a (non-trivial) connected component of $G[C]$, let $M \subseteq E(K)$ be a maximum matching of K. Put M into D and treat $V(K) \setminus \left(\bigcup_{e \in M} e\right)$ as isolates (see above).

Why is the set D that is finally obtained in the way described minimum with respect to the first two requirements? A possible violation of minimality can only be caused by the treatment of non-trivial connected components. If D' is a minimal set satisfying the first two requirements, consider how it looks on a connected component K. Such a restriction D'_K collecting all edges in D' incident with vertices in K is again minimum. The incidence of an edge from D'_K with a vertex from K can be caused in two different ways: either (1) $e \in D'_K$ is an edge in $E(K) \subseteq E(G[C])$ or (2) not. By minimality of D'_K, the collection M of all edges in D'_K that are edges in $E(K)$ is a maximal matching. If M were not a maximum matching, there would be (by the well-known algorithms for maximum matching) an augmenting alternating path in K on which M could be increased to M'. This means that there are two vertices x, y in K that did not belong to any edge from M but do belong to edges from M'. These two vertices are covered in D'_K by edges e_x and e_y. e_x and e_y do not contain other vertices than x and y from C. Hence, if we make the following changes to D'_K (and hence to D'), we get an EDGE DOMINATING SET D'' satisfying the two requirements but being smaller than D', contradicting the third requirement:

- Remove M from D'_K and replace it by M'. This increases the EDGE DOMINATING SET by one.
- Delete e_x and e_y from D'_K. This decreases the EDGE DOMINATING SET by two.

Therefore, we will reduce the size of D'_K by the two operations. We can do this as long as we can find such augmenting alternating paths. Hence, a D satisfying all three requirements must be caused by maximum matchings as described.

This shows the correctness of the procedure described in Alg. 74. ∎

Observe that in the proof of the preceding theorem, we basically showed how to compute a minimum edge cover for $G[C]$.

Let us add some further comments on Alg. 74.

Remark 8.26 *Alg. 74, as listed, needs exponential space. It is however easy to interleave the enumeration algorithm Alg. 72 in a way that only one minimal vertex cover is delivered at a time. Hence, Alg. 74 can be implemented to run in polynomial space.*

In the formulation of Theorem 8.25, we were not very specific about the running time of our algorithm. But since the running time is mainly determined by the enumeration algorithm, and since the time spent on each minimal vertex cover is polynomial (especially, the involved maximum matching algorithm), we may conclude:

Remark 8.27 *Alg. 74 runs in time $\mathcal{O}^*(4^k)$.*

What about the weighted variant(s) of EDGE DOMINATING SET? More precisely, let us consider—motivated by the introduction in [70]:

Problem name: WEIGHTED EDGE DOMINATING SET (WEDS)
Given: A graph $G = (V, E)$ with edge weights $\omega : E \to \mathbb{R}_{\geq 1}$
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $\omega(D) \leq k$?

Corollary 8.28 *Alg. 74 can be modified to work for* WEIGHTED EDGE DOMINATING SET *and then runs in time $\mathcal{O}^*(4^k)$ (and may only use polynomial space).*

Proof. First observe that, by our constraints on the edge weight function ω, any edge dominating set D with $\omega(D) \geq k$ obeys $|D| \geq k$. Hence, we can use the (non-weighted) listing of all minimal vertex covers C of size up to $2k$ as before. However, when constructing the corresponding weighted edge dominating sets, we have to use a weighted variant of maximum matching. Hence, we have to compute a minimum edge cover for $G[C]$, seen as a weighted graph. The analogous details are left to the reader. ∎

The fact that the algorithm for weighted variant of EDGE DOMINATING SET basically has the same running time as the non-weighted variant should make us a bit suspicious, recalling experiences from VERTEX COVER. In fact, as regards the approximability of these problems [70], the unweighted variant has an (easy) 2-approximation (via some maximal matching), while the weighted variant only enjoys a 2.1-approximation due to some reasoning about integer linear programs.

Can we design any reduction rules that deal with small vertex degrees? If so, it might be sufficient to only list minimal vertex covers in a way that the branching x vs. $N(x)$ has a branching vector of $(1, 2)$ or better. Let us try to follow these thoughts.

Incidentally, we have to use some coloring on vertices. More precisely, a vertex x is colored red to denote that one of the edges incident to x must still go into the edge dominating set. In a certain sense, we hence generalize the original problem towards the following one:

Problem name: GENERALIZED EDGE DOMINATING SET (GEDS)
Given: A graph $G = (V, E)$, a set $R \subseteq V$ of red vertices
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $|D| \leq k$ such that all vertices from R are covered by D?

Obviously, any EDGE DOMINATING SET instance can be also interpreted as a GENERALIZED EDGE DOMINATING SET instance by considering an empty set of red vertices.

Reduction rule 70 *Delete isolated vertices that are not red. If an isolated red vertex is found, answer* NO*.*

Reduction rule 71 *If x is a vertex of degree one, incident to $e = \{x, y\}$, then do:*

1. *If $\deg(y) = 1$ or if x is red, put e into the edge dominating set we construct.*
2. *Otherwise, delete x and color y red.*

Observe that reduction rule 71 is not true for the weighted case, similarly to what we found with WEIGHTED VERTEX COVER, since it might be cheaper to actually put edges into the dominated set that would have been erased due to Rule 71 (or some variant of it).

Lemma 8.29 *The reduction rules 70 and 71 are sound.*

Proof. If we have a red isolated vertex, then there cannot be any solution to this instance of GENERALIZED EDGE DOMINATING SET. By way of contrast, an isolated vertex that is not red need not be covered, so we can simply erase it from the instance.

A red vertex must be covered, and if it is of degree one, we have to put its incident edge into the edge dominating set.

A vertex x of degree one that is not red need not be covered. However, if its only adjacent vertex is denoted by y, the connecting edge e need only be put into the edge dominating set in one circumstance: $\deg(y) = 1$. Otherwise, it would be never worse not to put e into the edge dominating set but one of its neighboring edges. To ensure that this eventually happens, y is colored red. ∎

The idea to solve GENERALIZED EDGE DOMINATING SET would be incorporating parts of Alg. 72 into Alg. 74; finally, we would have to compute a minimum edge cover for the graph induced by the minimal cover vertices (as before) plus the red vertices obtained by applying the reduction rules.

But wait a moment: since finally red vertices and cover vertices are treated alike, it would be easier to unite both types of vertices in the beginning.

This therefore gives the following rules (if we start with a usual EDGE DOMINATING SET instance:

Reduction rule 72 *Delete isolated vertices.*

Reduction rule 73 *If x is a vertex of degree one, then put its unique neighbor into the cover set under construction.*

Corollary 8.30 *Alg. 75 returns a correct answer in time $\mathcal{O}^*((1.6181)^{2k}) = \mathcal{O}^*((2.6181)^k)$, given an instance of* EDGE DOMINATING SET.

Proof. The correctness of the algorithm can be seen as before. Notice one detail: we do not insist any longer in listing only minimal vertex covers, since this does not affect the correctness and the running time of the overall algorithm.

The run time bound can be seen from Lemma 5.2, since we are basically running Alg. 20. ∎

Observe that basically the same algorithm can be obtained by first working out how to compute edge dominating sets from minimal vertex covers in compact representation as discussed above. Obviously, this approach would also meet the run time bounds of Alg. 75.

Remark 8.31 *We can even obtain a kernel for* EDGE DOMINATING SET *via the connection with* VERTEX COVER*: first of all, we could use Buss' kernelization rules to get a kernel for the hidden* VERTEX COVER *enumeration instance, see above. Together with the (naturally valid) rule that gets rid of isolated vertices in the given* EDGE DOMINATING SET *graph instance (G, k), these rules will leave us with the following: (1) a set C_1 of vertices that must be part of any minimal vertex cover set of cardinality $2k$ we are after in this first phase of the algorithm for* EDGE DOMINATING SET*; (2) a set C_2 of*

Algorithm 75 A search tree algorithm for EDS, called EDS-fast

```
Input(s): a graph $G = (V, E)$, a positive integer $k$, a set $C$ of cover vertices
  from $H$, global read access to the original graph $H$
Output(s): YES if there is a subset $D \subset E$, $|D| \leq k$, that dominates all
  edges of $H$ or
  NO if no such set exists.

  Exhaustively, apply the reduction rules 72 and 73. The resulting instance
  is called as the original one.
  if $k \leq 0$ AND $E \neq \emptyset$ then
    return NO
  else if $k \geq 0$ AND $E = \emptyset$ then
    Create $H' = H[C]$.
    Find a maximum matching $M$ in $H'$.
    Let $V'$ be all vertices in $H'$ that are not covered by $M$.
    Let $D := M$.
    for all $v \in V'$ do
      Choose an edge $e \in E(H)$ such that $v \in e$.
      Add $e$ to $D$.
    end for
    return $|D| \leq k$.
  else
    Pick a vertex $x$.
    {By the reduction rules, $\deg(x) \geq 2$.}
    if EDS-fast($G - x, k - 1/2, C \cup \{x\}$) then
      return YES
    else
      return EDS-fast($G - N(x), k - \deg(x)/2, C \cup N(x)$)
    end if
  end if
```

vertices from which we still have to select cover vertices (in order to cover $G[C_2]$). Now, it is rather straightforward to observe that $G' = G[C_1 \cup C_2]$ has (similar to $G[C_2]$) $\mathcal{O}(k^2)$ edges and vertices. It can then be shown that it is sufficient to look for a solution to (G, k) by solving (G', k).

Moreover, since edge dominating sets for graphs with maximum degree two (even for trees and more general graph structures, see [238, 383]) can be easily solved in polynomial time, we might get further improvements by applying the triviality last principle, when we restrict the branching to vertices of degree at least three.

Remark 8.32 *Relation to approximation: A* parameterized *factor-2 approximation of* EDGE DOMINATING SET *(even weighted) can be obtained with the help of parameterized* VERTEX COVER- *or* WEIGHTED VERTEX COVER-*algorithms, see [70].*

Let us finally consider a related problem, also mentioned in [383]:

Problem name: MATRIX DOMINATION SET (MDS)
Given: A $n \times n$ matrix with entries from $\{0, 1\}$, positive integer k
Parameter: k
Output: Is there a set D of one-entries in the matrix, where $|D| \leq k$, such that every other one-entry has at least one row or one column in common with some one-entry from D?

Observe that this problem can be also seen as a chess piece domination problem: interpret the matrix as a chessboard showing places where it is allowed to place a rook or where not (by having a one- or a zero-entry in the corresponding position).

Lemma 8.33 (Yannakakis/Gavril) MATRIX DOMINATION SET *can be reduced (via $\mathcal{FPT}$ reduction) to* EDGE DOMINATING SET.

The corresponding reduction is formulated in Alg. 76.

Algorithm 76 Reducing MATRIX DOMINATION SET to EDGE DOMINATING SET.

Input(s): a matrix instance (M, k) of MATRIX DOMINATION SET.
Output(s): a graph instance (G, k) of EDGE DOMINATING SET such that (M, k) is a YES-instance iff (G, k) is a YES-instance.

```
Let $C$ be the set of columns of $M$.
Let $R$ be the set of rows of $M$.
Form the vertex set $V = C \cup R$ of $G = (V, E)$.
for all $i \in R$, $j \in C$ do
  Put $\{i, j\} \in E$ iff entry $(i, j)$ of $M$ is one.
end for
```

Put in different words, MATRIX DOMINATION SET is in one-to-one correspondence to EDS, restricted to bipartite graphs. Since our solution of EDGE DOMINATING SET is based on VERTEX COVER, and the latter (in its decision version!) is known to be easier on bipartite graph, the following corollary might see some improvements; however, we did not manage to get improvements in a straightforward manner, since we are rather relying on the enumeration than on the decision version of VC.

Corollary 8.34 MATRIX DOMINATION SET *can be solved in time* $\mathcal{O}^*((2.6181)^k)$.

In [134, Exercise 3.2.9], solving MATRIX DOMINATION SET by means of a kernelization and search tree based algorithm is proposed as an exercise. However, to our knowledge there is no published solution for this problem.

In the literature, several other variants of EDGE DOMINATING SET have been considered that can be attacked with similar methods; one recent paper is [281].

8.2.5 More parameterized enumeration problems

Let us finally mention that also the topic of enumeration is not new in itself: in the literature, also many "parameterized" results can be found, although they tend to be stated not as explicit parameterized problems.

As an example, let us rephrase the main results of a paper of A. Kanevsky [250], who considered the following problem:

Problem name: SEPARATING VERTEX SETS ENUMERATION (SVS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Enumerate all minimum size separating vertex sets of size at most k.

Theorem 8.35 (Kanevsky) SVS *can be solved in time* $\mathcal{O}(2^k|V(G)|)$ *for a given graph* G. *Using* $\mathcal{O}(4^k(|V(G)|/k^2))$ *processors (in the PRAM model), we can get down to a running time of* $\mathcal{O}(k \log n)$.

The last statement of the quoted theorem is insofar interesting, as it may show a way of establishing a parameterized theory of parallel computing, a topic nearly untouched up to now.

Without giving further explicit definitions, we quote a theorem due to Kaplan *et al.* [254] that is also a parameterized enumeration result:

Theorem 8.36 *All minimal* k*-triangulations of a graph* G *can be found in* $\mathcal{O}(16^k|E(G)|)$ *time.*[8]

That paper contains two more parameterized graph modification enumeration problems. It would be interesting to see if the pure decision versions of the mentioned three problem of Kaplan *et al.* would allow for better $\mathcal{FPT}$ algorithms.

[8]The decision version of this problem is also known as MINIMUM FILL-IN.

Let us finally discuss three more example that underline why do we think that parameterized enumeration is important.

Gramm and Niedermeier [213] developed a fixed parameter algorithm for the so-called MINIMUM QUARTET INCONSISTENCY PROBLEM (MQI) which is important for constructing evolutionary trees in biology. An *evolutionary tree* is a rooted binary tree whose leaves are bijectively labeled by taxa from a set S. A *quartet* is an evolutionary tree with four leaves. A problem instance of MQI consists of an n-element set of taxa S and $\binom{n}{4}$ quartets such that, to each four-element subset S' of S, there is exactly one quartet whose leaves are labeled with taxa from S'. The aim is to construct an evolutionary tree T whose leaves are bijectively labeled by taxa from S such that the number of sub-trees of T with four leaves which are different from the input quartet with the same leaf labels is bounded by a given error bound, the parameter k of the problem. In this application, it is interesting for the human expert to see and check *all* reconstructed evolutionary trees (satisfying the given error bound) in order to choose the tree variants which appear to him to be the most reasonable choice, given his additional background knowledge on the subject. In fact, Gramm and Niedermeier already showed how to enumerate all such minimal solutions in time $O(4^k p(n))$.

Of course, the enumerated solutions could also be the basis of further computations, even as a kind of heuristic estimate. For example, some researchers interested in computing a k-dominating set of a graph heuristically assume that such a dominating set is included within a $2k$-vertex cover and use the known (comparatively fast) vertex cover algorithm (computing some $2k$-cover) in a preprocessing phase.[9] This heuristic could be naturally improved by starting off from all (minimal) vertex covers. In fact, recall that we used a similar strategy to solve EDGE DOMINATING SET.

More of theoretical gist is our last application: Marx [286, Theorem 5.3] used parameterized enumeration in order to show $\mathcal{FPT}$ membership for the k-weighted satisfiability problem for a certain class of *weighted* $\mathcal{F}$*-SAT* formula. This turned out to be a basic building block of a dichotomy theorem as developed by Schaefer for classical complexity theory [345].

8.3 Parameterized Counting

Parameterized counting has been under scrutiny of different authors, see [29, 186, 290]. Most of the space in those paper is devoted to develop a

[9]U. Stege, personal communication about a Swedish bioinformatics group

hardness theory for parameterized counting. Since the present Habilitationsschrift is rather devoted to presenting parameterized algorithmics, let us also here only report on the positive results, as especially outlined in [186].

We first give two interpretations of what is meant by "counting" along the lines of the preceding section, this way a bit deviating from the mentioned papers:

For our favorite problem VERTEX COVER, this yields the following two alternative counting problems:

1. count all vertex covers of size (exactly) k
2. count all minimal vertex covers of size (exactly) k

Observe that one could also define variants where we replace "(exactly)" in the sentences above by "(at most)". However, it is not hard to see that, at a polynomial-time expense, the corresponding problems can be solved with the help of each other.

8.3.1 Classical graph parameters in view of counting: VC

In [186, p.868], an algorithm was presented to solve the first of the mentioned tasks, and this algorithm is basically (again) a variant of Mehlhorn's search tree algorithm 16. It relies on listing all minimal vertex covers of size at most k (of the given graph $G = (V, E)$) in a cover collection $\mathcal{C}$ of G and then computing the number of all vertex covers of size exactly k by applying the *inclusion-exclusion principle* from combinatorics (see, e.g., [274, Chapter 10]), relying on the fact that the number of all vertex covers of size exactly k equals

$$|\{X \subseteq V \mid \exists C \in \mathcal{C} : C \subseteq X \wedge |X| = k\}|.$$

Besides recurring to combinatorics, we could also directly work on Mehlhorn's search tree algorithm 16, or better, its binary branching variant, Alg. 18. Let us briefly follow this idea in Alg. 77.

Why is Alg. 77 correctly working ? Obviously, we have to avoid double-counting. So, whenever we branch into the case "take x into the cover", we do not exclude the possibility of taking all neighbors of x into the cover, too. Moreover, in that branch only vertex covers that do contain x are counted. However, in the other branch, x is added to the list of excluded vertices $\mathfrak{E}$, so that we will never account for any vertex cover that contains x in this branch. Whenever the parameter was decreased to zero, all possibilities of adding vertices to the cover obtained to that point are taken into consideration that

Algorithm 77 A simple search tree based counting algorithm, called VC-count

Input(s): a graph $G = (V, E)$, a positive integer k; the original number of vertices n and a set $\mathfrak{C}$ of excluded vertices.

Output(s): the number of vertex covers of G of size exactly k

```
if $k \leq 0$ and $E \neq \emptyset$ then
  return 0
else if $k \geq 0$ and $E = \emptyset$ then
  return $\binom{n - |\mathfrak{C}|}{k}$
else
  Choose edge $e = \{x, y\} \in E$
  Let $\ell :=$VC-count$(G - x, k - 1, n, \mathfrak{C} \cup \{x\})$.
  return $\ell$+VC-count$(G - N[x], k - \deg(x), n, \mathfrak{C} \cup N[x])$
end if
```

respect the list of excluded vertices (that also includes all vertices that are taken into the cover along that particular search tree branch).

Can we do better?

Notice that counting of vertex covers has also been treated from a non-parameterized perspective. More precisely, Goldberg, Spencer and Berque have developed [207] an intricate algorithm that counts all vertex covers of size k in an n-vertex graph in time $\mathcal{O}^*(2^{.325n})$. For graphs of degree at most three, they even come up with a counting algorithm running in time $\mathcal{O}^*(2^{.2292n})$. Observe that this latter result also (incidentally) improves on the counting algorithm we presented:

Corollary 8.37 *All vertex covers of size k in an n-vertex graph G of degree at most three can be counted in time $\mathcal{O}(1.8880^k k)$.*

Proof. Since G has maximum degree of three, k vertices can cover at most $3k$ edges. Hence, G has at most $4k$ vertices. ∎

Alternatively, note that we may also apply the inclusion-exclusion principle to the enumeration algorithms for minimal vertex covers in some sort of compact form. We won't pursue this venue here. Rather, let us return once more to Alg. 77. Do we need to branch all the way down through the graph ? No, rather we could stop branching say at vertices of degree two (actually, by doing some sort of dynamic programming, we could even stop with branching at vertices of degree three). Then, we are left with a forest of matching edges. If there are say r edges left, there are obviously 2^r possibilities to cover them.

This has to be incorporated in the base case in a straight-forward manner (increasing especially $\mathfrak{C}$ by r), reducing the complexity for counting vertex covers of size k to $\mathcal{O}^*(1.6182^k)$. [10]

Let us formulate the strongest variant of the counting result as a (formally unproved; however, the details should be clear) theorem:

Theorem 8.38 *All vertex covers of size k in an n-vertex graph G can be counted in time $\mathcal{O}(1.4656^k n)$.*

Finally, we mention that the variant of counting all minimal vertex covers can be treated quite analogously, using, e.g., according parameterized enumeration results.

8.3.2 Back to the Queens

Let us mention that counting in itself is one of the topics that is first of all quite interesting in many applications and secondly only superficially attacked in the parameterized framework. We feel that there is lots of work to be done in this area. In particular here, examples can be found in the literature without explicit reference to parameterized complexity As an example, we mention the following problem, continuing this way with Example 2.18:

Problem name: COUNTING INDEPENDENT SET OF QUEENS (CQIS)
Given: An $n \times n$ chessboard C
Parameter: a positive integer n
Output: In how many ways can n queens be positioned on C such that no two of them are mutually attacking?

As a historical aside, notice that it was this counting problem that Gauß and Schumacher were after in the sequence of letters reprinted in [204, pp. 19ff.]. The history of this problem is explained in some “remarks” on pages 28–29. It is noteworthy that neither Gauß nor Schumacher found the correct solution for $n = 8$, but that particular problem was solved by an (unnamed) person who was born blind.

Rivin and Zabih [337] showed the following:

Theorem 8.39 *Given n, all solutions to* COUNTING INDEPENDENT SET OF QUEENS *can be counted in time $\mathcal{O}^*(8^n)$ (using also $\mathcal{O}^*(8^n)$ space).*

[10]P. Rossmanith is quoted in [306] to have found a counting method for vertex covers running in time $\mathcal{O}^*(1.4656^k)$, which pretty much smells like the strategy of avoiding branches at vertices up to degree two, as sketched above. However, we have not seen any publication of that result.

They actually derived basically the same result for the n-Queens problem on a *toroidal chessboard*, i.e., on a chessboard with wrap-around connections, so that every *line (on a chessboard)*, i.e., a row, a column, or a diagonal, has n squares.

Algorithm 78 A dynamic programming algorithm for CQIS

Input(s): a positive integer n

Output(s): the number $Q(n)$ of ways in which n queens can be placed on an $n \times n$ board without attacking themselves

```
Set $Q$ to $\{(\emptyset, 1)\}$.
{$Q$ should be organized as a directed acyclic graph $G_Q$ reflecting the inclusion relation of the first components; the "root" of $G_Q$ is always $(\emptyset, 1)$. A directed spanning tree of $G_Q$ is used to investigate $G_Q$ in the second FOR-loop.}
for all squares $s$ of the chessboard do
  Let $T$ be the set of (at most six) lines that contain $s$.
  for all $(S, i) \in Q$ do
    {Start the search through $G_Q$ at the root}
    if $S \cap T = \emptyset$ then
      {a queen can be placed on $s$ without attacking any of the lines that are already dominated}
      Look for some $(S \cup T, j)$ in $Q$ using $G_Q$.
      if found then
        Replace $(S \cup T, j)$ by $(S \cup T, j + i)$
      else
        Add $(S \cup T, i)$ to $Q$ (maintaining $G_Q$)
      end if
    else
      {We can skip all successors of $(S, i)$ in $G_Q$.}
    end if
  end for
end for
Let $M$ be the set of $2n$ lines that are either rows or columns.
Initiate $Q(n) := 0$.
for all $(S, i)$ in $Q$ do
  if $M \subseteq S$ then
    $Q(n) := Q(n) + i$
  end if
end for
```

Algorithm 78 reproduces a preliminary version of Rivin and Zabih's algorithm running in time $\mathcal{O}^*(64^n)$.[11] It is based on dynamic programming (although this time written up in an iterative rather than in a recursive form) and uses as the basic data structure a queue Q containing pairs (S, i). Here, S is a set of lines (i.e., either rows, columns, or diagonals) each of those has been already become a *closed line*, i.e., a queen has been put on one of the squares of it. The integer i tells us in how many ways it is possible to place queens on a chessboard that do not attack themselves mutually but dominate exactly the lines in S. So, at the start, Q contains only the pair $(\emptyset, 1)$, since there is only one way to place no queens on the chessboard. The correctness and the claimed running time is based on the following observations:

- A chessboard has $6n - 2$ lines.
- For each set of lines S, there will be never two items (S, i) and (S, i') in the queue.
- Hence, there will never be more than 2^{6n-2} elements in the queue.
- The second FOR-loop visits each element of the queue at most a polynomial number of times (namely, notice that the set T used for updating $S \cup T$ contains at most 6 elements).
- A valid solution must put one queen in each row and in each column.

In essence, this counting algorithm is based on the idea of putting possible candidate solutions into the same equivalence class if the set of closed lines is the same. The improvements of the algorithm are then based on putting even more solutions into the same equivalence class. Moreover, the order in which all squares of the chessboard are scanned is fixed, namely to row-major, so that, whenever one row r has been completely scanned, all (S, i) with $r \notin S$ are deleted from S, since a valid solution must place a queen in every row.

It has been conjectured in [336] that the number of solutions $Q(n)$ to the n-Queens Problem, divided by $n \log n$, converges to some constant β. If this conjecture were true, then Algorithm 78 would be superior to any backtracking algorithm that actually constructs (enumerates) all solutions. However, the values of $Q(n)$ listed in [336] were obtained by such a backtracking algorithm due to the (at that time) seemingly unsurmountable usage of memory of Algorithm 78 (even in its improved form). To such backtracking algorithms, "intelligence" is usually added that exploits certain patterns for creating solutions, see [156, 159, 331].

[11] In actual fact, a naive direct implementation of their algorithm **Queens** contained in [337] has (contrary to what is claimed) a running time of $\mathcal{O}^*(64^{2n})$.

Can we use the ideas from Algorithm 78 to solve similar problems, as well? Let us try to do so with DOMINATING SET OF QUEENS, i.e., a decision problem (rather than a counting problem).

Algorithm 79 A dynamic programming algorithm for QDS

```
Input(s): positive integer $n$ and $k$
Output(s): Is it possible to place $k$ queens on an $n \times n$ board to dominate
  all squares?
  Set $Q$ to $\{(\emptyset, 0)\}$.
  {$Q$ should be organized as a directed acyclic graph $G_Q$ reflecting the in-
  clusion relation of the first components; the "root" of $G_Q$ is always $(\emptyset, 0)$.
  A directed spanning tree of $G_Q$ is used to investigate $G_Q$ in the second
  FOR-loop.}
  for all squares $s$ of the chessboard do
    Let $T$ be the set of (at most six) lines that contain $s$.
    for all $(S, i) \in Q$ do
      {Start the search through $G_Q$ at the root}
      if NOT $T \subseteq S$ then
        {putting a queen on $s$ would dominate new lines}
        Look for some $(S \cup T, j)$ in $Q$ using $G_Q$.
        if found then
          Replace $(S \cup T, j)$ by $(S \cup T, \min\{j, i + 1\})$
        else
          Add $(S \cup T, i + 1)$ to $Q$ (maintaining $G_Q$)
        end if
      end if
    end for
  end for
  Let $M$ be the set of all possible lines.
  Find $(M, j)$ in $Q$.
  return $k \leq j$
```

In Alg. 79, we again maintain a queue that contains information in the form of pairs (S, i); however, this time partial solutions are encoded. More specifically, S is a set of already dominated lines, and i shows how many queens are needed to dominate S. Hence, the initialization is different: the empty set is dominated by zero queens.

What is the running time of Alg. 79? Again, since for each S, at most one element (S, i) will be present in the queue, there are at most 2^{6n} elements ever showing up in the queue at the same time. Clearly, each subset of lines

S will be modified at most $\mathcal{O}(|S|^6)$ times (choose a subset T of at most six lines from S that, together with $S \setminus T$, might cause an update). Keeping in mind Theorem 4.27, we may hence conclude:

Theorem 8.40 *Alg. 79, when started upon a reduced kernel according to Theorem 4.27, solves a* DOMINATING SET OF QUEENS *instance in time* $\mathcal{O}^*(2^{12k})$.

Observe that this is worse than what we got by the treewidth-based approach in Theorem 7.15. However, there might be ways to improve the current estimate of Alg. 79 or even combine this methodology with the treewidth-based approach.

8.4 Other techniques for $\mathcal{FPT}$ algorithms

In this section, we briefly review other techniques for obtaining $\mathcal{FPT}$ results.

Alon, Yuster and Zwick [22] introduced the idea of using *color coding* into the domain of $\mathcal{FPT}$ algorithm design. In recent papers, there has been a revival of those ideas. We only mention two of them:

- Fellows *et al.* used this technique in [168] to get improved $\mathcal{FPT}$ results (all of type $\mathcal{O}^*(c^k)$) for geometric packing and matching problems.
- Marx [286] used color coding to show $\mathcal{FPT}$ results for certain types of constraint satisfaction problems.

As already mentioned in Chap. 2, there does also exist "machinery" from graph minor theory, and more general, from well-quasi-orderings. With the exceptions reported in Chap. 7, this machinery is well suited for deriving classification results, but usually the running times of these algorithms are worse than what can be obtained by other methods. One particular example shown in this Habilitationsschrift is Theorem 4.7 that got substantially improved in terms of running time in this chapter. [12]

Other techniques based on logical characterizations of problems could also be (again) mentioned here, although the concrete solving strategy does rely on algorithmic techniques as explained throughout this Habilitationsschrift.

8.5 Implementations

Unfortunately, reports on practical experiences with parameterized algorithms are relatively scarce throughout the literature. Nonetheless, we will

[12] Actually, we refrained from giving an exact running time bound in Theorem 4.7; the involved constants are simply astronomically huge.

try to underline the validity of the parameterized approach by reporting on some of the relevant aspects. This should also inspire people working on the more practical parts of algorithmics to constantly report their results. In fact, the ultimate goal in this area might be to compile a repository of programs that implement basic parameterized algorithms. Since this Habilitationsschrift is focusing on graph-theoretic problems, an incorporation of such a repository in LEDA, see [293] and [Chapter 11, Site 11].

8.5.1 Implementations of sequential algorithms

According to our observation, most of the implementation work done so far is focusing on treewidth-based algorithms, the quotations [10, 124, 132, 133] being only a (possibly random) selection of corresponding works.

The (partly dramatic) effect of data reduction rules has been described in [7]. Of course, also the paper of Weihe [375] deserves to be mentioned in this place, although it is not directly referring to parameterized algorithmics. In Sec. 4, we also mentioned the experiments of [3] when considering crown reductions for VERTEX COVER.

Recently, Hüffner reported [243] about an implementation of the bipartization algorithm we also presented in Alg. alg-GBP. He also relates that algorithm with the iterative compression idea briefly mentioned in Chap. 4.

As regards search tree algorithms, the reported experiments we are aware of [210, 357] mostly show that the worst case running time estimates that were theoretically derived for the according algorithms are rarely met in practice; the algorithms tend to behave far better than “expected” both on artificial and on real data. Of course, this might mean that the derived estimates are simply not good enough. Alternatively, it could mean that the worst case scenario that could be drawn from the theoretical analysis rarely shows up in practice. This might also encourage research on the average case complexity of exact, exponential-time algorithms, an area that is (to our knowledge) not even attempted at up to now.

8.5.2 Parallel implementations

Works on the parallelization of search tree algorithms have been reported in b[80]. More precisely, the algorithm they implemented for VERTEX COVER has two phases:

1. The first phase is basically executed to obtain instances that can be easily dispatched on different processors on a predetermined level of the overall search tree. This technique requires that the search tree is quite

regular. Therefore, in [80] even a worse than best-known sequential algorithm is chosen in this phase, based on [33, Theorem 1].

2. In the second phase, a rather up-to-date search tree algorithm (based on [33, Theorem 2]) is chose to be executed on each processor to work on the subproblems originating from the dispatcher phase one.

Such a coarse-grained parallelization of a search tree algorithm is a quite natural idea. In our context, it is interesting that the authors of [80] chose two different algorithms for the two processing phases, since they needed a simple branching behavior in the first phase. This is interesting, since the top-down approach to search tree algorithms we advocate yields particularly simple algorithms, so that for example in the case of HITTING SET, it would not be necessary to twice encode the same algorithm to be run on the different phases. The fact that in the binary branching of our HITTING SET algorithms, some branches will (nearly) keep the parameter value should not matter too much, since the seemingly more work-intensive jobs can be dispatched first, leaving the dispatching processor finally with the instance with the largest parameter. This strategy should help balance the overall workload.

In the case of VERTEX COVER, this would mean that we propose to use algorithm VCMH-BU as described in Sec. 5.1.1.

In [79], the authors report on how to solve VERTEX COVER instances through the addition of parallelism, thereby allowing even larger problem instances to be solved in practice. They implemented a parallel $\mathcal{FPT}$ method for VERTEX COVER using C and the MPI communication library, and tested it on a PC cluster. This has been the first experimental examination of parallel $\mathcal{FPT}$ techniques. The authors tested this method on protein sequences obtained from the National Center for Biotechnology Information. As part of the experiments, they solved larger instances of VC than in any previously reported implementations. For example, it is mentioned that problem instances with $k \geq 400$ in less than 1.5 hours can be exactly solved. Since the parallel $\mathcal{FPT}$ algorithm requires only very little communication between processors, it is expected that the method to also perform well on grids.

Chapter 9

Limitations to parameterized algorithmics

In this chapter, we rather briefly discuss limitations to parameterized algorithmics.

1. We will see that parameterizing problems does not always help overcome the $\mathcal{P}$ versus $\mathcal{NP}$ problem. More specifically, we will introduce (some aspects of) the parameterized hardness theory developed to show that—most likely—certain parameterized problems are not parameterized tractable.

2. We will show that for dualizable parameterized problems with linear kernels, the kernel sizes could not be "arbitrarily small."

We will also encounter many of the problems we have seen before on this way.

9.1 Parameterized intractability

In actual fact, there are now a certain number of ways to formalize the notion of parameterized intractability.

Since we believe that Turing machines—albeit being amongst the first formulations of computability—are still one of the models many computer scientists are most familiar with, we are going to present the most important classes of parameterized intractability "in the Turing way," following [74]. Some results in this direction can also be found in [90]. In [74], it is also shown that this approach is very useful and natural to prove that certain problems are belonging to a specific complexity class.

In this way, we will describe the basic levels of the *W-hierarchy*

$$\mathcal{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots \subseteq \mathrm{W}[\mathrm{SAT}] \subseteq \mathrm{W}[\mathrm{P}].$$

The examples from this section have not yet appeared anywhere else, although some of the results have been at least known for a long time.

9.1.1 W[1]

W[1] can be characterized by the following problem on Turing machines:

Problem name: SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION (SNTMC)
Given: A (single-tape) nondeterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

Formally, we can state:

Definition 9.1 A parameterized problem P with parameter k is in W[1] iff there is a function f and, for every instance I of P, a nondeterministic Turing machine $T_{P,I}$ such that $T_{P,I}$ halts within $f(k)$ steps, writing a one onto its tape, iff (I, k) is a YES-instance of P.

This means that you can prove both hardness (with respect to W[1]) and containment within W[1] by finding according parameterized reductions.

The question whether $\mathcal{FPT}$ equals W[1] has therefore the same philosophical backing as the question whether $\mathcal{P}$ equals $\mathcal{NP}$: it seems to be impossible to deterministically simulate a nondeterministic Turing machine that uses k steps within $f(k)$ steps on a deterministic Turing machine.

To illustrate this approach, let us show the following lemma:

Lemma 9.2 INDEPENDENT SET *is in W[1].*

Proof. Let $G = (V, E)$ be an instance of INDEPENDENT SET. We have to transform it into an instance (T, k') of SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION. We also assume that $k > 0$ ($k = 0$ is a trivial instance).

Let T have V as its set of non-blank tape symbols.

In a first phase, T guesses exactly k different symbols from V and writes them onto its tape. This can be done as follows: In a preliminary sub-phase,

T guesses k symbols from V and writes them onto its tape, using k steps. Let us now assume an ordering on V being fixed. Then, T can run a bubble-sort algorithm, needing $b(k)$ steps, where in the end the head of T is scanning the first blank symbol to the right of the k symbols from V. By another sweep to the left, T can verify in k more steps that actually all guessed symbols are different.

In a second phase, T will once more scan the guessed symbols and now see if they are independent. More specifically, T will perform, upon scanning a symbol v, perform the following steps:

- Memorize v in finite memory and erase v on the tape.
- Move to the right until you find the first blank symbol. On the way, check if any $u \in N(v)$ has been found. If such a $u \in N(v)$ is found, reject.

If and only if the guessed vertex set was an independent set, no rejection case can be found. Hence, the Turing machine can then accept. Assuming that this second phase takes $c(k)$ time, we can estimate the overall running time of T (assuming a correct guess) to be $t(k) = 2k + b(k) + c(k) \in \mathcal{O}(k^2)$. More precisely, the guessed vertex set is an independent set of size k iff T halts (in an accepting state) after having run $t(k)$ steps. ∎

The reader will have noticed that the Turing machine described in the preceding proof could have been designed to be more efficient. For example, we could get rid of all sub-phases but the guessing phase in the first phase if we checked if $u \in N[v]$ can be found during the second phase. However, we think that the present description is more illuminating since it clearly separates the two steps that are typical for Turing machines that should show membership in W[1]:

- In a first phase, a possible solution is guessed.
- In a second phase, this guess is checked; here, it is crucial to know that the instance of the original problem (as such) could be hard-wired into the Turing machine table that is constructed.

As an aside, let us mention that the necessity of this hard-wiring is essential for the questions if certain problems are in W[1] or rather (already) in $\mathcal{FPT}$.

Remark 9.3 *Another problem whose membership in W[1] can be seen in a similar fashion is* VERTEX INDUCED FOREST. *We only indicate the necessary modifications in what follows. After having guessed the k would-be forest* *vertices, only acyclicity has to be tested, i.e., for each pair of forest vertices, it has to be examined if there is at most one path between them (in the graph induced by the guessed vertices). This can be done in time $\mathcal{O}(k^3)$.*

Let us now consider the following variant of SNTMC:

Problem name: SHORT NONDETERMINISTIC SMALL TURING MACHINE COMPUTATION (SNSTMC)
Given: A (single-tape) nondeterministic Turing machine M whose size is bounded by $f(k)$, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

More precisely, Thm. 2.4 can be re-interpreted in the present context in the following way:

Corollary 9.4 *A problem P is in $\mathcal{FPT}$ if and only if it can be solved with the help of* SHORT NONDETERMINISTIC SMALL TURING MACHINE COMPUTATION*, for some memory delimiting function f.*

This is the basic content of [75, Corollary 6].

9.1.2 W[2]

W[2] can be characterized by the following problem on Turing machines:

Problem name: SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION (SMNTMC)
Given: A multi-tape nondeterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

This means that you can prove both hardness (with respect to W[2]) and containment within W[2] by finding according reductions.

To show these ideas, let us present one simple construction that shows that RED-BLUE DOMINATING SET (and hence related problems like HITTING SET) belong to W[2].

Lemma 9.5 RED-BLUE DOMINATING SET *belongs to W[2].*

Proof. Let $(G = (V_{red} \cup V_{blue}, E), k)$ be an instance of RED-BLUE DOMINATING SET. We have to transform it into an instance of SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION.

In fact, we won't give a completely formal description of the Turing machine but rather describe the ideas. Details of the construction can be found in [74, p.664ff.] in a slightly different setting (namely, STEINER TREE IN GRAPHS). We also assume that $k > 0$ ($k = 0$ is a trivial instance).

The corresponding Turing machine T has $|V_{blue}| + 1$ tapes; let they be indexed by $\{0\} \cup V_{blue}$. As tape symbols, we will use V_{red} on tape 0. The edge relation of G is "hard-wired" into the transition function of T as described below.

T will first nondeterministically guess the red-blue dominating set and write it on tape 0 using the letters from V_{red}. Since T need not worry about guessing the same vertex more than once (this would only mean that the corresponding RED-BLUE DOMINATING SET has less than k elements, which is fine), this guessing phase takes k steps.

In a second phase, T has to verify that the guess is correct. To this end, upon reading symbol $v \in V_{red}$ on tape 0, T

- will move the heads corresponding to tapes addressed by vertices from $N(v)$ one step to the right, and
- will move the head on tape 0 one step to the right.

Upon reading the blank symbol on tape 0, T will move all heads one step to the left; only if this is possible for all of the heads, T will accept.

The second phase will take another $k + 1$ steps.

It is now easy to see that G has a red-blue dominating set of size at most k iff T has an accepting computation within $2k+1$ steps, so that we actually described a parameterized reduction. ∎

So, the basic idea to show that a certain problem is in W[2] is to find a way in which a nondeterministic multi-tape Turing machine can solve it by making use of its many tapes, where the actual (original) problem instance (I, k) is usually stored within the finite memory of the Turing machine, and then only $f(k)$ steps may be performed by the Turing machine. The only thing that always needs some caution is that the Turing machine table can be written up in a succinct way, i.e., polynomial in I.

In the preceding proof, the only delicate thing to note here is that, in the second phase, all but the tape labeled 0 will contain a blank symbol under their heads, so that there is actually only one transition per symbol in V_{red} that has to put into the Turing table (plus one transition for the final step).

So, to be more specific, the Turing machine constructed in the preceding proof will contain $(k-1)|V_{red}|$ transitions for phase one (it has to count up to k within its finite memory), plus $|V_{red}|+1$ transitions for phase two.

More generally speaking, the mentioned condition will be met if all but a fixed number of tapes are only worked by using the head positions for counting, or if it is at least clear what symbols to expect under the corresponding heads. Otherwise, the corresponding Turing tables might be of size $\Omega(2^{|I|})$.

A similar but slightly more complicated proof for showing that DOMINATING SET is contained in W[2] is given in [75, Theorem 6], following basically the same ideas; simplifications are contained [90, Cor. 1]. We mention in passing that also variants like INDEPENDENT DOMINATING SET can be treated this way. Similarly, the problem of finding a *digraph kernel* as examined in [221] can be shown to lie in W[2] by very closely following these lines. Note that membership in W[2] of that problem was left open in [221].

Let us give one more example of this technique:

Lemma 9.6 ROMAN DOMINATION *is in W[2].*

The proof is actually pretty similar to the preceding one.

Proof. Let $G=(V,E)$ be an instance of ROMAN DOMINATION. We have to transform it into an instance of SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION. We also assume that $k>0$ ($k=0$ is a trivial instance).

The corresponding Turing machine T has $|V|+1$ tapes; let they be indexed by $\{0\}\cup V$. As tape symbols, we will use $(V\times\{1,2\})$ on tape 0. The edge relation of G is "hard-wired" into the transition function of T as described below.

T will first nondeterministically guess the Roman domination function R and write it on tape 0 using the letters from $V\times\{1,2\}$ as follows: whenever $R(v)=1$, it will write $(v,1)$, and if $R(v)=2$, then it will write $(v,2)$. The values $R(v)=0$ are implicitly stored (by having no (v,i) written on the tape 0).

If we look a bit more closely how the second phase works (described below), we will see that if it happens that by these nondeterministic guesses both $(v,1)$ and $(v,2)$ are written onto tape 0, we will see that such a notation is interpreted as two $(v,2)$ write-ups would be. Hence, T need not worry about picking some vertex v more than once for writing either $(v,1)$ or $(v,2)$ onto tape 0 (this would only mean that the corresponding RED-BLUE DOMINATING SET has less than k elements, which is fine). However, the Turing machine should care about whether (or not) the second components of the elements stored on tape 0 sum up to a number that is at most k. This check can be done "in parallel" with the guessing itself, storing intermediate sums in finite memory. Therefore, this guessing phase takes k steps and uses up a polynomial number of transitions in the Turing table.

In a second phase, T has to verify that the guess is correct. To this end, upon reading symbol $(v, 1)$ on tape 0, T

- will move the head on the tape addressed by v one step to the right, and
- will move the head on tape 0 one step to the right.

Upon reading $(v, 2)$, T

- will move the heads on to tapes addressed by vertices from $N[v]$ one step to the right, and
- will move the head on tape 0 one step to the right.

Upon reading the blank symbol on tape 0, T will move all heads one step to the left; only if this is possible for all of the heads, T will accept.

The second phase will take another $k + 1$ steps.

It is now easy to see that (G, k) is a YES-instance to ROMAN DOMINATION iff T has an accepting computation within $2k + 1$ steps, so that we actually described a parameterized reduction. ∎

More discussion on hardness of decision problems related to Turing machines can be found in [75]. In particular, note that if we allow multiple heads in addition or instead of multiple tapes for a Turing machine, we get a problem similar to SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION that is also W[2]-complete.[1]

Let us give one example of a hardness proof within the W-hierarchy, based on the fact (see [134]) that RED-BLUE DOMINATING SET is W[2]-complete (with no bound on the degrees of the blue vertices).

To prove the hardness result, we need one fact about the Roman domination of complete graphs.

Lemma 9.7 *For the complete graph K_n on n vertices, the Roman domination number is two iff $n \geq 2$.*

[1]More precisely, the theorems from [75] only give hardness of these problems. In order to prove that the corresponding problem(s) are in W[2], a technique similar to the one applied in the proof of [74, Theorem 3] can be used.

Proof. The mapping that assigns two to an arbitrary vertex of K_n and zero to all other vertices is obviously a valid Roman domination function.

If one is assigned to any vertex by some Roman domination function for K_n, this function can only meet the claimed Roman domination number of two if $n = 2$, since then to the other vertex, one could be assigned, as well. Otherwise, to one further vertex the value of two would have to be assigned, so that this Roman domination function would be worse than the one given in the first paragraph. ∎

Theorem 9.8 Roman domination *is W[2]-complete.*

Proof. Due to Lemma 9.6, we already know that Roman domination lies in W[2].

Assume that $G = (V, E)$ is an instance of red-blue dominating set, restricted to bipartite graphs (see Lemma 2.14), i.e., $V = V_{\text{red}} \uplus V_{\text{blue}}$.

Without loss of generality, we can assume that $|V_{\text{red}}| > 1$, since otherwise an optimal solution to the given RBDS instance can be found in polynomial time and can be accordingly translated into an equivalent Roman domination instance.

In the simulating Roman domination instance, we construct a graph $G' = (V', E')$, where

$$V' = (V_{\text{red}} \cup \{1, \dots, 2k+1\}) \times \{1, \dots, k\} \cup V_{\text{blue}},$$

and E' contains the following edges (and no others):

1. $G'[V_{\text{red}} \times \{i\}]$ is a complete graph for each $i \in \{1, \dots, k\}$.

2. For all $i \in \{1, \dots, k\}$ and $x \in V_{\text{red}}$, $y \in V_{\text{blue}}$, $\{x, y\} \in E$ iff $\{[x, i], y\} \in E'$.

3. For all $i \in \{1, \dots, k\}$, $j \in \{1, \dots, 2k+1\}$ and $x \in V_{\text{red}}$: $\{[x, i], [j, i]\} \in E'$.

We are going to show the following claim: G has a red-blue dominating set D of size k iff G' has a Roman domination function R with $\sum_{x \in D_R} R(x) = 2k$.

If G has a red-blue dominating set $D = \{d_1, \dots, d_k\}$ of size k, then consider the following function $R : V' \to \{0, 1, 2\}$: R assigns zero to all vertices but to $d_i' = [d_i, i]$, to which R assigns two. Since d_i is connected to all vertices in $(V_{\text{red}} \cup \{1, \dots, 2k+1\}) \times \{i\}$, the vertices in $V' \setminus V$ are all dominated by this assignment. Moreover, since D is a red-blue dominating set of G, all vertices in V_{blue} are dominated in G', as well.

Now consider a Roman domination function R for G' with $\sum_{x \in D_R} R(x) = 2k$. Due to Lemma 9.7 and according to the first condition on edges, the Roman domination number of each induced graph $G'[V_{\text{red}} \times \{i\}]$ is two, assuming $|V_{\text{red}}| > 1$. Since $G'[V_{\text{red}} \times \{1, \dots, k\}]$ can be decomposed into k components, the Roman domination number of $G'[V_{\text{red}} \times \{1, \dots, k\}]$ is $2k$. More specifically, to achieve that bound, the domination function would have to assign two to one vertex from $V_{\text{red}} \times \{i\}$ for each i and zero to all other vertices. Observe that such an assignment would be also a valid Roman domination function R' for $G'[(V_{\text{red}} \cup \{1, \dots, 2k+1\}) \times \{1, \dots, k\}]$ if we assign zero to all vertices from $\{1, \dots, 2k+1\} \times \{1, \dots, k\}$.

Since there are "too many" vertices in $\{1, \dots, 2k+1\} \times \{1, \dots, k\}$, we cannot simply replace one or more vertices to which R' assigns two by vertices from $\{1, \dots, 2k+1\} \times \{1, \dots, k\}$ to which R' (as constructed) had assigned zero.

Observe that we have left over yet some degrees of freedom for finally constructing a valid Roman domination function R from R'; namely, we have not been specific about how to choose a vertex from $V_{\text{red}} \times \{i\}$ (for each i) to which we assign two. However, if we find k assignments of two to vertices from $V_{\text{red}} \times \{1, \dots, k\}$ such that also all vertices from V_{blue} are dominated, i.e., $D_R = \{[d_1, 1], \dots, [d_k, k]\} = R^{-1}(\{2\})$, then $D = \{d_1, \dots, d_k\}$ is a valid dominating set of G.

Since there are no connections between vertices from $\{1, \dots, 2k+1\} \times \{1, \dots, k\}$ and V_{blue}, there is no way of replacing some of the vertices selected from $(V_{\text{red}} \cup \{1, \dots, 2k+1\}) \times \{1, \dots, k\}$ (by assigning two to them) by vertices from V_{blue}, so that there cannot be a Roman domination function R that assigns one or two to any of the vertices from V_{blue} without violating the condition $\sum_{x \in D_R} R(x) = 2k$. Hence, the Roman domination function (tentatively) constructed above is the only possibility; that construction works if and only if G has a dominating set of size k. ∎

Let us finally mention one further problem, also taken from [318]; in fact, some more (and similar) problems can be found there and treated alike.

Problem name: DOMINATING REARRANGEMENT (DR)
Given: A graph $G = (V, E)$, a subset $S \subseteq V$
Parameter: a positive integer $k = |S|$
Output: Is there a *dominating rearrangement* $r : S \to N[S], s \mapsto r(s) \in N[s]$ such that $r(S) \subseteq V$ is a dominating set?

Again, this problem can be viewed from a military perspective: S is the set of locations where currently armies are placed on, and the question is if by a one-step rearrangement of each army (if necessary) a situation can be created in which each region (modeled by graph vertices) is sheltered by either a defending army in the region itself or in a neighboring region.

This problem is interesting for at least two reasons from a parameterized perspective:

- The parameterization is not standard. Actually, there is no obvious optimization problem hidden here, although the variant that only k out of the vertices from S may be actually rearranged gives another obviously W[2]-complete problem.
- It is a specific example of a problem where the problem is if a given "solution" can be improved in order to satisfy a certain property, here: being a dominating set. Such type of problems can show up in many disguises in practice, we are sure.

Theorem 9.9 DOMINATING REARRANGEMENT *is W[2]-complete.*

Proof. Membership in W[2] can be seen by a guess-and-verify strategy as seen before. We don't give details but only mention that it appears to be best to take $S \times V$ as the alphabet on tape 0, where (s, v) is indicating that an army on vertex s has been moved to v.

For the hardness, take again an instance $(G = (V = V_{red} \cup V_{blue}, E), k)$ of RED-BLUE DOMINATING SET. Let $S = \{1, \dots, k\}$ be disjoint from V, and consider the graph $G' = (V', E')$ with $V' = V \cup E$ and $E' = E \cup (S \times V_{red})$. Hence, $G'[S \cup V_{red}]$ forms a complete bipartite graph. This gives the instance (G', S) of DOMINATING REARRANGEMENT. Obviously, $D \subseteq V_{red}$ is a dominating set of size (at most) k iff (G', S) can be solved by moving $|D|$ of the armies in S onto the vertices from D. ∎

9.1.3 Beyond W[2]

In [74], also a Turing machine characterization for the natural upperbound W[P] of the W-hierarchy was obtained. Cesati used the following problem, which actually uses a different parameterization for nondeterministic Turing machines:

Problem name: BOUNDED NONDETERMINISM TURING MACHINE COMPUTATION (BNTMC)
Given: A (single-tape) nondeterministic Turing machine M, an input string x, an integer n coded in unary
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most n steps and uses at most k nondeterministic steps?

A Turing machine problem for W[SAT] was obtained in [76]. They considered the following problem:

Problem name: COMPACT DETERMINISTIC TURING MACHINE COMPUTATION (CDTMC)
Given: A deterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that visits at most k squares?

They showed that COMPACT DETERMINISTIC TURING MACHINE COMPUTATION is W[SAT]-hard. Whether or not it is belonging to W[SAT] seems to be open. Nonetheless, this problem appears to be interesting, since it is a natural parameterized analogue to the classical complexity class $\mathcal{PSPACE}$ collecting all problems that can be solved on a deterministic Turing machine using only polynomial space. So, in a sense the question if COMPACT DETERMINISTIC TURING MACHINE COMPUTATION belongs to W[1] is the parameterized analogue to whether $\mathcal{P}$ coincides with $\mathcal{PSPACE}$.

In fact, this intuitive relation was used to translate a $\mathcal{PSPACE}$-hardness proof of a motion planning problem into a W[SAT]-hardness result for that problem. Details can be found in [76]. In this context, it might be interesting to observe that for other motion planning problems, membership in $\mathcal{FPT}$ had been shown [178], also confer Sec. 3.1.2.

We refrain from giving the (rather complicated circuit) definition of each level of the W-hierarchy here but refer to the book [134]. For the purpose of this Habilitationsschrift—putting specific emphasis on parameterized algorithmics—it should be enough to know the downsides of parameterized complexity as presented.

Let us briefly mention that there does exist a machine characterization of each level W[t] of the W-hierarchy in terms of machines, based on so-called W-RAMs. The interested reader is referred to [90]. There, it is shown that each level W[t] can be characterized by t alternations in (alternating) W-RAMs.

An alternative to the W-hierarchy was suggested by M. Grohe, the so-called *A-hierarchy* based on model checking [215]. It is still an open questions if both hierarchies coincide. Observe that a characterization of all levels of the W-hierarchy (corresponding to the W-RAM characterization mentioned above) by alternating Turing machines is open. More specifically, Chen and Flum defined in [90] a complexity class L[t] basically corresponding to alternating Turing machines with t alternations. It is known that W[t] $\subseteq$ L[t] $\subseteq$ A[t].

9.1.4 More downsides

In fact, the whole area of intractability in the parameterized sense is still in a developing state. Recent advances include (and this is not meant to be a complete account):

- M. Fellows [165, 164] and his co-authors developed a hardness theory that can be seen as an alternative to the W-hierarchy that we presented above. The basic class, called *MINI-1* or *M[1]*, is based upon the miniaturization version of some classical problem. The following is an example of an M[1]-hard problem:

  Problem name: MINIATURIZED VERTEX COVER (VCMINI)
  Given: A graph $G = (V, E)$
  Parameter: a positive integer k
  Output: Is there a *vertex cover* $C \subseteq V$ with $|C| \leq k \log(|V|)$?

  It is known that $\mathcal{FPT} \subseteq$ M[1] $\subseteq$ W[1].

  Showing M[1]-hardness is often simpler than proving W[1]-hardness, although the basic message is almost the same: it it commonly not believed that $\mathcal{FPT}$ equals M[1] due to the following result.

  Theorem 9.10 (Cai and Juedes [69]) *$\mathcal{FPT}$ =M[1] iff n-variable* 3-SAT *instances can be solved in time $2^{o(n)}$.*

  The theory of miniaturization was further developed by Chen and Flum [91] who exhibited a whole hierarchy of "miniaturized complexity classes" between $\mathcal{FPT}$ and W[1]. They also accordingly generalized Theorem 9.10.

- Let us mention the work of J. Chen, X. Huang, I. A. Kanj, and G. Xia [83] who developed new techniques for deriving very strong computational lower bounds for a class of well-known $\mathcal{NP}$-hard problems, including WEIGHTED SATISFIABILITY, DOMINATING SET, HITTING SET, CLIQUE, and INDEPENDENT SET. For example, although a trivial enumeration can easily test in time $\mathcal{O}(n^k)$ if a given graph of n vertices has a clique of size k, they prove that unless an unlikely collapse occurs in parameterized complexity theory, the problem is not solvable in time $f(k)n^{o(k)}$ for any function f, even if we restrict the parameter value k to be bounded by an arbitrarily small function of n. Similar results have been reported in [81]. In particular, it is shown that any problem in the syntactic complexity class $\mathcal{SNP}$ as defined in [319] (that class includes optimization problems like VERTEX COVER and INDEPENDENT SET) can be solved in time $n^{o(k)}$ if CLIQUE can be solved in time $n^{o(k)}$.

- In a similar spirit, Cai and Juedes [69] excluded the possibility of $O(2^{o(\sqrt{k})})$-algorithms for PLANAR VERTEX COVER, PLANAR DOMINATING SET and many other problems on planar graphs, since this would entails that $\mathcal{MAXSNP}$-complete problems are all solvable in time $\mathcal{O}(2^{o(k)}p(n))$.
- We already mentioned the relations of $\mathcal{FPT}$ and approximability as described in [84]; of course, we can also read these connections in a way that non-approximability results imply non-$\mathcal{FPT}$-results.

9.2 How small can kernels get ?

In previous chapters, we have often seen some sort of races to get better and better parameterized algorithms. Is there a particular "end" to this kind of races? Unfortunately, we actually do not know any way to limit these races in the area of search tree based algorithms. It has been even indicated in [211] that by more and more (computer-assisted) sophisticated analysis, it would be possible to continually improve on search tree algorithms (and on the bases of the estimates that delimited the exponential parts of the running times of such algorithms).

Another point here may be the approach of Weyer [377] that introduces more fine-sliced complexity classes to look into $\mathcal{FPT}$ from a structural perspective; however, even then it is unclear how to rule out say an $\mathcal{O}(1.3^k p(n))$ algorithm once an $\mathcal{O}(2^k p(n))$ algorithm is known for a certain problem, and it is this kind of race that is usually going on in parameterized algorithmics. The only thing that has been successfully shown (see the previous section) is that certain problems cannot have $\mathcal{O}^*(c^{o(k)})$ or $\mathcal{O}^*(c^{o(\sqrt{k})})$ algorithms, providing reasonable complexity assumptions. This is surely a good starting point for future research.

In the area of devising kernelization algorithms, the situation is different. In a recent paper [174], whose content relevant to this chapter also appeared in [82], we were able to derive lower bounds on kernel sizes. We will sketch this approach in the following.

In a certain sense, kernels are the essential ingredient of parameterized algorithmics, since a problem is fixed-parameter tractable iff it admits a problem kernel, see Thm. 2.4. The smaller the problem kernel, the "more tractable" is the corresponding problem. As we will see in this section, we cannot hope for arbitrarily small kernels for $\mathcal{NP}$-hard problems (unless $\mathcal{P} = \mathcal{NP}$), especially if both primal and dual problem (as discussed in Sec. 3.5, a section the reader might to revise) are fixed-parameter tractable.

Lemma 9.11 *If* $(\mathcal{P}, \text{size}())$ *is a parameterized problem with size function and if* $\mathcal{P}$ *admits a kernelization* K *such that* $\text{size}(K(I,k)) \leq \alpha k$ *for some* $\alpha < 1$*, then* $\mathcal{P}$ *is in* $\mathcal{P}$*.*

Proof. By assumption, $\text{size}(I', k') \geq k'$ for each instance (I', k') according to our definition of a size function. This is in particular true for the parameter k' of the problem kernel instance $I' = r(I,k)$. So, $k' \leq \alpha k$ for some $\alpha < 1$. Repeatedly kernelizing, we arrive at a problem with arbitrary small parameter and hence of arbitrarily small size. Basically, we need $\mathcal{O}(\log k)$ many such kernelizations, each of them requiring polynomial time. Hence, the classical language $L_c(\mathcal{P})$ (as defined in Sec. 2.1) can be decided in polynomial time. ∎

Remark 9.12 *The assumption* $\text{size}(I,k) \geq k$ *is crucial here. As a concrete "counterexample," consider the* DECISION TREE PROBLEM*, specified by* n *objects* $X = \{x_1, \dots, x_n\}$*,* t boolean test*s* $T = \{t_1, \dots, T_t\}$ *and a parameter* k*. In this setting, a decision tree is a binary tree* B *whose leaves are (uniquely) labeled with objects and whose inner nodes are labeled with tests such that on the path from the root to the leaf labeled* x_i *tests are performed that uniquely distinguish* x_i *from all other objects. As cost function, the overall length of all paths from the root to each leaf is usually considered. The question is if there exists a decision tree with cost bounded by* k*. This problem is known to be* $\mathcal{NP}$*-complete, see [245].*

If say $n = 2^\ell$*, the decision tree with optimal cost is surely the complete binary tree (possibly not attainable with the given set of tests), since it is optimally balanced. Hence,* $k > n \log_2 n$ *(else, an algorithm can simply answer* NO*); this can be seen as a trivial kernelization algorithm. Therefore,* $n \in o(k)$*. Conversely, this can be interpreted as giving the (to our knowledge) first natural parameterized problem having a sub-linear kernel.*[2] *On the other hand, this relation also implies that* $\text{size}(I,k) < k$ *is true here, so that the previous lemma does not lead to a contradiction with the known* $\mathcal{NP}$*-hardness result.*

More specifically, the problem here is the seemingly innocuous choice of the size function as being $n = |X|$*. Observe that any "reasonable" encoding of an instance would rather use* $n \log n$ *bits, since each element of* X *would need to get a name. This way, the seeming problem would disappear.*

[2]Also confer our discussions of kernelization schemes.

We now discuss the case when both the primal and the dual version of a problem admits a linear kernel.

Theorem 9.13 *Let $\mathcal{P}$ be an $\mathcal{NP}$-hard parameterized problem with size function. If $\mathcal{P}$ admits an αk-size kernel and its dual $\mathcal{P}_d$ admits an $\alpha_d k_d$-size kernel ($\alpha, \alpha_d \geq 1$), then*

$$(\alpha - 1)(\alpha_d - 1) \geq 1$$

unless $\mathcal{P}$ equals $\mathcal{NP}$.

Proof. Let $r(\cdot)$ denote the assumed linear kernelization reduction for $\mathcal{P}$. Similarly, $r_d(\cdot)$ is the linear kernelization for $\mathcal{P}_d$. Consider Alg. 80 for a reduction R, given an instance (I, k) of $\mathcal{P}$:

Algorithm 80 A reduction R based on combining the kernelizations $r(\cdot)$ and $r_d(\cdot)$.

```
if $k \leq \frac{\alpha_d}{\alpha+\alpha_d}\text{size}(I)$ then
  compute $r(I, k)$
else
  compute $r_d(I, \text{size}(I) - k)$
end if
```

For the size of the R-reduced instance I', we can compute:

- If $k \leq \frac{\alpha_d}{\alpha+\alpha_d}\text{size}(I)$, then $\text{size}(I') \leq \alpha k \leq \frac{\alpha\alpha_d}{\alpha+\alpha_d}\text{size}(I)$.
- Otherwise,

$$\begin{aligned}
\text{size}(I') &\leq \alpha_d k_d \\
&= \alpha_d(\text{size}(I) - k) \\
&< \alpha_d\left(\text{size}(I) - \frac{\alpha_d}{\alpha + \alpha_d}\text{size}(I)\right) \\
&= \frac{\alpha\alpha_d}{\alpha + \alpha_d}\text{size}(I)
\end{aligned}$$

By repeatedly applying R, the problem $\mathcal{P}$ is solvable in polynomial time, if $\frac{\alpha\alpha_d}{\alpha+\alpha_d} < 1$. (Details are very similar to Lemma 9.11.) ∎

Remark 9.14 *For the following examples, it is handy to rewrite the conclusion of Theorem 9.13 as*

$$\alpha \geq \frac{\alpha_d}{\alpha_d - 1}.$$

From the previous theorem, we can immediately deduce a couple of corollaries, *always assuming that $\mathcal{P}$ is not equal to $\mathcal{NP}$.*

Corollary 9.15 *For any $\epsilon > 0$, there is no $(4/3 - \epsilon)k$ kernel for* PLANAR VERTEX COVER*.*

Proof. Recall that PLANAR INDEPENDENT SET has a $4k_d$ kernel due to the four-color theorem, see Chap. 4. ∎

This "negative result" immediately transfers to more general graph classes in the following manner: If there is any way to produce a kernel smaller than $4/3k$ for VERTEX COVER on general graphs, then the corresponding reduction rules must "somehow" possibly introduce $K_{3,3}$ or K_5 as subgraphs (or as minors) into the reduced instance.

Namely, assume that there were a kernelization algorithm which does not introduce $K_{3,3}$ or K_5 as subgraphs (or as minors) into the reduced instances. Then, this would also be a kernelization algorithm for PLANAR k-VERTEX COVER, since it would be planarity preserving due to Kuratowski's theorem. Therefore, Cor. 9.15 applies.

Unfortunately, we cannot conclude that there is no $(4/3k - \epsilon)$-kernel for the general VERTEX COVER problem.

Corollary 9.16 *For any $\epsilon > 0$, there is no $(2 - \epsilon)k_d$ kernel for* PLANAR INDEPENDENT SET*.*

*Likewise, there is no $(2-\epsilon)k_d$ kernel for k_d-*INDEPENDENT SET ON GRAPHS OF MAXIMUM DEGREE BOUNDED BY THREE*.*

*This is even true for the combination problem (which is still $\mathcal{NP}$-hard): There is no $(2 - \epsilon)k_d$ kernel for k_d-*INDEPENDENT SET ON PLANAR GRAPHS OF MAXIMUM DEGREE BOUNDED BY THREE*.*

Proof. The general k-VERTEX COVER has a $2k$ kernel based on a Theorem due to Nemhauser and Trotter [86, 304], see Chap. 5. For our purposes, it is enough to know that that rule identifies a subset of vertices V', $|V'| \leq 2k$ of the given graph instance $G = (V, E)$ and a parameter $k' \leq k$ such that G has a k-vertex cover iff the induced subgraph in $G[V']$ has a k'-vertex cover. Since the class of planar graphs, as well as the class of graphs of a specified bounded degree, are closed under taking induced subgraphs, the claims are true by Theorem 9.13. ∎

Based on a theorem due to Grötzsch (which can be turned into a polynomial-time coloring algorithm; see [72, 219]) it is known that planar triangle-free graphs are 3-colorable. This implies:

Lemma 9.17 *k_d-*INDEPENDENT SET*, restricted to planar triangle-free graphs, has a $3k_d$ kernel.*

Corollary 9.18 *For any $\epsilon > 0$, there is no $(1.5 - \epsilon)k$ kernel for k-*VERTEX COVER RESTRICTED TO TRIANGLE-FREE PLANAR GRAPHS.

Observe that the Nemhauser/Trotter kernelization preserves planarity and triangle-freeness, so that we actually know of a $2k$ kernel for this particular restriction of VERTEX COVER.

This last corollary interesting due to the following result that can be found in even more restricted form in [371, Chapter 7].

Lemma 9.19 k-VERTEX COVER RESTRICTED TO TRIANGLE-FREE PLANAR GRAPHS *is $\mathcal{NP}$-hard.*

Remark 9.20 *Since "Euler-type" theorems exist for graphs of arbitrary genus g, it can be shown that there is a constant c_g such that each graph of genus g is c_g-colorable. Hence, according lower bounds for kernel sizes of k-*VERTEX COVER ON GRAPHS OF GENUS *g can be derived. For triangle-free graphs of genus g, Thomassen has shown that the corresponding constant c'_g is in $\mathcal{O}(g^{1/3}(\log g)^{-2/3})$, see [205, 368].*

Remark 9.21 *Alber and Fiala have provided a linear kernel for* INDEPENDENT SET ON DISKS GRAPHS*, where the concrete factor determining the kernel size depends on the entities α and λ that—strictly speaking—determine the class of disk graphs under consideration. More precicely, the kernel size (measured in terms of number of vertices of the graph) is upperbounded by*

- $36(\alpha/\lambda)^2 k_d$ *if* $\lambda \leq 2$ *and by*
- $9\alpha^2 k_d$ *if* $\lambda > 2$.

In other words, if α is close to 1 and $\lambda = 2$, their approach nearly yields a $9k_d$-kernel for INDEPENDENT SET ON DISKS GRAPHS*, which can be opposed to the $2k$-kernel for* VERTEX COVER.

Corollary 9.22 *There is no $(67/66-\epsilon)k_d$ kernel for* PLANAR k_d-NONBLOCKER *for any choice of $\epsilon > 0$.*

Proof. A $67k$ kernel for PLANAR DOMINATING SET is sketched in Chapter 4. Hence, the lower bound follows. ∎

Corollary 9.23 *For any $\epsilon > 0$, there is no $(2 - \epsilon)k$ kernel for* PLANAR DOMINATING SET.

This is also true when further restricting the graph class to planar graphs of maximum degree three (due to the known $\mathcal{NP}$-hardness of that problem). For a proof, we can simply refer to Cor. 4.24.

Remark 9.24 *Recently, Fomin and Thilikos [189] were able to extend the linear kernel result for* DOMINATING SET *to graphs on surfaces of arbitrary genus. Therefore, our lower bound result extend to these more general graph classes, as well.*

The presented results open up a completely new line of research:

- Can we find examples of problems such that the derived kernel sizes are optimal (unless $\mathcal{P}$ equals $\mathcal{NP}$)?
- If not, can we close the gaps more and more? According to our previous discussion, PLANAR VERTEX COVER ON TRIANGLE-FREE GRAPHS is our "best match:" we know how to derive a kernel of size $2k$ (due to Nemhauser & Trotter), and (assuming $\mathcal{P} \neq \mathcal{NP}$) we know that no kernel smaller than $1.5k$ is possible.
- Are there other, possibly more sophisticated arguments for showing lower bounds on kernel sizes? Especially, it would be interesting to have arguments ruling out say the existence of a kernel of size $o(k^3)$ in a situation when a kernel of size $\mathcal{O}(k^3)$ has been obtained. The kind of algebra we used in the proof of Theorem 9.13 does not extend.
- Although we are only able to derive results for problems where both the primal and the dual parameterization allow for linear size kernels, this might already give a good starting point, especially for graph problems. Observe that many $\mathcal{NP}$-hard graph problems are still $\mathcal{NP}$-hard when restricted to the class of planar graphs. However, in the planar case, our general impression is that linear bounds can be obtained due to the known linear relationships amongst the numbers of edges, faces and vertices.

Chapter 10

The non-parameterized view

The idea to exactly solve computationally hard problems rather than to use approximation or (meta-)heuristic algorithms is not particularly new. However, there seems to be a revived interest in this area in recent times, as exemplified by the survey papers of Schöning and of Woeginger [378, 346].

A classical example is the MAXIMUM INDEPENDENT SET algorithm(s) due to Robson [342, 343] which are the last in a series of papers dealing with improving on the complexity of search-tree algorithms for MAXIMUM INDEPENDENT SET, see [246, 362] amongst others. We refrain from giving any details here, but be assured that the corresponding algorithms are far from trivial, also see Site [Chapter 11, Site 18].

Let us merely state the currently best result [343]; to list the corresponding algorithm in reasonable space is close to impossible, since the whole report [343] is basically a description of just this one algorithm.

Theorem 10.1 MAXIMUM INDEPENDENT SET *can be solved in time* $\mathcal{O}^*(1.1893^n)$ *for* n*-vertex graphs.*

In this context, also the enumeration version—how to efficiently generate all maximal independent sets—deserves to be mentioned, see [269, 370]. A good overview on (not only exact) algorithms handling MAXIMUM CLIQUE and hence also MAXIMUM INDEPENDENT SET can be found in [56]. For restricted graph classes, better algorithms are known. For instance, in the case of graphs whose degree is bounded by three, Chen *et al.* obtained on $\mathcal{O}^*(1.152^n)$ algorithm [89].

Why are these problems interesting in the context of parameterized algorithmics? There are indeed a couple of reasons for dealing with them in this Habilitationsschrift.

- If one looks at how the results in non-parameterized exact algorithmics are stated, they can be directly interpreted as parameterized results, as well. For example, already the title of Jian's paper on MAXIMUM INDEPENDENT SET shows that the complexity is examined with respect to the number of vertices of the graph. This is indeed a valid graph parameter: at least for graphs with relatively many edges it makes a thorough difference whether one aims at an $\mathcal{O}(c_1^n)$ algorithm or at an $\mathcal{O}(c_2^m) = \mathcal{O}(c_2^{n^2})$ algorithm. We also refer to the discussion on parameterizations in Chapter 3.

- The techniques used to develop non-parameterized algorithms are quite similar to the usual parameterized approach. In particular, both reduction rules and search tree techniques are employed.
- Sometimes, parameterized algorithms can be used to speed up non-parameterized algorithms (as subroutines) or non-parameterized algorithms may speed up parameterized algorithms. Examples treated in this Habilitationsschrift are: MINIMUM 3-HITTING SET (treated below) and NONBLOCKER SET.
- The technique with which lower bounds on kernel sizes were derived in Section 9.2 can be also interpreted as a way to develop non-parameterized algorithms from parameterized ones. This will be explained below.

Examples for non-parameterized exact algorithms for hard problems fall in different categories: since we mostly focus on graph-theoretic questions, we give a (surely incomplete) list of such papers in the following:

- 3-COLORING: Is a given graph colorable with three colors? [40]
- MAXIMUM INDEPENDENT SET, as discussed above. Due to the intimate relationship with MAXIMUM CLIQUE, also the corresponding exact algorithms for that problem are worth mentioning here [71].
- MINIMUM DOMINATING SET, see [187].
- MINIMUM 3-HITTING SET, see [373].
- MAXIMUM CUT, see [160].
- MINIMUM FEEDBACK VERTEX SET, see [188].

Lot of work on exact algorithmics is also devoted to logic (satisfiability); we already mentioned the "parameterization" by the number of clauses for

MAXIMUM SATISFIABILITY in the work of Gramm, Hirsch, Niedermeier and Rossmanith [212] in Chap. 3.

Although all exact algorithms for hard problems that can be found in the literature can be somehow interpreted as parameterized results, we will pay special focus in this chapter on the interplay between the parameterized and the nonparameterized view on problems. Observe that we have already found this idea in some places throughout this Habilitationsschrift, e.g., in Chap. 8 when discussing parameterized counting.

10.1 Dealing with DOMINATING SET

10.1.1 Looking for dominating sets in vertex cover sets

To underline the connections to parameterized algorithmics, let us mention one of the results from [187] in more details here. They consider (amongst others) the following problem:

Problem name: DOMINATING SET, GIVEN VERTEX COVER NUMBER (DSVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k such that G has a vertex cover of size k
Output: Is there a minimum *dominating set* $D \subseteq V$?

Fomin, Kratsch and Woeginger were able to derive the following result:

Theorem 10.2 DOMINATING SET, GIVEN VERTEX COVER NUMBER *can be solved in time* $\mathcal{O}^*(3^k)$*, where* k *upperbounds the size of a vertex cover for the given graph instance.*

The corresponding algorithm is presented in Alg. 81. Regarding its running time, the computing of a minimum vertex cover set is comparatively negligible. Observe that one call of Alg. 69 costs $\mathcal{O}^*(2^{k-|X|})$, so that the overall running time can be estimated by

$$\sum_{i=1}^{k} \binom{k}{i} \mathcal{O}^*(2^{k-|X|}) = \mathcal{O}^*(3^k).$$

This result is particularly interesting from the viewpoint of graph parameters as discussed in Chapter 7, since it also links the domination number and the vertex cover number of a graph. This appears to be a particularly interesting form of alternative parameterization, see Chapter 3.

Algorithm 81 How to solve DOMINATING SET, GIVEN VERTEX COVER NUMBER, a procedure called DSwithVC

Input(s): a graph $G = (V, E)$, an upperbound k on the size of the minimum vertex cover for G

Output(s): a minimum dominating set D for G

```
Find a minimum vertex cover $C$ of $G$, by using one of the algorithms
presented in Chapter 5.
$D := V$
for all $X \subseteq C$ do
  $Y := V \setminus (C \cup N(X))$;
  $Z := N[X] \cap (V \setminus C)$;
  $C' := C \setminus (N[X] \cup N[Y])$;
  Compute a vertex set $Z' \subseteq Z$ of minimum cardinality subject to $C' \subseteq$
  $N[Z']$ with the help of Alg. 69.
  {More specifically, the hypergraph $H$ is derived as follows: the set of
  hyperedges corresponds to $C'$, and there are at most $|C - X| \leq k - |X|$
  many of them; the vertices of the hypergraph are $N[X] \cap (V \setminus C)$; a
  hyperedge $h$ (as element of $C'$) "collects" all vertices that are neighbors
  to $h$ in $G$.}
  $D' := X \cup Y \cup Z'$
  if $|D'| < |D|$ then
    $D := D'$
  end if
end for
```

10.1.2 Some remarks on bipartite graphs

Theorem 10.2 has the following nice consequence for non-parameterized algorithmics, since every bipartite n-vertex graph has a vertex cover of size at most $n/2$.

Corollary 10.3 DOMINATING SET *can be solved in time* $\mathcal{O}^*((\sqrt{3})^k)$ on bipartite graphs.

This corollary is interesting, since it shows the difference between bipartite and general graphs from a parameterized perspective. Namely, recall the following facts:[1]

- VERTEX COVER is $\mathcal{NP}$-complete but in $\mathcal{FPT}$ on general graphs.

[1]For the results on VC, we also refer to Chapter 5.

- VERTEX COVER can be solved in polynomial time on bipartite graphs by matching techniques.
- The two-parameter version CONSTRAINT BIPARTITE VERTEX COVER (on bipartite graphs) is $\mathcal{NP}$-complete but in $\mathcal{FPT}$.

This somehow contrasts with the corresponding results for DOMINATING SET. To formally state this summary, let us first formally introduce the following problem:

Problem name: CONSTRAINT BIPARTITE DOMINATING SET (CBDS)
Given: A bipartite graph $G = (V_1, V_2, E)$
Parameter: positive integers k_1, k_2
Output: Is there a *dominating set* $D \subseteq V_1 \cup V_2$ with $|D \cap V_i| \leq k_i$ for $i = 1, 2$?

As an addendum to Chapter 9, let us demonstrate the following fact:

Lemma 10.4 CONSTRAINT BIPARTITE DOMINATING SET *is W[2]-hard.*

Proof. It is easy to solve RED-BLUE DOMINATING SET with the help of CONSTRAINT BIPARTITE DOMINATING SET by setting one of the parameters to zero. ∎

- DOMINATING SET is $\mathcal{NP}$-complete and W[2]-complete on general graphs.
- DOMINATING SET on bipartite graphs is in $\mathcal{FPT}$.
- The two-parameter version CONSTRAINT BIPARTITE DOMINATING SET (on bipartite graphs) is W[2]-hard.

10.1.3 Edge domination revisited

However, it need not be always the case that bipartiteness helps alleviate the complexity of graph problems. For example, reconsider the problem EDGE DOMINATING SET. Observe that the construction of Lemma 8.33 can be kind of reversed if the EDGE DOMINATING SET instance is bipartite. More precisely, the matrix of the MATRIX DOMINATION SET instance will correspond to the (bipartite) adjacency matrix $A_B(G)$ of the (bipartite) EDGE DOMINATING SET instance G. As derived in Sec. 8.2.4, both EDS and MDS are in $\mathcal{FPT}$ (and both problems are $\mathcal{NP}$-hard). However, a solution to MATRIX DOMINATION SET was obtained by relating it to EDGE DOMINATING SET, so that it it quite imaginable that we could get better algorithms for MATRIX DOMINATION SET as a problem on its own right, i.e., it might be that finally the parameterized algorithmics for the bipartite case is better than for the general case. Let us finally mention that a "two-parameter version" (as defined for VERTEX COVER and for DOMINATING SET above) is not meaningful for EDGE DOMINATING SET, since it is not vertices that are constrained.

Another issue, however, deserves mentioning here: recall that a *line graph* is a graph $L(G)$ whose edge relation comes from another graph $G = (V, E)$ by interpreting E as a vertex set and putting an edge (in $L(G)$) between e_1 and e_2 iff e_1 and e_2 share a vertex in G. It has been observed in many occasions that problems that are hard on general graphs may become easier on line graphs. For example, VERTEX COVER in line graphs is better known as EDGE COVER and hence solvable in polynomial time. This is also true for DOMINATING SET, since DS on line graphs is nothing else than EDGE DOMINATING SET in disguise.

10.1.4 Back to NONBLOCKER SET

In this section, we show how to use the exact results of Fomin, Kratsch and Woeginger [187] to improve parameterized algorithms, in our case NONBLOCKER SET that was tackled in Chap. 4.

More precisely, let us first recall the result of [187] on general graphs:

Theorem 10.5 MINIMUM DOMINATING SET *can be solved in time* $\mathcal{O}^*(1.9379^n)$ *on arbitrary* n*-vertex graphs.*

Due to the $5/3 \cdot k_d$-kernel for NONBLOCKER SET—see Cor. 4.22, we conclude:

Corollary 10.6 *By applying the algorithm of Fomin, Kratsch and Woeginger [187] to solve* MINIMUM DOMINATING SET *to a reduced instance* (G, k_d) *of* NONBLOCKER SET, *this problem can be solved in time* $\mathcal{O}^*(3.0121^{k_d})$.

Can we also use Cor. 10.3 for bipartite graphs? We must be careful here, since our reduction rules for NONBLOCKER SET do not preserve bipartiteness as stated. The problem is that the catalytic vertex would be merged with either type of vertices from the bipartition. Hence, the solution consists in putting up separate catalytic vertices for each bipartition and accordingly modified reduction rules. We leave the details to the reader. However, it is clear that we can apply the result of Blank, McCuaig, and Shepherd [50, 291] to conclude:

Corollary 10.7 *By applying the algorithm of Fomin, Kratsch and Woeginger [187] to solve* MINIMUM DOMINATING SET *on bipartite graphs to a reduced instance* (G, k_d) *of* NONBLOCKER SET*, restricted to bipartite graphs, this problem can be solved in time* $\mathcal{O}^*(2.4932^{k_d})$.

Let us finally mention that due to the fact that the kernel we got for NONBLOCKER SET is really small, it might be actually worthwhile looking into an algorithm that uses exponential space, as explained in details in [77, 306] in the case of VERTEX COVER. The basic algorithm would then be Alg. 82. It is rather straightforward to compute the cut-off value α: namely, one has to balance the time spent to compute the entries of OPT against the savings of the actual search-tree part. If c is the exponential base of the algorithm of Fomin, Kratsch and Woeginger, then the initialization needs

$$\sum_{j=1}^{\alpha k_d} c^j \binom{5/3 \cdot k_d}{\alpha k_d} \approx c^{\alpha k_d + 1} \binom{5/3 \cdot k_d}{\alpha k_d}$$

time. Letting $\ell = \alpha k_d$, Lemma 4.1 gives

$$\binom{5/(3\alpha) \cdot \ell}{\ell} \approx (5/(3\alpha))^\ell \left(\frac{5/(3\alpha)}{5/(3\alpha) - 1}\right)^{(5/(3\alpha)-1)\ell}.$$

Since we stop the branching of the “usual” search-tree algorithm when the graph has αk_d vertices or less, the corresponding time has shrunk down to $\mathcal{O}^*(c^{(5/3-\alpha)\cdot k_d})$. Depending on c (whether or not we solve the problem on bipartite graphs, general graphs, …) we get different cut-off values that are computed by equating:

$$(5/(3\alpha))^{\alpha k_d} \left(\frac{5/(3\alpha)}{5/(3\alpha) - 1}\right)^{(5/(3\alpha)-1)\alpha k_d} = c^{(5/3-2\alpha)\cdot k_d}.$$

The important thing that must be satisfied when applying this technique is that the search-tree algorithm that is used does only create subinstances that are vertex-induced subgraphs. Otherwise, the preprocessing on small instances would not work.

Let us take a first shot at this equation. Guessing $\alpha = 1/6$, we compute 1.7192^{k_d} as an estimate for the left-hand side (roughly the preprocessing cost) of the equation, while we get 2.4161^{k_d} for the right-hand side. We observe two things: first of all, this approach may be worthwhile doing in terms of time complexity, since even the worse of both expressions is well below the $3.\ldots^{k_d}$ (more precisely, we get an $\mathcal{O}^*(2.6978^{k_d})$ estimate for the running time of that algorithm) we obtained when restricting ourselves to polynomial space.

However, both sides of the equation are still some way apart. More precisely, we spend much more time in the main processing than in the preprocessing. Therefore, we might afford further raising the fraction α.

With $\alpha \approx 0.2724$, we can conclude:

Corollary 10.8 *By using exponential space,* NONBLOCKER SET *can be solved in time (and space)* $\mathcal{O}^*(2.5154^{k_d})$.

Similar computations can be performed for the bipartite case.

Algorithm 82 A sketch of an exponential-space algorithm for NONBLOCKER SET

```
Input(s): a graph $G = (V, E)$, $k_d$ giving the size of the nonblocker set we
  are after
Output(s): YES iff $G$ has a nonblocker set of size $k_d$

  Kernelize according to Alg. 9.
  {Let $k_d$ be also the new parameter value and $G = (V, E)$ the graph.}
  Determine cut-off value $\alpha$.
  for all $X \subseteq V$ of size at most $\alpha k_d$ do
    Determine maximum nonblocker set with the algorithm of Fomin,
    Kratsch, and Woeginger. Put its size in a table OPT($X$).
  end for
  Do the branching with the algorithm of Fomin, Kratsch, and Woeginger
  up to graphs with at most $\alpha k_d$ many vertices.
  For small graphs, look the solution up within OPT.
```

10.2 A nonparameterized view on 3-HITTING SET

When ignoring the specific role of the parameter k in 3-HITTING SET, we could of course use the algorithm(s) sketched in Sec. 5.3.3. Measured in terms of the number of vertices n, this would hence yield a straightforward estimate of roughly $\mathcal{O}^*(2.2^n)$ in order to find a minimum 3-hitting set, since $k \leq n$.

However, it is immediately clear that we can do better by simply testing all $\mathcal{O}^*(2^n)$ possible hitting sets by brute force. Wahlström came up with a nice algorithm that considerably improves on the two algorithms sketched so far; interestingly enough, it does rely on having a fast algorithm for the (standardly) parameterized problem.

10.2.1 Connections to Logic

Let us (before actually reviewing Wahlström's algorithm) point to the connections to a problem from logic:

Problem name: SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (VARIABLE PARAMETERIZATION) (3-SAT (VARIABLE))
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X, each clause having at most three literals
Parameter: $|X|$
Output: Is there a satisfying assignment $\alpha : X \to \{0, 1\}$ for F?

Monien and Speckenmeyer [296] have developed an exact algorithm for 3-SAT (VARIABLE) that runs in time $\mathcal{O}(1.619^n)$, where n is the number of variables of the given formula. One first step (as explained in [266, 346]) towards their algorithm that already breaks the trivial $\mathcal{O}(2^n)$ algorithm is explained in Alg. 83. In that algorithm, $F[x = 1]$ denotes the formula obtained from F by setting the variable x to 1 (true) and simplifying F accordingly, i.e., deleting clauses that are satisfied through this assignment of x and by deleting the negation of x from all (other) clauses. Similarly, $F[x = 0]$ is understood.

The recurrence that describes the corresponding search tree size is:

$$T(n) \leq T(n-1) + T(n-2) + T(n-3) \leq 1.8393^n.$$

Hence, we are already improving on the naive $\mathcal{O}(2^n)$ estimate. The reason for this improvement is basically the "binary variable branching" that the algorithm does. There are obviously simple reduction rules dealing with variables that only occur once in the whole formula:

Reduction rule 74 *If F is a formula in which x occurs only once, then reduce to $F[x = 1]$ if the literal that contains x is positive, and reduce to $F[x = 0]$ if the literal that contains x is negative.*

This means that in a reduced formula each variable occurs at least once. This is true in particular for the x we choose in F according to Alg. 83. Hence, in the case $F[x = 1 - \ell_x]$, we have at least *two* clauses that only contain two variables. A straightforward (binary) branching according to the settings of the involved four variables yields the following search tree size estimate:

$$T(n) \leq T(n-1) + T(n-3) + 2T(n-4) + T(n-5) \leq 1.7944^n.$$

Algorithm 83 A simple search tree algorithm for 3-SAT (VARIABLE), called DSATv-simple

Input(s): a Boolean formula F in conjunctive normal form (CNF), with variables X, each clause having at most three literals
Output(s): YES iff a satisfying assignment for F exists

```
if $F$ is trivial then
  return according solution
else
  Choose some clause $C$ containing the variables $x, y, z$.
  {Possibly, $C$ contains less variables; this only makes the branching better.}
  {Let $\ell_x = 1$ if the literal that contains $x$ (in $C$) is positive and $\ell_x = 0$ if the literal is negative; similarly, $\ell_y$ and $\ell_z$ are understood.}
  if DSATv-simple($F[x = \ell_x]$) then
    return YES
  else if DSATv-simple($F[x = 1 - \ell_x][y = \ell_y]$) then
    return YES
  else
    return DSATv-simple($F[x = 1 - \ell_x][y = 1 - \ell_y][z = \ell_z]$)
  end if
end if
```

Further improvements (in particular according to the *autarkness principle*) are rather specific to SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (VARIABLE PARAMETERIZATION) and hence omitted here. The interested reader is referred to [296, 266]. We only mention that already Monien and Speckenmeyer's analysis uses a sort of auxiliary "parameter" analysis (preferring branches on small clauses) which is however quite special-purpose and rather intricate in itself. Refinement of these techniques has led to an $\mathcal{O}(1.5045^n)$ algorithm for 3-SAT (VARIABLE) in [266]. Let us also mention that there have been also truly parameterized approaches to satisfiability and similar problems. We only mention two groups of papers here:

- Szeider [358] has parameterized SATISFIABILITY by the so-called maximum deficiency of a formula. More precisely, if the maximum deficiency over all subsets of a formula F is at most k, then one can decide in time $\mathcal{O}(2^k n^3)$ whether F is satisfiable or not. Here, the *deficiency* of a propositional formula F in CNF with n variables and m clauses is defined as $m - n$. He also compares this parameterization with graph parameters (e.g., treewidth) that are belonging to (hyper-)graphs related to SAT (as it is also explained in this section), also see [359]. This treewidth approach is also investigated in [167] for a different kind of SATISFIABILITY problem.

- Marx [286] has extended Schaefer's dichotomy theorem from classical to parameterized complexity; in this context, he investigated which Boolean constraints put the WEIGHTED SATISFIABILITY problem in $\mathcal{FPT}$, where the *weight* of an assignment is the number of variables that are set to true.

10.2.2 Back to 3-HITTING SET

But let us return to SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (VARIABLE PARAMETERIZATION). What has this problem to do with our problem 3-HITTING SET ? In fact, in the explanations and rules used up to now we barely ever used the actual "logical" properties of SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION); rather, we were looking for an assignment that "hits" all clauses. Hence, it is easy to convert (even the improved variant of) Alg. 83 into a 3-HITTING SET algorithm; a pseudocode for this algorithm can be found in Alg. 84. The reduction rule we are referring to (and that is substituting Rule 74) is:

Reduction rule 75 *Let $G = (V, E)$ be an instance of* 3-HS. *If x is a vertex of degree one, then reduce to $G - x = (V \setminus \{x\}, \{e \setminus x \mid e \in E\} \setminus \{\emptyset\})$.*

Algorithm 84 A simple search tree algorithm for 3-HITTING SET, called THS-simple

Input(s): a hypergraph $G = (V, E)$, each edge having at most three vertices
Output(s): the size k of a minimum hitting set $C \subset V$ of G.

```
Reduce $G$; the reduced instance is also called $G$.
if $G$ is trivial then
  return according solution
else
  Choose some edge $e$ with the least number of vertices, containing the
  vertices $x, y$ (and possibly $z$).
  $k_1$:=THS-simple($G - x$);
  $k_2$:=THS-simple(($V \setminus \{x\}, E \setminus \{e \in E \mid x \in e\}$);
  return min($k_1, k_2 + 1$)
end if
```

Observe that once the binary branching was performed at an edge with three vertices, then the two subsequent branches on each search tree path will be at edges of size two; further vertices might be treated with Rule 75. Hence, we may state:

Lemma 10.9 *Alg. 84 solves* 3-HITTING SET *in time* $\mathcal{O}(1.7944^n)$.

Wahlström then used more reduction rules (actually basically the same as we presented in Chap. 2 for the parameterized version of the problem) and refined heuristic priorities to further lower the base of the run time estimate. Moreover, he used another mathematical property that we like to present in the following, since it shows how parameterized algorithms can be used to improve on non-parameterized algorithms, which is indeed an interesting application of parameterized algorithmics.

Theorem 10.10 (Wahlström) *Let* $G = (V, E)$ *be a hypergraph where each edge has size (exactly) three. Assume that* $|E| \leq \delta n$*, where* $n = |V|$*. Then, the size of a minimum hitting set for* G *is upperbounded by*

$$\left\lceil \frac{6\delta + 1 - \sqrt{12\delta + 1}}{6\delta} n \right\rceil .$$

For example, if each vertex has degree of at most three, then there cannot be more than n edges in a hypergraph where each edge has size (exactly) three. Hence, the size of a minimum hitting set is upperbounded by

$$\lceil \frac{7 - \sqrt{13}}{6} n \rceil \leq 0.5658n.$$

We can therefore use our earlier derived parameterized algorithm for 3-HITTING SET, parameterized by $k = \lceil 0.5658n \rceil$, to improve on the running time of our non-parameterized algorithm for 3-HS by introducing the following heuristic priorities:

H0-THS Select a vertex of degree at least four for branching.

H1-THS Among the vertices of degree four, choose one that is contained in an edge of size two (if possible).

If it is *not* possible to find a high-degree vertex according to H0-THS, then we would use the parameterized 3-HITTING SET algorithm. What is the overall running time of such an algorithm?

- For the cases that are solved with the parameterized algorithm, we find a search tree size of roughly $\mathcal{O}(2.1788^{.5658n}) = \mathcal{O}(1.5537^n)$.

- Otherwise, we get the following formula for the search tree size:

$$T(n) \leq T(n-1) + \sum_{\ell=0}^{4} \binom{4}{\ell} T(n-5-\ell)$$

  Hence,

$$\begin{aligned} T(n) &\leq T(n-1) + T(n-5) + 4T(n-6) + \\ &\quad 6T(n-7) + 4T(n-8) + T(n-9) \\ &\leq 1.7184^n. \end{aligned}$$

This gives an obvious improvement over the bound we have derived before. Since the two estimates for the two cases are still some way apart, we can make the same reasoning for hypergraphs of degree upperbounded by four: then, $\delta = 4/3$. Hence, the parameterized algorithm would be run for $k = .6097n$, giving a time of $\mathcal{O}(1.6078^n)$. Similarly, hypergraphs of degree upperbounded by five yield $\delta = 5/3$. Hence, we run the parameterized algorithm for $k = .6418$, giving a time of $\mathcal{O}(1.6485^n)$. Conversely, the estimate for the running time of the non-parameterized algorithm (part) will further drop, assuming a minimum degree of six:

$$T(n) \leq T(n-1) + \sum_{\ell=0}^{6} \binom{6}{\ell} T(n-7-\ell)$$

Hence,

$$\begin{aligned} T(n) &\leq T(n-1) + T(n-7) + 6T(n-8) + 15T(n-9) \\ &\quad +20T(n-10) + 15T(n-11) + 6T(n-12) + T(n-13) \\ &\leq 1.6919^n. \end{aligned}$$

We stop here our analysis but only quote the following theorem of Wahlström that is based on a more sophisticated use of heuristic priorities and reduction rules:

Theorem 10.11 (Wahlström) 3-HITTING SET *can be solved in time* $\mathcal{O}(1.6538^n)$.

Observe that that algorithm is relying on the parameterized 3-HITTING SET algorithm of Niedermeier and Rossmanith; taking ours instead (see the discussion in Chap. 5) improves the running time further down to $\mathcal{O}(1.617^n)$.[2]

[2]personal communication of M. Wahlström

10.3 A dual approach to eliminate a parameter

It is natural that algorithms developed in the parameterized framework can also be used to solve the "non-parameterized" versions of the problem, in many cases simply by possibly testing all parameter values. As shown in the case of solving the INDEPENDENT SET PROBLEM ON GRAPHS OF MAXIMUM DEGREE THREE, sometimes upper bounds on the possible parameter values are known. In the mentioned example, the size of a minimum vertex cover is upperbounded by $2/3n$ for connected graphs, where n here and in the following is the number of vertices of the graph instance. Chen, Kanj and Xia [88] used this result to turn their $\mathcal{O}(1.194^k + kn)$ algorithm for k-VERTEX COVER ON GRAPHS OF MAXIMUM DEGREE THREE into an $\mathcal{O}(1.194^{2n/3}) = \mathcal{O}(1.1254^n)$ algorithm for INDEPENDENT SET PROBLEM ON GRAPHS OF MAXIMUM DEGREE THREE. So, knowing bounds on the possible parameter values helps considerably reduce the bounds on the computing time. In a similar spirit, the four-color theorem teaches us that each n-vertex planar graph has a minimum vertex cover of size at most $3/4n$. The known $\mathcal{O}(1.29^k + kn)$ algorithm for k-VERTEX COVER this way implies an $\mathcal{O}(1.285^{3n/4}) = \mathcal{O}(1.207^n)$ algorithm for PLANAR INDEPENDENT SET, which is slightly better than Robson's published algorithm [342] (for general graphs) needing $\mathcal{O}(1.211^n)$ time. However, we already mentioned that Robson later improved that algorithm considerably, and that algorithm not only applies to graphs of maximum degree three.

With problems having both $\mathcal{FPT}$ algorithms for their primal and for their dual parameterizations, we have the possibility of converting both algorithms into one non-parameterized algorithm, kind of attacking the problem from two sides. This means that we can use either of the two $\mathcal{FPT}$ algorithms.

Theorem 10.12 *Let (P, s) be a parameterized problem with size function and P_d its dual. Assume that both P and P_d are in $\mathcal{FPT}$. Let f be some monotone function. Assume that there is an algorithm A for solving P on instance (I, k), having running time $\mathcal{O}(f(\beta k)p(s(I)))$ for some polynomial p, and that A_d is an algorithm for solving P_d on instance (I, k_d) running in time $\mathcal{O}(f(\beta_d k_d)p_d(s(I)))$ for a polynomial p_d.*

Then, there is an algorithm A' for solving the non-parameterized problem instance I running in time

$$\mathcal{O}(f(\frac{\beta\beta_d}{\beta+\beta_d}s(I))p'(s(I)))$$

for some polynomial p'.

Algorithm 85 Algorithm A' based on the parameterized algorithms A and A_d

```
for all parameter values $k$ do
  if $k \leq \frac{\beta_d}{\beta+\beta_d} s(I)$ then
    compute $A(I,k)$
  else
    compute $A_d(I, s(I)-k)$
  end if
end for
output the 'best' of all computed solutions
```

Proof. Algorithm A' will use A as long as it is better than using A_d. This means we have to compare

$$f(\beta k) \quad \text{versus} \quad f(\beta_d(s(I)-k_d))$$

Since f is monotone, this means we simply have to compare

$$\beta k \quad \text{versus} \quad \beta_d(s(I)-k_d)$$

Some algebra shows that the algorithm A' (see Alg. 85) is then "best" for the de-parameterized problem P, given an instance I.

Considering the boundary case $k = \frac{\beta_d}{\beta+\beta_d} s(I)$ gives the claimed worst case running time. Here, $p'(j) = j(p(j) + p_d(j))$. ∎

Let us explain this theorem by some *example computations.*

1. By taking the $\mathcal{O}(1.194^k+n)$ algorithm for k-VERTEX COVER ON GRAPHS OF MAXIMUM DEGREE THREE and the (trivial) $\mathcal{O}(4^{k_d}n)$ for the dual k_d-INDEPENDENT SET PROBLEM ON GRAPHS OF MAXIMUM DEGREE THREE, we obtain an $\mathcal{O}(1.171^n)$ algorithm for MAXIMUM INDEPENDENT SET ON GRAPHS OF MAXIMUM DEGREE THREE. This algorithm is worse than the one obtained by Chen, Kanj and Xia (see above). Why? The case distinction within the combined algorithm is at $k \leq 0.8866n$, while we *know* that always $k \leq 0.666n$. Hence, the parameterized independent set algorithm will be never employed.

2. We can play the same game for MAXIMUM INDEPENDENT SET ON PLANAR GRAPHS.

   Combining the $\mathcal{O}(6^{k_d} + p(n))$-algorithm for k_d-INDEPENDENT SET ON PLANAR GRAPHS and the known $\mathcal{O}(1.285^k + kn)$ algorithm for VERTEX COVER (on general graphs) [86], we get an $\mathcal{O}(1.246^n)$ algorithm, clearly worse than Robson's. This is still true if we use PLANAR INDEPENDENT SET branching algorithms based on Theorem 5.12.

Alternatively, we can start with the parameterized algorithms of "type" $\mathcal{O}(c^{\sqrt{k}} + n)$ which are known for both problems. This means (in the setting of the theorem) that we let $f(x)$ to be $2^{\sqrt{x}}$.

Plugging in the best-known constants, i.e.,

- $\beta = 4.5^2 = 20.25$ in the case of k-VERTEX COVER ON PLANAR GRAPHS [193] and
- $\beta_d = 48$ in the case of k_d-INDEPENDENT SET ON PLANAR GRAPHS (long version of [13]),

we get an

$$\mathcal{O}(2^{3.773\sqrt{n}}) = \mathcal{O}(13.68^{\sqrt{n}})$$

algorithm for MAXIMUM INDEPENDENT SET ON PLANAR GRAPHS. Using that a minimum vertex cover in planar graphs has at most $3/4n$ vertices this time gives us a worse result this time, namely an algorithm running in time $\mathcal{O}(2^{\sqrt{20.25*3/4*n}}) = \mathcal{O}(2^{3.898\sqrt{n}}) = \mathcal{O}(14.90^{\sqrt{n}})$.

More precisely, taking (in the spirit of klam values as described in [134]) a value of 10^{20} "operations" as "benchmark" for how far each type of algorithm might take us, we see that with Robson's algorithm graphs with about 250 vertices are still manageable, while our new algorithm can cope with graphs with over more than 300 vertices.

By a completely different approach, namely by bounding the tree-width of any planar graph $G = (V, E)$ by $3.182\sqrt{|V|}$, see Sec. 7.6.3, Fomin and Thilikos were recently able to obtain an even better algorithm, running in time $\mathcal{O}(9.08^{\sqrt{n}})$. This means that actually planar graphs with up to 500 vertices are manageable.

3. We now consider FEEDBACK VERTEX SET. Since on general graphs the parameterized dual is hard [257, Cor. 7], we again consider the problem *restricted to planar graphs.*

   Based on a coloring theorem of Borodin [59] and on the reasoning given by Goemans and Williamson [206], in parts explicitly formulated in terms of parameterized complexity in [262], the following two lemmas can be shown (also confer Cor. 4.19):[3]

[3]For the best results on k-FEEDBACK VERTEX SET ON PLANAR GRAPHS, we take the constants from the mentioned sophisticated analysis of FACE COVER from Chap. 5 and use geometric dualization.

Lemma 10.13 VERTEX INDUCED FOREST *has a* $2.5k_d$ *kernel and can hence be solved in time*

$$\mathcal{O}\left(\left(2.5\left(\frac{2.5}{1.5}\right)^{1.5}\right)^{k_d} + p(n)\right) = \mathcal{O}(5.3792^{k_d} + p(n))$$

by brute force.

Lemma 10.14 k-FEEDBACK VERTEX SET ON PLANAR GRAPHS *can be solved in time* $\mathcal{O}(4.5414^k + n^2)$*.*

Taken together, this gives an algorithm for MAXIMUM VERTEX INDUCED FOREST IN PLANAR GRAPHS running in time $\mathcal{O}(2.4624^n)$. Taking the 10^{20}-benchmark, this means that planar graphs with more than 50 vertices can be treated this way.

Of course, there is still the rather trivial $\mathcal{O}(2^n)$ algorithm for this problem that simply tests all vertex assignments that cannot be beaten this way. It is interesting to see that very recent results of Fomin and Pyatkin [188] show that MINIMUM FEEDBACK VERTEX SET can be solved in time $\mathcal{O}^*(1.8621^n)$. This not only improves on the non-parameterized result presented above, but also on Lemma 10.13, which can be replaced by the following one:

Lemma 10.15 VERTEX INDUCED FOREST *has a* $2.5k_d$ *kernel and can hence be solved in time* $\mathcal{O}^*(1.8621^{2.5k_d}) = \mathcal{O}^*(4.7316^{k_d})$ *by using the algorithm described in [188].*

Similar results can be also obtained for other problems and other graph families, as they are described in [114]. As a non-graph-theoretic example, let us mention the *tardy task problem* from [170]. The parameter k is the number of tasks t which don't meet their deadline $d(t)$ on a one-processor schedule, this way describing the problem k-LATE TASKS. The size of an instance would be the number n of tasks to be scheduled. This defines then the dual problem k_d TASKS ON TIME. Then, as a kind of "second parameter," the width m of the order given by the precedence constraints of the tasks was introduced. For k-LATE TASKS, an $\mathcal{O}(m^{k+1}n+n^{2.5})$-algorithm has been given. For k_d TASKS ON TIME, an $\mathcal{O}(m^{mk_d}n + n^{2.5})$-algorithm was developed. For $m = 2$, this gives an overall algorithm running in time $\mathcal{O}(1.588^n)$. For $m = 3$, the classical running time is $\mathcal{O}(2.280^n)$ and for $m = 4$, $\mathcal{O}(3.032^n)$. Keeping the second "parameter" m described in that paper, our method provides an $\mathcal{O}(m^{mn/(m+1)})$ algorithm for this problem, where n is the number of tasks.

Chapter 11

The WWW repository

In the following, we present a list of sites on the WWW where papers, talks etc. on parameterized complexity and algorithmics can be found. Of course, this list is neither meant to be exhaustive nor can it be guaranteed that all items still "work" when you try to follow them up. This is one of the unfortunate consequences of the dynamic nature of the World Wide Web. Nonetheless, the list may give useful hints where to find more about this area.

1. The web resources on algorithmics are very impressive nowadays. For the contents of this books, in particular `http://www.nist.gov/dads/` is of interest, a rather recently built "Dictionary of Algorithms and Data Structures."

2. A good account on *Bell's number* can be found in `http://mathworld.wolfram.com/BellNumber.html`. In fact, Wolfram's page contains a collection of many useful mathematical facts.

3. The *Compendium of Parameterized Problems* compiled and updated by M. Cesati can be found at `http://bravo.ce.uniroma2.it/home/cesati/research/compendium.pdf`.

4. A general coverage of *chess puzzles*, in particular of the queens problems discussed in the introductory chapter, are contained in `http://mathworld.wolfram.com/Chess.html` and in `http://www.dsitri.de/wiki.php?page=NQP`. A good animation for the $n \times n$ Queens Problem can be found at `http://www.math.utah.edu/~alfeld/queens/queens.html`. Further hints on chess-related puzzles can be found in Velucchi's page `http://anduin.eldar.org/~problemi/papers.html`.

5. Cliquewidth of special classes of graphs are listed in `http://www.laria.u-picardie.fr/~vanherpe/cwd/cwd.html`; unfortunately, the links given in that page are usually out of date. The most authorative list of publications on the topic is surely the one by Courcelle himself: `http://www.labri.fr/Perso/~courcell/ActSci.html`. Regarding "older" publications `http://www.labri.fr/Perso/~courcell/Textes/BiblioReecritureGraphes%281995%29.pdf` contains a rather comprehensive survey with the title "Réécriture de graphes: orientation bibliographique."

6. `http://www.ics.uci.edu/~eppstein/junkyard/euler/` contains an interesting collection of 17 different proofs for *Euler's formula* that says that, for any *connected* plane graph $G = (V, E)$,

   $$|V| - |E| + |F| = 2,$$

   where F is the set of faces of G. Nice 3D-visualizations for various embeddings onto the "sphere" (this is how these drawings can be interpreted) can be found in `http://www.math.ohio-state.edu/~fiedorow/math655/Euler.html`. A simple deduction of the fact that each planar graph has (at least one) vertex of degree at most five can be found in `http://www-math.mit.edu/18.310/planarity_coloring.html`.

   The other entries in Eppstein's "junkard" are also recommendable, so give it a try!

7. `http://www.math.gatech.edu/~thomas/FC/fourcolor.html` gives a brief summary of a new proof of the *Four Color Theorem* and a four-coloring algorithm found by Neil Robertson, Daniel P. Sanders, Paul Seymour and Robin Thomas.

8. `http://wwwteo.informatik.uni-rostock.de/isgci` contains a nice collection of *graph classes* and their interrelations.

9. Many nice examples of *Mathematical Games* that can be formulated as integer linear programs or as graph problems can be found at `http://ite.informs.org`. This site (and the corresponding electronic journal) is devoted to the presentation of Operations Research techniques in education. Nice and recommendable is also the site `http://www.chlond.demon.co.uk/academic/puzzles.html` called "Integer Programming in Recreational Mathematics."

10. http://tracer.lsi.upc.es/minla/minla_info.php seems to be a place where up-to-date information on LINEAR ARRANGEMENT is kept.

11. http://www.algorithmic-solutions.com/enledabeschreibung.htm contains a description of the (formerly public domain) software package LEDA. To quote from that websited: LEDA provides algorithmic in-depth knowledge in the field of graph- and network problems, geometric computations, combinatorial opimization and other.

12. The slides of the talk of D. Marx at CCC / Freiburg can be found at http://www.cs.bme.hu/~dmarx/papers/marx-freiburg-slides.pdf. The picture of the mentioned Swiss army knife is on page 6.

13. Searching through the internet for notions like *matching* will give you many hits, including good slides etc. One example is http://www.cs.rpi.edu/~goldberg/05-AAH/matching.pdf which also shows a nice illustration of a proof of Theorem 4.30. Nice applications of Hall's Theorem can be found in the coursenotes http://www.stat.uchicago.edu/~lalley/Courses/388/Matching.pdf.

14. http://mathworld.wolfram.com is a very nice, easily accessible internet-based mathematical dictionary.

15. There are research groups dedicated to *model-based reasoning* throughout the world. A European example is http://www.cs.ru.nl/~peterl/mbr.html, where further pointers can be found.

16. The site http://parc.lboro.ac.uk/research/projects/parseqgd/ not only contains a good project description of a recent project dedicated to graph drawing, in particular, to crossing minimization problems, but also some nice overview papers, e.g., [106].

17. The background on ROMAN DOMINATION is nicely described in the online John Hopkins Magazine, more specifically, look into http://www.jhu.edu/~jhumag/0497web/locate3.html. Interestingly, a small exercise in connection with that problem is posed, whose solution is presented in http://www.jhu.edu/~jhumag/0697web/revelle.html.

18. http://dept-info.labri.u-bordeaux.fr/~robson/mis/techrep.html contains a HTML version of the currently best algorithm for MAXIMUM INDEPENDENT SET, see [343]. Also, the mathematical background explaining the analysis of the memorization part of the algorithm can be found in that directory.

19. We simply quote from the Homepage of Thinkfun: `http://www.puzzles.com/products/rushhour.htm`: “Rush Hour is a genuine puzzle phenomenon with legions of fans. There are some great places on the Web dedicated to this most successful sliding block puzzle of these years. We’d like to introduce here some of them. Good Luck!” As can be seen, there are numerous links on Rush Hour to be found at that site.

Chapter 12

Problems

In this chapter, we collect all definitions of problems encountered in this book. In a certain way, this collection also complements the corresponding appendix in [134]. However, since we are mostly dealing with graph problems in this book, we will rather have a finer grained classification of those graph problem lists; problems from other areas will we rather scarce.

12.1 Cover problems and their relatives

Problem name: VERTEX COVER (VC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $|C| \leq k$?

From the viewpoint of classical complexity, the following two problems are eqivalent to VERTEX COVER.

Problem name: CLIQUE (CQ)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *clique* $C \subseteq V$ with $|C| \geq k$?

Problem name: INDEPENDENT SET (IS)

Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Also variants of this problem can be considered. In this text, we looked at:

Problem name: GENERALIZED VERTEX COVER (VCGEN)
Given: A graph $G = (V, E)$, a subset V' of vertices
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V'$ with $|C| \leq k$?

Problem name: CLIQUE (CQE)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *edge-induced clique* $C \subseteq E$ with $|C| \geq k$?

Problem name: PROFIT VERTEX COVER (PRVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer p
Output: Is there a *profit vertex cover* $C \subseteq V$ with $|E| - |E(G[V \setminus C])| - |C| \geq p$?

Problem name: t-VERTEX COVER (TVC)
Given: A graph $G = (V, E)$
Parameter: positive integers k and t
Output: Is there a *t-vertex cover* $C \subseteq V$ with $|C| \leq k$, i.e., $|\{e \in E \mid C \cap e \neq \emptyset\}| = |E| - |E(G[V \setminus C])| \geq t$?

Problem name: MINIATURIZED VERTEX COVER (VCMINI)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $|C| \leq k \log(|V|)$?

Problem name: CLIQUE COMPLEMENT COVER (CCC)

Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *clique complement cover* $C \subseteq E$ with $|C| \leq k$?

Problem name: VERTEX CLIQUE COMPLEMENT COVER (VCCC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *vertex clique complement cover* $C \subseteq V$ with $|C| \leq k$?

Problem name: PLANAR INDEPENDENT SET (PIS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Problem name: INDEPENDENT SET ON DISKS GRAPHS (DIS)
Given: A disk graph $G = (V, E)$ whose disk model has radii between 1 and α and is λ-precision
Parameter: a positive integer k
Output: Is there an *independent set* $I \subseteq V$ with $|I| \geq k$?

Problem name: WEIGHTED VERTEX COVER (WVC)
Given: A graph $G = (V, E)$ with vertex weights $\omega : V \to \mathbb{R}_{\geq 1}$
Parameter: a positive integer k
Output: Is there a *vertex cover* $C \subseteq V$ with $\omega(C) \leq k$?

Problem name: CONSTRAINT BIPARTITE VERTEX COVER (CBVC)
Given: A bipartite graph $G = (V_1, V_2, E)$
Parameter: positive integers k_1, k_2
Output: Is there a *vertex cover* $C \subseteq V_1 \cup V_2$ with $|C \cap V_i| \leq k_i$ for $i = 1, 2$?

Problem name: VERTEX COVER, PARAMETERIZED BY NUMBER OF VERTICES OF DEGREE THREE OR LARGER (VCDEG)
Given: A graph $G = (V, E)$
Parameter: a positive integer ℓ that equals the number of vertices in G of degree three or larger

Output: What is the size of a minimum *vertex cover* $C \subseteq V$?

Problem name: VERTEX COVER, PARAMETERIZED BY TREEWIDTH (VCTW)
Given: A graph $G = (V, E)$ together with some tree decomposition
Parameter: a positive integer k that equals the width of the tree decomposition
Output: What is the size of a minimum *vertex cover* $C \subseteq V$?

Problem name: COUNTING INDEPENDENT SET OF QUEENS (CQIS)
Given: An $n \times n$ chessboard C
Parameter: a positive integer n
Output: In how many ways can n queens be positioned on C such that no two of them are mutually attacking?

12.2 Dominating problems and their relatives

Another important graph-theoretic problem is:

Problem name: DOMINATING SET (DS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *dominating set* $D \subseteq V$ with $|D| \leq k$?

Again, variants can be considered:

Problem name: PLANAR DOMINATING SET (PDS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *dominating set* $D \subseteq V$ with $|D| \leq k$?

Problem name: DOMINATING SET ON BOUNDED GENUS GRAPHS
Given: A graph $G = (V, E)$ of genus bounded by γ

Parameter: a positive integer k
Output: Is there a *dominating set* $D \subseteq V$ with $|D| \leq k$?

Problem name: RED-BLUE DOMINATING SET (RBDS)
Given: A graph $G = (V, E)$ with V partitioned as $V_{\text{red}} \cup V_{\text{blue}}$
Parameter: a positive integer k
Output: Is there a *red-blue dominating set* $D \subseteq V_{\text{red}}$ with $|D| \leq k$, i.e., $V_{\text{blue}} \subseteq N(D)$?

Problem name: ROMAN DOMINATION (ROMAN)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *Roman domination* function R such that $\sum_{x \in V} R(x) \leq k$?

Problem name: PLANAR ROMAN DOMINATION (pROMAN)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *Roman domination* function R such that $\sum_{x \in V} R(x) \leq k$?

Problem name: DOMINATING REARRANGEMENT (DR)
Given: A graph $G = (V, E)$, a subset $S \subseteq V$
Parameter: a positive integer $k = |S|$
Output: Is there a *dominating rearrangement* $r : S \to N[S], s \mapsto r(s) \in N[s]$ such that $r(S) \subseteq V$ is a dominating set?

Problem name: DOMINATING SET, GIVEN VERTEX COVER NUMBER (DSVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k such that G has a vertex cover of size k
Output: Is there a minimum *dominating set* $D \subseteq V$?

Problem name: CONSTRAINT BIPARTITE DOMINATING SET (CBDS)
Given: A bipartite graph $G = (V_1, V_2, E)$
Parameter: positive integers k_1, k_2

Output: Is there a *dominating set* $D \subseteq V_1 \cup V_2$ with $|D \cap V_i| \leq k_i$ for $i = 1, 2$?

Problem name: INDEPENDENT DOMINATING SET (IDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an independent *dominating set* $D \subseteq V$ with $|D| \leq k$?

Problem name: PLANAR INDEPENDENT DOMINATING SET (PIDS)
Given: A planar graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *independent dominating set* $D \subseteq V$ with $|D| \leq k$?

Problem name: CONNECTED DOMINATING SET (CDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *connected dominating set* $D \subseteq V$ with $|D| \leq k$, i.e., D is both a connected set and a dominating set?

Problem name: NONBLOCKER SET (NB)
Given: A graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq k_d$?

Problem name: NONBLOCKER SET WITH CATALYTIC VERTEX (NBCAT)
Given: A graph $G = (V, E)$, a catalytic vertex c
Parameter: a positive integer k_d
Output: Is there a *nonblocker set* $N \subseteq V$ with $|N| \geq k_d$ such that $c \notin N$?

Problem name: DOMINATING SET OF QUEENS (QDS)
Given: An $n \times n$ chessboard C
Parameter: a positive integer k
Output: Is it possible to place k queens on C such that all squares are dominated ?

Problem name: DOMINATING SET OF QUEENS (QDS), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: An $n \times n$ chessboard C
Parameter: a positive integer k
Output: Is it possible to place $n/2 + k$ queens on C such that all squares are dominated ?

Likewise, edges or faces may be in the focus of domination:

Problem name: EDGE DOMINATING SET (EDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $|D| \leq k$?

Problem name: GENERALIZED EDGE DOMINATING SET (GEDS)
Given: A graph $G = (V, E)$, a set $R \subseteq V$ of red vertices
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $|D| \leq k$ such that all vertices from R are covered by D?

Problem name: WEIGHTED EDGE DOMINATING SET (WEDS)
Given: A graph $G = (V, E)$ with edge weights $\omega : E \to \mathbb{R}_{\geq 1}$
Parameter: a positive integer k
Output: Is there an *edge dominating set* $D \subseteq E$ with $\omega(D) \leq k$?

12.3 Graph modification problems

Graph modification problems deliver a wealth of hard problems that tend to classify in $\mathcal{FPT}$. The examples contained here are:

Problem name: TRIANGLE EDGE DELETION (TED)
Given: A graph $G = (V, E)$

Parameter: a positive integer k
Output: Is there an edge set $C \subseteq E$ with $|C| \leq k$ whose removal produces a graph without triangles as vertex-induced subgraphs?

Problem name: TRIANGLE VERTEX DELETION (TVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a graph without triangles as vertex-induced subgraphs?

Problem name: CLUSTER VERTEX DELETION (CVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a graph being a union of vertex-induced cliques?

Problem name: COGRAPH VERTEX DELETION (CoVD)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an vertex set $C \subseteq V$ with $|C| \leq k$ whose removal produces a cograph ?

Problem name: BIPARTIZATION (BP)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *bipartization set* $C \subseteq V$ with $|C| \leq k$ whose removal produces a bipartite graph?

Problem name: BIPARTIZATION IMPROVEMENT
Given: A graph $G = (V, E)$, a bipartization set $C \subseteq V$ with $|C| = k + 1$
Parameter: a positive integer k
Output: Is there a bipartization set $C' \subseteq V$ with $|C'| \leq k$?

Problem name: BIPARTIZATION, EDGE VARIANT (BPEDGE)
Given: A graph $G = (V, E)$

Parameter: a positive integer k
Output: Is there a *bipartization set* $C \subseteq E$ with $|C| \leq k$ whose removal produces a bipartite graph?

Also, cover-like problems can be viewed as graph modification problems. Examples discussed in the text are:

Problem name: FEEDBACK VERTEX SET (FVS)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *feedback vertex set* of size at most k, i.e.,

$$\exists F \subseteq V, |F| \leq k, \forall c \in C(G)(F \cap c \neq \emptyset)?$$

Here, $C(G)$ denotes the set of cycles of G, where a *cycle* is a sequence of vertices (also interpreted as a set of vertices) $v_0, v_1, \dots, v_\ell$ such that $\{v_i, v_{(i+1) \bmod \ell}\} \in E$ for $i = 0, \dots, \ell - 1$.

Problem name: FEEDBACK EDGE SET (FES)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *feedback edge set* of size at most k, i.e.,

$$\exists F \subseteq E, |F| \leq k, \forall c \in C(G)(F \cap c \neq \emptyset)?$$

Here, $C(G)$ denotes the set of cycles of G, where a *cycle* is a sequence of vertices (also interpreted as a set of edges) $v_0, v_1, \dots, v_\ell$ such that $\{v_i, v_{(i+1) \bmod \ell}\} \in E$ for $i = 0, \dots, \ell - 1$.

---

as well as their duals:

Problem name: VERTEX INDUCED FOREST (ViF)
Given: a (simple) graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *vertex-induced forest* of size at least k_d, i.e.,

$$\exists F \subseteq V, |F| \geq k_d, C(G[F]) = \emptyset?$$

Problem name: EDGE INDUCED FOREST (EIF)
Given: a (simple) graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *edge-induced forest* of size at least k_d, i.e.,

$$\exists F \subseteq E, |F| \geq k_d, C(G[E]) = \emptyset?$$

On planar graphs, corresponding problems are:

Problem name: FACE COVER (FC)
Given: A plane graph $G = (V, E)$ with face set F
Parameter: a positive integer k
Output: Is there a *face cover set* $C \subseteq F$ with $|C| \leq k$?

Problem name: VERTEX INDUCED FOREST IN PLANAR GRAPHS (PVIF)
Given: a (simple) planar graph $G = (V, E)$
Parameter: a positive integer k_d
Output: Is there a *vertex-induced forest* of size at least k_d, i.e.,

$$\exists F \subseteq V, |F| \geq k_d, C(G[F]) = \emptyset?$$

12.4 Further graph-theoretic problems

Further graph-theoretic problems discussed in the text include:

Problem name: MAXIMUM LEAF SPANNING TREE (MAXLST)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k_d

Output: Is there a *spanning tree* of G with at least k_d leaves?

Problem name: MINIMUM INNER NODE SPANNING TREE (MININST)
Given: A (simple) graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *spanning tree* of G with at most k inner nodes?

Problem name: MINIMUM MAXIMAL INDEPENDENT SET (MMIS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a maximal independent set of cardinality $\leq k$?

Problem name: MAXIMUM MINIMAL VERTEX COVER (MMVC)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a minimal vertex cover set of cardinality $\geq k$?

Problem name: MAXIMUM MINIMAL DOMINATING SET (MMDS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Does there exist a minimal dominating set of cardinality $\geq k$?

Problem name: MAXIMUM CUT (MAXCUT)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a *cut set* $C \subseteq E$ with $|C| \geq k$, i.e., $(V, V \setminus C)$ is a bipartite graph?

Problem name: SEPARATING VERTEX SETS ENUMERATION (SVS)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Enumerate all minimum size separating vertex sets of size at most k.

12.5 Graph drawing problems

The following problems can be considered as graph drawing problems.

Problem name: CROSSING NUMBER (CRN)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is $CR(G) \leq k$?

Problem name: BANDWIDTH (BW)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that $\forall \{u, v\} \in E : |\sigma(u) - \sigma(v)| \leq k$?

Problem name: CUTWIDTH (CW)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that $\forall 1 \leq i < |V| : |\{\{u, v\} \in E \mid \sigma(u) \leq i < \sigma(v)\}| \leq k$?

Problem name: LINEAR ARRANGEMENT (LA)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| \leq k \ ?$$

Problem name: LINEAR ARRANGEMENT (LA), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A graph $G = (V, E)$
Parameter: a positive integer k

Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that

$$\sum_{\{u,v\} \in E} |\sigma(u) - \sigma(v)| \leq k + |E|?$$

Problem name: DIRECTED LINEAR ARRANGEMENT (DLA)
Given: A directed graph $G = (V, A)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ that respects the orientation of G, i.e., $\sigma(u) < \sigma(v)$ whenever $(u, v) \in A$, such that

$$\sum_{(u,v) \in A} |\sigma(u) - \sigma(v)| \leq k \ ?$$

Problem name: DIRECTED LINEAR ARRANGEMENT (DLA), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A directed graph $G = (V, A)$
Parameter: a positive integer k
Output: Is there a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ that respects the orientation of G such that

$$\sum_{(u,v) \in A} |\sigma(u) - \sigma(v)| \leq k + |A| \ ?$$

Problem name: LINEAR ARRANGEMENT BY DELETING EDGES (LADE)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there an edge set E' with $|E'| \leq k$ and a one-to-one mapping $\sigma : V \to \{1, \dots, |V|\}$ such that

$$\sum_{\{u,v\} \in E \setminus E'} |\sigma(u) - \sigma(v)| = |E \setminus E'| \ ?$$

Problem name: ONE-SIDED CROSSING MINIMIZATION (OSCM)
Given: A bipartite graph $G = (V_1, V_2, E)$ and a linear order $\prec_1$ on V_1.
Parameter: a positive integer k
Output: Is there a linear order $\prec$ on V_2 such that, when the vertices from V_1 are placed on a line (also called layer) L_1 in the order induced by $\prec_1$ and the vertices from V_2 are placed on a second layer L_2 (parallel to L_1) in the order induced by $\prec$, then drawing straight lines for each edge in E will introduce no more than k edge crossings?

Problem name: TWO-LAYER PLANARIZATION (TLP)
Given: A graph $G = (V, E)$
Parameter: a positive integer k
Output: Is there a set $C \subseteq E$, $|C| \leq k$, whose removal makes the graph biplanar?

Problem name: ONE-LAYER PLANARIZATION (OLP)
Given: A bipartite graph $G = (V_1, V_2, E)$, a linear ordering $<$ on V_1
Parameter: a positive integer k
Output: Is there a set $C \subseteq E$, $|C| \leq k$, whose removal allows a biplanar drawing of the graph that respects $<$ on V_1?

Problem name: TWO-TREE CROSSING MINIMIZATION (TTCM)
Given: A two-tree (T_1, T_2) with leaf labels Λ
Parameter: a positive integer k
Output: Can (T_1, T_2) be drawn with at most k crossings ?

Problem name: ONE-TREE CROSSING MINIMIZATION (OTCM)
Given: A two-tree (T_1, T_2) with leaf labels Λ, where the ordering of the vertices of T_1 is fixed
Parameter: a positive integer k
Output: Can (T_1, T_2) be drawn with at most k crossings ?

Problem name: TWO-TREE DRAWING BY DELETING EDGES (TTDE)
Given: A two-tree (T_1, T_2) with leaf labels Λ
Parameter: a positive integer k
Output: Is there a label set $L \subseteq \Lambda$ with $|L| \leq k$ such that the two-tree

$(T_1 \langle \Lambda \setminus L \rangle, T_2 \langle \Lambda \setminus L \rangle)$ can be drawn without crossings ?

Problem name: ONE-TREE DRAWING BY DELETING EDGES (OTDE)
Given: A binary tree T_1 with leaf labels Λ, a linear ordering $\prec$ on Λ
Parameter: a positive integer k
Output: Is there a label set $L \subseteq \Lambda$ with $|L| \leq k$ such that the tree $T_1 \langle \Lambda \setminus L \rangle$ can be drawn without crossings in the plane, so that the leaves in $\Lambda \setminus L$ are arranged according to the ordering $\prec$ on some line ?

12.6 Hypergraph problems

Problem name: d-HITTING SET (d-HS)
Given: A hypergraph $G = (V, E)$ with *hyperedge size* bounded by d
Parameter: a positive integer k
Output: Is there a *hitting set* $C \subseteq V$ with $|C| \leq k$?

Problem name: MINIMUM HITTING SET, PARAMETERIZED BY # EDGES (HSE)
Given: A hypergraph $G = (V, E)$
Parameter: $|E|$
Output: Find a minimum *hitting set* $C \subseteq V$

Problem name: SET COVER (SC)
Given: A groundset X, a collection T of subsets of X
Parameter: a positive integer k
Output: Is there a *set cover* $C \subseteq T$ with $|C| \leq k$, i.e., every element in X belongs to at least one member of C?

Problem name: MULTI-HITTING SET (HSMULTI)
Given: A hypergraph $G = (V, E)$
Parameter: positive integers k, ℓ
Output: Is there a *multi-hitting set* $C \subseteq V$ with $|C| \leq k$, i.e., C satisfies

$\forall e \in E \exists c \subseteq e(|c| \geq \ell \wedge c \subseteq C)$?

12.7 Network problems

Problem name: CallControl
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ and a corresponding feasible path set $p(A)$ resulting from assigning to each request $r \in A$ some path $p(r)$ such that $p(A)$ is feasible and the set of *rejected requests* $R \setminus A$ contains no more than k elements?

Problem name: CallControl-Pre
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Problem name: d-CallControl-Pre
Given: A communication network represented by an undirected graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a set $R \subseteq V \times V$ of communication requests, a path assigning function $p : R \to P(G)$
Parameter: a positive integer k, an edge capacity bound d
Output: Is there a subset $A \subset R$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Problem name: CallControl in trees of rings
Given: A communication network represented by an undirected graph $G = (V, E)$ that is a tree of rings with unit edge capacities, a set $R \subseteq V \times V$ of

communication requests
Parameter: a positive integer k
Output: Is there a subset $A \subset R$ and a path assigning function $p : R \to P(G)$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

Problem name: CallControl in trees with capacities one or two (CallControl in trees 1-2)
Given: A communication network represented by an undirected graph $G = (V, E)$ that is a tree with $E_i \subseteq E$ being the edges of capacity $i \in \{1, 2\}$, a set $R \subseteq V \times V$ of communication requests, a positive integer k
Parameter: $\ell = |E_2|$
Output: Is there a subset $A \subset R$ and a path assigning function $p : R \to P(G)$ such that $p(A)$ is feasible and the set of rejected requests $R \setminus A$ contains no more than k elements?

12.8 Automata problems

Problem name: short nondeterministic small Turing machine computation (SNSTMC)
Given: A (single-tape) nondeterministic Turing machine M whose size is bounded by $f(k)$, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

Problem name: short nondeterministic Turing machine computation (SNTMC)
Given: A (single-tape) nondeterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

Problem name: SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION (SMNTMC)
Given: A multi-tape nondeterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

Problem name: BOUNDED NONDETERMINISM TURING MACHINE COMPUTATION (BNTMC)
Given: A (single-tape) nondeterministic Turing machine M, an input string x, an integer n coded in unary
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that reaches a final accepting state in at most n steps and uses at most k nondeterministic steps?

Problem name: COMPACT DETERMINISTIC TURING MACHINE COMPUTATION (CDTMC)
Given: A deterministic Turing machine M, an input string x
Parameter: a positive integer k
Output: Is there an accepting computation of M on input x that visits at most k squares?

12.9 Logical problems

Logical problems might have deserved more attention in this book. Here are a few of them, anyways:
Problem name: MAXIMUM SATISFIABILITY (MAXSAT)
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X
Parameter: a positive integer k
Output: Is there an assignment $\alpha : X \to \{0, 1\}$ such that at least k clauses in F evaluate to 1 (true) under α?

Problem name: MAXIMUM SATISFIABILITY (MAXSAT), PARAMETERIZED ABOVE GUARANTEED VALUE
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X
Parameter: a positive integer k
Output: Is there an assignment $\alpha : X \to \{0, 1\}$ such that at least $m/2 + k$ clauses in F evaluate to 1 (true) under α?

Problem name: SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (CLAUSE PARAMETERIZATION) (3-SAT)
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X, each clause having at most three literals
Parameter: a positive integer k, upperbounding the number of clauses of size three
Output: Is there a satisfying assignment $\alpha : X \to \{0, 1\}$ for F?

Problem name: SATISFIABILITY PROBLEM WITH CLAUSES OF SIZE THREE (VARIABLE PARAMETERIZATION) (3-SAT (VARIABLE))
Given: A Boolean formula F in conjunctive normal form (CNF), with variables X, each clause having at most three literals
Parameter: $|X|$
Output: Is there a satisfying assignment $\alpha : X \to \{0, 1\}$ for F?

12.10 Miscellaneous and applications

Some problems don't fit into the categories listed above. They are collected here.

Problem name: MAXIMUM AGREEMENT SUBTREE (MAST)
Given: A set $\{T_1, \dots, T_n\}$ of binary rooted trees with equal label set $L = L(T_1) = \cdots = L(T_n)$
Parameter: a positive integer k
Output: Is there a set of labels $L' \subseteq L$, $|\Lambda| \leq k$, such that all trees $T_i \setminus \Lambda$ are isomorphic?

Problem name: SPARE ALLOCATION (SAP)
Given: A $n \times m$ binary matrix A representing an erroneous chip with $A[r,c] = 1$ iff the chip is faulty at position $[r,c]$
Parameter: positive integers k_1, k_2
Output: Is there a *reconfiguration strategy* that repairs all faults and uses at most k_1 spare rows and at most k_2 spare columns?

Problem name: MINIMAL DIAGNOSIS (MD)
Given: A finite set of faults F, a set of effects M, a function $e : F \to 2^M$ relating faults and effects, a set of observed effects $M' \subseteq M$, an integer k
Parameter: the (size of the) relating function e
Output: Is there a set $F' \subset F$ with $|F'| \leq k$ such that $M' \subseteq \bigcup_{f \in F'} e(f)$?

Problem name: MATRIX DOMINATION SET (MDS)
Given: A $n \times n$ matrix with entries from $\{0,1\}$, positive integer k
Parameter: k
Output: Is there a set D of one-entries in the matrix, where $|D| \leq k$, such that every other one-entry has at least one row or one column in common with some one-entry from D?

Problem name: MATRIX ROW COLUMN MERGING (MRCM)
Given: a $n \times m$ $\{0,1\}$-matrix M
Parameter: a positive integer k
Output: Is it possible to get the all-zeros-matrix by merging at most k neighboring rows or columns?

Problem name: MODULE PLACEMENT PROBLEM (MPP)
Given: a set of modules M, a set of wires W connecting modules, i.e., each wire $w \in W$ is a subset of M
Parameter: a positive integer k
Output: Is it possible to find a mapping $\sigma : M \to \{1, \ldots, |M|\}$ such that the overall wire length is less than or equal to k?

Problem name: POSITIVE WEIGHTED COMPLETION OF AN ORDERING

(PCO)
Given: An ordered digraph $P = (V, A)$ and a cost function $\mathfrak{c}$ mapping $A(D([U(P)]^c))$ into the positive integers; by setting $\mathfrak{c}$ to zero for arcs in $A(D(U(P)))$, we can interpret the domain of $\mathfrak{c}$ as $V(P) \times V(P)$.
Parameter: a positive integer k.
Output: Is there a selection A' of arcs from $A(D([U(P)]^c))$ such that the transitive closure $(A' \cup A(P))^+$ is a linear order and

$$\sum\{\mathfrak{c}(a) \mid a \in (A' \cup A(P))^+\} \leq k \ ?$$

Problem name: FACILITY LOCATION (FL)
Given: A bipartite graph $B = (F \uplus C, E)$, consisting of a set F of potential *facility location*s, a set C of *customer*s, and an edge relation E, where $\{f, c\} \in E$ indicates that c can be served from the facility (at) f; and a weight functions $w_F : F \to \mathbb{N}$ and $w_E : E \to \mathbb{N}$ (both called w if no confusion may arise)
Parameter: $k \in \mathbb{N}$
Output: Is there a set $F' \subseteq F$ of facility locations and a set $E' \subseteq E$ of ways to serve customers such that (1) $E' \cap F = F'$, (2) $E' \cap C = C$, and (3) $\sum_{f \in F'} w_F(f) + \sum_{e \in E'} w_E(e) \leq k$?

Problem name: MINIMUM PARTITION (PART)
Given: A finite set $X = \{x_1, \dots, x_n\}$, a weight function $w : X \to \mathbb{R}_{\geq 1}$
Parameter: $k \in \mathbb{N}$
Output: Is there a set $Y \subset X$ such that

$$\max\{\sum_{y \in Y} w(y), \sum_{z \notin Y} w(y)\} \leq k \ ?$$

Problem name: MAXIMUM KNAPSACK (KS)
Given: n items $\{x_1, \dots, x_n\}$ with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k. All numbers are natural numbers encoded in binary.
Parameter: k
Output: Is there a subset of items which yield a profit larger than k and

has an overall size of less than b?

Problem name: MAXIMUM KNAPSACK, MINIMUM WEIGHT (KSMW)
Given: n items $\{x_1, \dots, x_n\}$ with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k. All numbers are natural numbers encoded in binary.
Parameter: b
Output: Is there a subset of items which yield a profit larger than k and has an overall size of less than b?

In Chapter 3, we discussed several algorithmic problems related to RUSH HOUR. For the rather specific definitions surrounding RUSH HOUR, we refer to that section.

Problem name: RUSH HOUR, PARAMETERIZED BY CARS (RH (CARS))
Given: A RH tuple (C, S, p^0, d, Z) of an APR instance
Parameter: a positive integer k, upperbounding $|C|$
Output: Is there a sequence of legal moves that solves the given RH instance?

Problem name: RUSH HOUR, PARAMETERIZED BY MOVES (RH (MOVES))
Given: A RH tuple (C, S, p^0, d, Z) of an APR instance
Parameter: a positive integer m
Output: Is there a sequence of at most m legal moves that solves the given RH instance?

Bibliography

[1] J. A. Abraham et al. Fault tolerance techniques for systolic arrays. *IEEE Computer*, pages 65–75, July 1987.

[2] F. Abu-Khzam and M. Langston. A direct algorithm for the parameterized face cover problem. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 213–222. Springer, 2004.

[3] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Sutters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In L. Arge, G. F. Italiano, and R. Sedgewick, editors, *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 62–69. SIAM, 2004.

[4] N. Adam and J. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21:515–556, 1989.

[5] D. Adolphson and T. C. Hu. Optimal linear orderings. *SIAM J[ournal of] Appl[ied] Math[ematics]*, 25:403–423, 1973.

[6] V. A. Aksionov, O. V. Borodin, L. S. Mel'nikov, G. Sabidussi, M. Stiebitz, and B. Toft. Deeply asymmetric planar graphs. Technical Report PP–2000–14, University of Southern Denmark, IMADA, Odense, Denmark, September 2000.

[7] J. Alber. *Exact Algorithms for NP-hard Problems on Networks: Design, Analysis, and Implementation.* Dissertation, Universität Tübingen, 2002.

[8] 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*, 33:461–493, 2002.

[9] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In M. M. Halldórsson, editor, *7th Scandinavian Workshop on Algorithm Theory SWAT 2000*, volume 1851 of *LNCS*, pages 97–110, 2000.

[10] J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics*, 145:219–231, 2005.

[11] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege. Refined search tree techniques for the PLANAR DOMINATING SET problem. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *LNCS*, pages 111–122. Springer, 2001. Long version to appear in Journal of Computer and System Sciences.

[12] J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM*, 51(3), 2004.

[13] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speedup for planar graph problems. In F. Orejas, P. G. Spirakis, and J. v. Leeuwen, editors, *International Colloquium on Automata, Languages and Programming ICALP'01*, volume 2076 of *LNCS*, pages 261–272. Springer, 2001.

[14] J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. *Journal of Computer and System Sciences*, 67:808–832, 2003.

[15] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speedup for planar graph problems. *Journal of Algorithms*, 52:26–56, 2004.

[16] J. Alber and J. Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *Journal of Algorithms*, 52:134–151, 2004.

[17] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics*, 229(1):3–27, 2001.

[18] L. G. Aleksandrov and H. N. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J[ournal of] Discrete Math[ematics]*, 9:129–150, 1996.

[19] N. Alon. Problems and results in extremal combinatorics—I. *Discrete Mathematics*, 273:31–53, 2003.

[20] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proc. 22nd Symp. Theory of Computing*, pages 293–299. Assoc. Comput. Mach., 1990.

[21] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *Journal of the AMS*, 3:801–808, 1990.

[22] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[23] R. S. Anand, T. Erlebach, A. Hall, and S. Stefanakos. Call control with k rejections. *Journal of Computer and System Sciences*, 67:707–722, 2003.

[24] B. Anrig and J. Kohlas. Model-based reliability and diagnostic: A common framework for reliability and diagnostics. *Int. J. of Intell. Systems*, 18(10):1001–1033, 2003.

[25] K. Appel and W. Haken. Every planar map is four colorable. Part I. Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[26] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II. Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.

[27] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[28] B. Aronov, T. Asano, N. Katoh, K. Mehlhorn, and T. Tokuyama. Polyline fitting of planar points under min-sum criteria. In R. Fleischer and G. Trippen, editors, *Algorithms and Computation: 15th International Symposium ISAAC 2004*, volume 3341 of *LNCS*, pages 77–88. Springer, 2004.

[29] V. Arvind and V. Raman. Approximation algorithms for some parameterized counting problems. In P. Bose and P. Morin, editors, *Algorithms and Computation, 13th International Symposium, ISAAC 2002*, volume 2518 of *LNCS*, pages 453–464. Springer, 2002.

[30] G. Ausiello, P. Creczenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation; Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[31] Y. Azar, L. Epstein, Y. Richter, and G. J. Woeginger. All-norm approximation algorithms. *Journal of Algorithms*, 52:120–133, 2004.

[32] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.

[33] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[34] R. Bar-Yehuda. A linear time 2-approximation algorithm for the min clique-complement problem. Technical Report CS0933, Technion, 1998.

[35] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.

[36] R. Bar-Yehuda and D. Rawitz. Approximating element-weighted vertex deletion problems for the complete k-partite property. *Journal of Algorithms*, 42(1):20–40, 2002.

[37] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing; Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[38] M. Baur and U. Brandes. Crossing minimization in circular layouts. In Hromkovic et al. [242], pages 332–343.

[39] C. Bazgan, J. Monnot, V. Th. Paschos, and F. Serriére. Greedy differential approximations for min set cover. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM*, volume 3381 of *LNCS*, pages 62–71. Springer, 2005.

[40] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3446^n)$: a no-MIS algorithm. Technical Report TR95-033, ECCC, Trier, 1995.

[41] S. Benecke. Higher order domination of graphs. Master's thesis, Department of Applied Mathematics of the University of Stellebosch, South Africa, `http://dip.sun.ac.za/~vuuren/Theses/Benecke.pdf`, 2004.

[42] P. Berman, B. DasGupta, and S. Muthukrishnan. Approximation algorithms for MAX-MIN tiling. *Journal of Algorithms*, 47:122–134, 2003.

[43] P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Proceedings*, volume 3122 of *LNCS*, pages 39–50. Springer, 2004.

[44] F. Bernhart and P. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27:320–331, 1979.

[45] V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. In *15th Ann. Combinatorial Pattern Matching Symposium (CPM)*, volume 3109 of *LNCS*, pages 205–219. Springer, 2004.

[46] D. K. Bhavsar and J. H. Edmondson. Alpha 21164 testability strategy. *IEEE Design and Test*, 14(1):25–33, 1997.

[47] D. Bienstock and C.L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5:93–109, 1990.

[48] D. Bienstock and N. Ozbay. Tree-width and the Sherali-Adams operator. *Discrete Optimization*, 1:13–21, 2004.

[49] P. Bille. Tree edit distance, alignment distance and inclusion. Technical Report TR-2003-23, The IT University of Copenhagen, 2003.

[50] M. Blank. An estimate of the external stability number of a graph without suspended vertices (in Russian). *Prikl. Math. i Programmirovanie Vyp.*, 10:3–11, 1973.

[51] J. P. Blanks. Near-optimal quadratic-based placement for a class of IC layout problems. *IEEE Circuits and Devices*, ??:31–37, September 1985.

[52] J.-P. Bode and H. Harborth. Independence for knights on hexagon and triangle boards. *Discrete Mathematics*, 272:27–35, 2003.

[53] H. Bodlaender. Discovering treewidth. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM*, volume 3381 of *LNCS*, pages 1–16. Springer, 2005.

[54] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[55] R. Boliac and V. V. Lozin. Independent domination in finitely defined classes of graphs. *Theoretical Computer Science*, 301:271–284, 2003.

[56] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.

[57] P. S. Bonsma, T. Brueggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In B. Rovan and P. Vojtáś, editors, *Mathematical Foundations of Computer Science 2003*, volume 2747 of *LNCS*, pages 259–268. Springer, 2003.

[58] R. B. Borie, R. G. Parker, and C. A. Tovey. Deterministic decomposition of recursive graph classes. *SIAM J[ournal of] Discrete Math[ematics]*, 4:481–501, 1991.

[59] O. Borodin. On acyclic colorings of planar graphs. *Discrete Mathematics*, 25:211–236, 1979.

[60] L. Branković and H. Fernau. Approximability of a $\{0, 1\}$-matrix problem. Presented at ACCMCC in Auckland, NZ, 2004.

[61] L. Branković, P. Horak, and M. Miller. An optimization problem in statistical databases. *SIAM Journal on Discrete Mathematics*, 13:346–353, 2000.

[62] A. P. Burger and C. M. Mynhardt. An upper bound for the minimum number of queens covering the $n \times n$ chessboard. *Discrete Applied Mathematics*, 121:51–60, 2002.

[63] A. P. Burger and C. M. Mynhardt. An improved upper bound for queens domination numbers. *Discrete Mathematics*, 266:119–131, 2003.

[64] J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal Comput.*, 22(3):560–572, 1993.

[65] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. In *Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems PODS'95*, pages 151–162, 1995.

[66] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176:89–120, 2002.

[67] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.

[68] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54:465–474, 1997.

[69] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67:789–807, 2003.

[70] R. Carr, T. Fujito, G. Konjevod, and O. Parekh. A 2 1/10 approximation algorithm for a generalization of the weighted edge-dominating set problem. *Journal of Combinatorial Optimization*, 5:317–326, 2001.

[71] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[72] N. de Castro, F. J. Cobos, J. C. Dana, A. Márquez, and M. Noy. Triangle-free planar graphs as segment intersection graphs. *Journal of Graph Algorithms and Applications*, 6:7–26, 2002.

[73] S. Ceria, P. Nobili, and A. Sassano. Set covering problems. In *Annotated Bibliographies in Combinatorial Optimization*, pages 415–428. Wiley, 1997.

[74] M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67:654–685, 2003.

[75] M. Cesati and M. Di Ianni. Computation models for parameterized complexity. *Mathematical Logic Quarterly*, 43:179–202, 1997.

[76] M. Cesati and H. T. Wareham. Parameterized complexity analysis of robot motion planning. In *Proc. 25th IEEE Int. Conf. on Systems, Man and Cybernetics*, 1995.

[77] L. Sunil Chandran and F. Grandoni. Refined memorization for vertex cover. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 61–70. Springer, 2004.

[78] M. Chean and J. A. B. Fortes. A taxonomy of reconfiguration techniques for fault-tolerant processor arrays. *IEEE Computer*, pages 55–69, January 1990.

[79] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. A parallel FPT application for clusters. In *CCGRID*, pages 70–77. IEEE Computer Society, 2003.

[80] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse-grained parallel machines. *Journal of Computer and System Sciences*, 67:691–706, 2003.

[81] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. In *Proc. 19th Annual IEEE Conference on Computational Complexity CCC*, pages 150–160, 2004.

[82] J. Chen, H. Fernau, I. A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. In V. Diekert and B. Durand, editors, *Symposium on Theoretical Aspects of Computer Science STACS 2005*, volume 3404 of *LNCS*, pages 269–280. Springer, 2005.

[83] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing STOC*, pages 212–221. ACM Press, 2004.

[84] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Polynomial time approximation schemes and parameterized complexity. In *Mathematical Foundations of Computer Science MFCS 2004*, volume 3153 of *LNCS*, pages 500–512. Springer, 2004.

[85] J. Chen and I. A. Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithmics. *Journal of Computer and System Sciences*, 67:833–847, 2003.

[86] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[87] J. Chen, I. A. Kanj, L. Perkovic, E. Sedgwick, and G. Xia. Genus characterizes the complexity of graph problems: Some tight results. In *ICALP 2003*, volume 2719 of *LNCS*, pages 845–856, 2003.

[88] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. In T. Ibaraki, N. Katoh, and H. Ono, editors, *Proc. 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *LNCS*, pages 148–157, 2003.

[89] J. Chen, L. Liu, and W. Jia. Improvement for VERTEX COVER on low-degree graphs. *Networks*, 35:253–259, 2000.

[90] Y. Chen and J. Flum. Machine characterization of the classes of the W-hierarchy. In M. Baaz and J. A. Makowsky, editors, *Computer Science Logic, 17th International Workshop, CSL 2003*, volume 2803 of *LNCS*, pages 114–127. Springer, 2003.

[91] Y. Chen and J. Flum. On miniaturized problems in parameterized complexity theory. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 108–120. Springer, 2004.

[92] Z.-Z. Chen. Approximation algorithms for independent sets in map graphs. *Journal of Algorithms*, 41:20–40, 2001.

[93] Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou. Map graphs. *Journal of the ACM*, 49(2):127–138, 2002.

[94] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.

[95] M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. Technical Report TR04-101, Electronic Colloquium on Computational Complexity ECCC Trier, 2004.

[96] M. Chlebík and J. Chlebíková. Improvement of Nemhauser-Trotter theorem and its applications in parametrized complexity. In T. Hagerup and J. Katajainen, editors, *Algorithm Theory—SWAT 2004: 9th Scandinavian Workshop on Algorithm Theory*, volume 3111 of *LNCS*, pages 174–186. Springer, 2004.

[97] M. J. Chlond. IP modeling of chessboard placements and related puzzles. *INFORMS Transactions on Education*, 2(2):56–57, 2002.

[98] H.-A. Choi, K. Nakajima, and C. S. Rim. Graph bipartization and via minimization. *SIAM J[ournal of] Discrete Math[ematics]*, 2(1):38–47, 1989.

[99] F. R. K. Chung. On optimal linear arrangements of trees. *Comp Maths Appl (Comp. & Maths with Appls.)*, 10:43–60, 1984.

[100] E. J. Cockayne. Chessboard domination problems. *Discrete Mathematics*, 86:13–20, 1990.

[101] E. J. Cockayne, S. E. Goodman, and S. T. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4:41–44, 1975.

[102] B. Courcelle. The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299:1–36, 2003.

[103] B. Courcelle. The monadic second-order logic of graphs XV: On a conjecture by D. Seese. To appear in Journal of Applied Logic, 2005.

[104] B. Courcelle, J. Makowsky, and U. Rotics. Linear time solvable optimization problems on certain structured graph families. *Theory of Computing Systems*, 2000.

[105] B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discrete Applied Mathematics*, 101:77–104, 2000.

[106] E. Czabarka, O. Sykora, L.Székély, and I. Vr´to. Biplanar crossing numbers i: A survey of results and problems. year unknown.

[107] E. Czabarka, O. Sýkora, L. A. Szḱely, and I Vrt'o. Outerplanar crossing numbers, the circular arrangement problem and isoperimetric functions. *The Electronic Journal of Combinatorics*, 11:R81, November 2004.

[108] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 1–12. Springer, 2004.

[109] A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 1998.

[110] F. Dehne, M. Fellows, F. Rosamond, and P. Shaw. Greedy localization, iterative compression and modeled crown reductions: New FPT techniques, an improved algorithm for set splitting and a novel $2k$ kernelization for vertex cover. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 271–280. Springer, 2004.

[111] E. D. Demaine. Playing games with algorithms: algorithmic combinatorial game theory. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *LNCS*, pages 18–32. Springer, 2001.

[112] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. Technical Report 257, Department of Informatics, University of Bergen (Norway), 2003. To appear in Proc. LATIN 2004.

[113] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. In *International Colloquium on Automata, Languages and Programming ICALP*, volume 2719 of *LNCS*, pages 829–844. Springer, 2003.

[114] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. Technical Report 251, Department of Informatics, University of Bergen (Norway), 2003. To appear in Proc. SODA 2004.

[115] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J[ournal of] Discrete Math[ematics]*, 18:501–511, 2004.

[116] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded

genus and H-minor-free graphs. In *Symposium on Discrete Algorithms SODA*, pages 823–832. ACM, 2004.

[117] E. D. Demaine and M. Hajiaghayi. Fast algorithms for hard problems: bidimensionality, minors, and local treewidth. In *Graph Drawing GD*, volume 3383 of *LNCS*, pages 517–533. Springer, 2004.

[118] E. D. Demaine and M. Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *Symposium on Discrete Algorithms SODA*, pages 590–601. ACM Press, 2005.

[119] E. D. Demaine and M. Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth. In *Symposium on Discrete Algorithms SODA*, pages 682–689. ACM Press, 2005.

[120] E. D. Demaine, M. Hajiaghayi, and D. M. Thilikos. The bidimensional theory of bounded-genus graphs. In *Mathematical Foundations of Computer Science MFCS*, volume 3153 of *LNCS*, pages 191–203. Springer, 2004.

[121] E. D. Demaine and M. T. Hajiaghayi. Diameter and treewidth in minor-closed graph families, revisited. *Algorithmica*, 40:211–215, 2004.

[122] E. D. Demaine and M. T. Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *Symposium on Discrete Algorithms SODA*, pages 833–842. ACM, 2004.

[123] E. D. Demaine, M. T. Hajiaghayi, and D. Thilikos. Exponential speedup of fixed parameter algorithms on $k_{3,3}$-minor-free or k_5-minor-free graphs. Technical Report MIT-LCS-TR-838, MIT, March 2002. Abstract version in P. Bose and P. Morin (editors): 13th Annual International Symposium on Algorithms and Computation, ISAAC 2002, volume 2518 of *LNCS*, pp. 262–273. Springer-Verlag, 2002.

[124] Z. Deng. Exploiting parse trees for graphs of bounded treewidth. Master's thesis, University of Auckland, 2001.

[125] D. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[126] J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

[127] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors. *Amer. Journal Math.*, 35:413–422, 1913.

[128] H. Djidjev and S. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.

[129] H. Djidjev and I. Vrt´o. Crossing numbers and cutwidths. *Journal of Graph Algorithms and Applications*, 7:245–251, 2003.

[130] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM J. Algebraic Discrete Methods*, 3(2):229–240, 1982.

[131] H. N. Djidjev. A separator theorem for graphs of fixed genus. *Serdica*, 11:319–329, 1985.

[132] F. Dorn. Tuning algorithms for hard planar graph problems. Master's thesis, Universität Tübingen, January 2003.

[133] F. Dorn. Special branch decomposition: A natural link between tree decompositions and branch decompositions. Diplomarbeit, Universität Tübingen, April 2004.

[134] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[135] R. G. Downey, M. R. Fellows, and V. Raman. The complexity of irredundant set parameterized by size. *Discrete Applied Mathematics*, 100:155–167, 2000.

[136] R. G. Downey, M. R. Fellows, and U. Stege. Computational tractability: the view from Mars. *EATCS Bulletin*, 69:49–99, 1999.

[137] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99. 1999.

[138] R. G. Downey and C. McCartin. Some new directions and questions in parameterized complexity. In C. S. Calude, E. Calude, and M. J. Dinneen, editors, *Developments in Language Theory: 8th International Conference, DLT 2004*, volume 3340 of *LNCS*, pages 12–26. Springer, 2004.

[139] R. G. Downey and C. McCartin. Bounded persistence pathwidth. In M. Atkinson and F. Dehne, editors, *Eleventh Computing: The Australasian Theory Symposium (CATS2005)*, volume 41 of *CRPIT*, pages

51–56, Newcastle, Australia, 2005. ACS (Australian Computer Society).

[140] F. Drewes. Tree-based picture generation. *Theoretical Computer Science*, 246:1–51, 2000.

[141] P. A. Dreyer. *Applications and Variations of Domination in Graphs.* PhD thesis, Rutgers University, New Jersey, USA, PhD Thesis, 2000.

[142] V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. In F. Meyer auf der Heide, editor, *9th Annual European Symposium on Algorithms ESA*, volume 2161 of *LNCS*, pages 488–499. Springer, 2001.

[143] V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. A fixed-parameter approach to two-layer planarization. In Mutzel et al. [299], pages 1–15.

[144] V. Dujmović, H. Fernau, and M. Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In G. Liotta, editor, *Graph Drawing, 11th International Symposium GD 2003*, volume 2912 of *LNCS*, pages 332–344. Springer, 2004.

[145] V. Dujmovič and S. Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. In M. T. Goodrich and S. G. Kobourov, editors, *Graph Drawing GD 2002*, volume 2528 of *LNCS*, pages 118–129. Springer, 2002.

[146] V. Dujmović and S. Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40:15–32, 2004.

[147] T. Dwyer and F. Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation. In *Proc. Australasian Symp. on Information Visualisation (InVis.au 2004)*, volume 35 of *CPRIT*, pages 109–115, 2004.

[148] P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 13:361–374, 1994.

[149] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[150] T. Easton, S. B. Horton, and R. G. Parker. A solvable case of the optimal linear arrangement problem on halin graphs. *Congressus Numerantium (Utilitas mathematica)*, 119:3–17, 1996.

[151] T. Eiter. Exact transversal hypergraphs and application to boolean μ-functions. *Journal of Symbolic Computation*, 17(3):215–225, 1994.

[152] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[153] T. Eiter and G. Gottlob. Hypergraph transversal computation and related problems in logic and ai. In *Proc. 8th European Conference on Logics in Artificial Intelligence JELIA*, volume 2424 of *LNCS*, pages 549–564. Springer, 2002.

[154] J. Ellis, H. Fan, and M. Fellows. The dominating set problem is fixed parameter tractable for graphs of bounded genus. *Journal of Algorithms*, 52:152–168, 2004.

[155] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proc. 15th Symp. Discrete Algorithms SODA*, pages 781–790. ACM and SIAM, January 2004.

[156] C. Erbas, M. M. Tanik, and Z. Aliyazicioglu. Linear conguence equations for the solutions of the n-queens problem. *Information Processing Letters*, 41:301–306, 1992.

[157] V. Estivill-Castro and L. Branković. Data swapping: balancing privacy against precision in mining for logic rules. In *Data Warehousing and Knowledge Discovery DaWaK'99*, volume 1676 of *LNCS*, pages 389–398. Springer, 1999.

[158] R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int'l Test Conf.*, pages 49–55, 1981.

[159] B.-J. Falkowski and L. Schmitz. A note on the queens' problem. *Information Processing Letters*, 23:39–46, 1986.

[160] S. Fedin and A. Kulikov. A $2^{|E|/4}$ algorithm for MAXCUT. *Zapiski Nauchnov Seminarov POMI; English translation to appear in: Journal of Mathematical Sciences*, 293:129–138, 2002.

[161] S. S. Fedin and A. S. Kulikov. Automated proofs of upper bounds on the running time of splitting algorithms. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 248–259. Springer, 2004.

[162] A. Felfernig, G. E. Friedrich, D. Janach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. In *Proc. 14th European Conf. on Artif. Intell. ECAI 2000*, pages 146–150. IOS Press, 2000.

[163] M. Fellows. Parameterized complexity: the main ideas and connections to practical computing. *Electronic Notes in Theoretical Computer Science*, 61, 2002.

[164] M. Fellows. Blow-ups, win/win's, and crown rules: some new directions in FPT. In H. L. Bodlaender, editor, *WG 2003*, volume 2880 of *LNCS*, pages 1–12, 2003.

[165] M. Fellows. New directions and new challenges in algorithm design and complexity, parameterized. In F. Dehne, J. R. Sack, and M. Smid, editors, *Algorithms and Data Structures, 8th International Workshop, WADS 2003*, volume 2748 of *LNCS*, pages 505–520. Springer, 2003.

[166] M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, and J. A. Telle. Exact algorithms for finding k disjoint triangles in an arbitrary graph. In Hromkovic et al. [242], pages 235–244.

[167] M. Fellows, S. Szeider, and G. Wrightson. On finding short resolution refutations and small unsatisfiable subsets. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 223–234. Springer, 2004.

[168] M. R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. A. Rosamond, U. Stege, D. M. Thilikos, and S. Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. In S. Albers and T. Radzik, editors, *ESA*, volume 3221 of *LNCS*, pages 311–322. Springer, 2004.

[169] M. R. Fellows and M. A. Langston. On well-partial-order theory and its applications to combinatorial problems in VLSI design. *SIAM J[ournal of] Discrete Math[ematics]*, 5(1):117–126, 1992.

[170] M. R. Fellows and C. McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298:317–324, 2003.

[171] H. Fernau. On parameterized enumeration. In O. H. Ibarra and L. Zhang, editors, *Computing and Combinatorics, Proceedings COCOON 2002*, volume 2383 of *LNCS*, pages 564–573. Springer, 2002.

[172] H. Fernau. Complexity of a $\{0,1\}$-matrix problem. *The Australasian Journal of Combinatorics*, 29:273–300, 2004.

[173] H. Fernau. Extracting minimum length Document Type Definitions in NP-hard. In G. Paliouras and Y. Sakakibara, editors, *Grammatical Inference: Algorithms and Applications; 7th International Colloquium ICGI*, volume 3264 of *LNCS/LNAI*, pages 277–278. Springer, 2004.

[174] H. Fernau. Parametric duality: Kernel sizes and algorithmics. Technical Report TR04-027, Electronic Colloquium on Computational Complexity ECCC, 2004.

[175] H. Fernau. A top-down approach to search-trees: Improved algorithmics for 3-Hitting Set. Technical Report TR04-073, Electronic Colloquium on Computational Complexity ECCC, 2004.

[176] H. Fernau. Two-layer planarization: Improving on parameterized algorithmics. Technical Report TR04-078, Electronic Colloquium on Computational Complexity ECCC, 2004.

[177] H. Fernau. Two-layer planarization: Improving on parameterized algorithmics. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM*, volume 3381 of *LNCS*, pages 137–146. Springer, 2005.

[178] H. Fernau, T. Hagerup, N. Nishimura, P. Ragde, and K. Reinhardt. On the parameterized complexity of a generalized rush hour puzzle. In *Canadian Conference on Computational Geometry, CCCG 2003*, pages 6–9, 2003.

[179] H. Fernau and D. Juedes. A geometric approach to parameterized algorithms for domination problems on planar graphs. In *Mathematical Foundations of Computer Science MFCS 2004*, volume 3153 of *LNCS*, pages 488–499. Springer, 2004.

[180] H. Fernau and R. Niedermeier. An efficient exact algorithm for constraint bipartite vertex cover. *Journal of Algorithms*, 38(2):374–410, 2001.

[181] A. Fijany, F. Vatan, A. Barrett, and R. Mackey. New approaches for solving the diagnosis problem. Technical Report IPN Progress Report 42–149, Jet Propulsion Laboratory, California Institute of Technology, May 2002.

[182] D. C. Fisher, K. Fraughnaugh, and S. M. Seager. Domination of graphs with maximum degree three. Technical Report UCD-CCM-090, 1, 1996.

[183] P. Flajolet and Sedgewick. Analytic combinatorics, chapters i-ix. to obtain from `http://algo.inria.fr/flajolet/Publications/books.html`, November 2004.

[184] G. W. Flake and E. B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". Available at `http://www.neci.nj.nec.com/homepages/flake/rushhour.ps`, 2001.

[185] J. Flum and M. Grohe. Parameterized complexity and subexponential time. *EATCS Bulletin*, 84:71–100, 2004.

[186] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM Journal Comput.*, 33:892–922, 2004.

[187] F. Fomin, D. Kratsch, and G. Woeginger. Exact (exponential) algorithms for the dominating set problem. In Hromkovic et al. [242], pages 245–256.

[188] F. V. Fomin and A. V. Pyatkin. Finding minimum feedback vertex set in bipartite graph. Technical Report 291, Department of Informatics, University of Bergen (Norway), February 2005.

[189] F. V. Fomin and D. Thilikos. Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 2004.

[190] F. V. Fomin and D. Thilikos. A simple and fast approach for solving problems on planar graphs. In V. Diekert and M. Habib, editors, *Symposium on Theoretical Aspects of Computer Science STACS*, LNCS, pages 56–67. Springer, 2004.

[191] F. V. Fomin and D. M. Thilikos. Dominating sets and local treewidth. In *Proceedings of the 11th Annual European Symposium on Algorithms, ESA 2003*, LNCS. Springer, 2003.

[192] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003*, pages 168–177, 2003.

[193] F. V. Fomin and D. M. Thilikos. A simple and fast approach for solving problems on planar graphs. Technical Report 258, Department of Informatics, University of Bergen (Norway), 2003. To appear in Proc. STACS 2004.

[194] G. N. Frederickson and S. E. Hambrusch. Planar linear arrangements of outerplanar graphs. *IEEE Trans. Circuits Syst.*, 35:323–332, 1988.

[195] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. In *26th ICALP'99*, volume 1644, pages 331–340, 1999.

[196] P. Fröhlich. *DRUM-II: Efficient Model-Based Diagnosis of Technical Systems*. Phd thesis, Universität Hannover, 1998.

[197] M. P.J. Fromherz and M. H. Shirley. Supporting service technicians: Model-based diagnosis in context. In *Workshop on AI in Service and Support at AAAI'93*, 1993.

[198] D. Fukagawa and T. Akutsu. Fast algorithms for comparison of similar unordered trees. In R. Fleischer and G. Trippen, editors, *Algorithms and Computation: 15th International Symposium ISAAC 2004*, volume 3341 of *LNCS*, pages 452–463. Springer, 2004.

[199] M. R. Garey and D. S. Johnson. *Computers and Intractability.* New York: Freeman, 1979.

[200] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[201] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming.* John Wiley & Sons, 1972.

[202] D. K. Garnick and N. A. Nieuwejaar. Total domination of the $m \times n$ chessboard by kings, crosses, and knights. *Ars Combinatoria*, 41:45–56, 1995.

[203] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: learning document type descriptors from XML document collections. *Data Mining and Knowledge Discovery*, 7:23–56, 2003.

[204] C. F. Gauß. *Werke*, volume 12. Berlin: Julius Springer, 1929.

[205] J. Gimbel and C. Thomassen. Coloring triangle-free graphs with fixed size. *Discrete Mathematics*, 219:275–277, 2000.

[206] M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998.

[207] M. K. Goldberg, T. H. Spencer, and D. A. Berque. A low-exponential algorithm for counting vertex covers. *Graph Theory, Combinatorics, Algorithms, and Applications*, 1:431–444, 1995.

[208] C. P. Gomes. Structure, duality, and randomization — common themes in ai and or. invited talk (invited plenary talk). In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, 2000.

[209] R. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Reading, MA: Addison-Wesley, 3. edition, 1989.

[210] J. Gramm. *Fixed-Parameter Algorithms for the Consensus Analysis of Genomic Data.* Dissertation, Wilhelm-Schickard-Institut für Informatik, UniversitäT Tübingen, 2003.

[211] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39:321–347, 2004.

[212] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT. *Discrete Applied Mathematics*, 130:139–155, 2003.

[213] J. Gramm and R. Niedermeier. Quartet inconsistency is fixed parameter tractable. In A. Amir and G. M. Landau, editors, *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, volume 2089 of *LNCS*, pages 241–256. Springer, 2001.

[214] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41:79–88, 1989.

[215] M. Grohe. Descriptive and parameterized complexity. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic CSL'99*, volume 1683 of *LNCS*, pages 14–31. Springer, 1999.

[216] M. Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, To appear, 2001.

[217] M. Grohe and J. Mariño. Definability and descriptive complexity on databases with bounded tree-width. In Beeri and Bunemann, editors, *Proc. 7th International Conference on Database Theory*, volume 1540 of *LNCS*, pages 70–82, 1999.

[218] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Reseach*, 32:1195–1220, 1984.

[219] H. Grötzsch. Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel. *Wiss. Zeitschrift der Martin-Luther-Univ. Halle-Wittenberg, Math.-Naturwiss. Reihe*, 8:109–120, 1959.

[220] J. Guo, F. Hüffner, and R. Niedermeier. A structural view on parameterizing problems: distance from triviality. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004.

[221] G. Gutin, T. Kloks, and C. Lee. Kernels in planar digraphs. Technical report, Optimization Online, Mathematical Programming Society, Philadelphia, 2001.

[222] R. W. Haddad, A. T. Dahbura, and A. B. Sharma. Increased throughput for the testing and repair of RAMs with redundancy. *IEEE Transactions on Computers*, 40(2):154–166, February 1991.

[223] F. Hadlock. Finding a maximum cut in a planar graph in polynomial time. *SIAM Journal Comput.*, 4:221–225, 1975.

[224] R. Haenni. Generating diagnosis from conflict sets. Technical report, www.aaai.org, 1997.

[225] T. Hagerup. Dynamic algorithms for graphs of bounded treewidth. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, Proc. 24th ICALP*, volume 1256 of *LNCS*, pages 292–302, 1997.

[226] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal Comput.*, 31:1608–1623, 2002.

[227] K. Handa and K. Haruki. A reconfiguration algorithm for memory arrays containing faulty spares. *IEICE Trans. Fundamentals*, E83-A(6):1123–1130, 2000.

[228] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*, volume 208 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, 1998.

[229] T. W. Haynes and M. A. Henning. Changing and unchanging domination: a classification. *Discrete Mathematics*, 272:65–79, 2003.

[230] S. T. Hedetniemi and R. C. Laskar. Bibliography on domination in graphs and some basic definitions of domination parameters. *Discrete Mathematics*, 86:257–277, 1990.

[231] M. A. Henning. Defending the Roman Empire from multiple attacks. *Discrete Applied Mathematics*, 271:101–115, 2003.

[232] D. Hochbaum. Approximating clique and biclique problems. *Journal of Algorithms*, 29:174–200, 1998.

[233] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal Comput.*, 11(3):555–556, 1982.

[234] D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In K. Jansen and D. S. Hochbaum, editors, *Approximation Algorithms for Combinatorial Optimization, International Workshop, Proceedings of APPROX'98*, volume 1444 of *LNCS*, pages 111–122. Springer, 1998.

[235] M. Holzer and S. Schwoon. Assembling molecules in atomix is hard. *Theoretical Computer Science*, 303(3):447–462, 2004.

[236] J. N. Hooker, G. Ottosson, E. S. Thorsteinsson, and H.-J. Kim. A scheme for unifying optimization and constraint satisfaction methods.

Knowledge Engineering Review, special issue on AI/OR, 15(1):11–30, 2000.

[237] P. Horak, L. Branković, and M. Miller. A combinatorial problem in database security. *Discrete Applied Mathematics*, 91:119–126, 1999.

[238] J. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM J[ournal of] Discrete Math[ematics]*, 6:375–387, 1993.

[239] S. B. Horton. *The optimal linear arrangement problem: algorithms and approximation.* Phd thesis, School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, GA, USA, 1997.

[240] S. B. Horton, T. Easton, and R. G. Parker. The linear arrangement problem on recursively constructed graphs. *Networks*, 42(3):165–168, 2003.

[241] S. B. Horton, R. G. Parker, and R. B. Borie. On minimum cuts and the linear arrangement problem. *Discrete Applied Mathematics*, 103:127–139, 2000.

[242] J. Hromkovic et al., editors. *30th International Workshop on Graph-Theoretic Concepts in Computer Science WG 2004*, volume 3353 of *LNCS*. Springer, 2004.

[243] F. Hüffner. Algorithm engineering for optimal graph bipartization. In *Workshop on Algorithm Engineering WEA*, LNCS, page To appear, 2005.

[244] F. Hüffner, S. Edelkamp, H. Fernau, and R. Niedermeier. Finding optimal solutions to atomix. In F. Baader, G. Brewka, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence*, volume 2174 of *LNCS/LNAI*, pages 229–243. Springer, 2001.

[245] R. Hyafil and R. L. Rivest. Constructing optimal binary trees is NP-complete. *Information Processing Letters*, 5:15–17, 1976.

[246] T. Jian. An $O(2^{0.304n})$-algorithm for solving maximum independent set problem. *IEEE Transactions on Computers*, C-35(9):847–851, 1986.

[247] D. Juedes, B. Chor, and M. Fellows. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In Hromkovic et al. [242], pages 257–269.

[248] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1:1–25, 1997.

[249] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36:153–168, 1992.

[250] A. Kanevsky. Finding all mimimum-size separating vertex sets in a graph. *Networks*, 23:533–541, 1993.

[251] I. Kanj. *Vertex Cover: Exact and Approximation Algorithms and Applications.* Phd thesis, Texas A& M University, 2001.

[252] I. A. Kanj and L. Perkovič. Improved parameterized algorithms for planar dominating set. In K. Diks and W. Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *LNCS*, pages 399–410. Springer, 2002.

[253] I. A. Kanj and L. Perkovič. A note on Baker's algorithm. Technical Report 04-006, DePaul University, November 2004.

[254] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal Comput.*, 28(5):1906–1922, October 1999.

[255] G. Karakostas. A better approximation ratio for the vertex cover problem. Technical Report ECCC Report TR04-084, ECCC Trier, 2004.

[256] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proc. 28th ACM Symp. Theory of Computing*, pages 329–337, 1996.

[257] S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289:997–1008, 2002.

[258] N. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Information Processing Letters*, 142:345–350, 1992.

[259] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[260] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–129, 1987.

[261] T. Kloks. *Treewidth. Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

[262] T. Kloks, C. M. Lee, and J. Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In L. Kučera, editor, *Graph-Theoretic Concepts in Computer Science WG 2002*, volume 2537 of *LNCS*, pages 282–295. Springer, 2002.

[263] J. Kohlas, D. Berzati, and R. Haenni. Probabilistic argumentation systems and abduction. *Annals of Mathematics and Artificial Intelligence, Special Issue (AMAI)*, 34:177–195, 2002.

[264] D. König. Über Graphen und Matrizen. *Matematikai és Fizikai Lápok*, 38:116–119, 1931.

[265] H. Kronk and J. Mitchem. A seven-color theorem on the sphere. *Discrete Mathematics*, 5:255–260, 1973.

[266] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[267] S.-Y. Kuo and W.K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, 4:24–31, February 1987.

[268] S. Langerman and W. Steiger. Optimization in arrangements. In H. Alt and M. Habib, editors, *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *LNCS*, pages 50–61. Springer, 2003.

[269] E. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal Comput.*, 9:558–565, 1980.

[270] C. Letavec and J. Ruggiero. The n-queens problem. *INFORMS Transactions on Education*, 2(3):101–103, 2002.

[271] M. E. Levitt. Designing UltraSparc for testability. *IEEE Design and Test*, 14(1):10–17, 1997.

[272] P. Liberatore. Monotonic reductions, representative equivalence, and compilation of intractable problems. *Journal of the ACM*, 48:1091–1125, 2001.

[273] L. Lin and Y. Jiang. The computation of hitting sets: review and new algorithms. *Information Processing Letters*, 86:177–184, 2003.

[274] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

[275] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J[ournal of] Appl[ied] Math[ematics]*, 36(2):177–189, 1979.

[276] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal Comput.*, 9(3):615–627, 1980.

[277] W. Liu and A. Vannelli. Generating lower bounds for the linear arrangement problems. *Discrete Applied Mathematics*, 59:137–151, 1995.

[278] F. Lombardi and W. K. Huang. Approaches to the repair of VLSI/WSI PRAMs by row/column deletion. In *International Symposium on Fault-Tolerant Computing (FTCS '88)*, pages 342–347, Washington, D.C., USA, June 1988. IEEE Computer Society Press.

[279] C. P. Low and H. W. Leong. A new class of efficient algorithms for reconfiguration of memory arrays. *IEEE Transactions on Computers*, 45(5):614–618, 1996.

[280] V.V. Lozin and D. Rautenbach. On the band-, tree- and clique-width of graphs with bounded vertex degree. *SIAM J[ournal of] Discrete Math[ematics]*, 18:195–206, 2004.

[281] C. L. Lu, M.-T. Ko, and C. Y. Tang. Perfect edge domination and efficient edge domination in graphs. *Discrete Applied Mathematics*, 119(3):227–250, 2002.

[282] A. Lubiw. The boolean basis problem and how to cover some polygons by rectangles. *SIAM J[ournal of] Discrete Math[ematics]*, 3:98–115, 1990.

[283] D. Wagner M. Kaufmann. *Drawing Graphs, Methods and Models*, volume 2025 of *LNCS*. Springer, 2001.

[284] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. Technical Report TR97-033, ECCC Trier, 1997.

[285] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[286] D. Marx. Parameterized complexity of constraint satisfaction problems. In *19th IEEE Annual Conference on Computational Complexity (CCC'04)*, pages 139–149. IEEE, 2004.

[287] S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39:124–127, 1990.

[288] L. R. Matheson and R. E. Tarjan. Dominating sets in planar graphs. *European Journal of Combinatorics*, 17:565–568, 1996.

[289] L. Mathieson, E. Prieto, and C. Sloper. Packing edge-disjoint triangles: a paramaterized view. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 127–137. Springer, 2004.

[290] C. McCartin. Parameterized counting problems. In K. Diks and W. Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002*, volume 2420 of *LNCS*, pages 556–567. Springer, 2002.

[291] B. McCuaig and B. Shepherd. Domination in graphs of minimum degree two. *Journal of Graph Theory*, 13:749–762, 1989.

[292] K. Mehlhorn. *Graph algorithms and NP-completeness.* Heidelberg: Springer, 1984.

[293] K. Mehlhorn and S. Näher. *LEDA: A Platform of Combinatorial and Geometric Computing.* Cambridge University Press, Cambridge, England, 1999.

[294] B. Mohar. Face cover and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82:102–117, 2001.

[295] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.

[296] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[297] B. Monien and I. H. Sudborough. Bandwidth constrained NP-complete problems. In *Proc. 13th Ann. Symp. Theory of Computing STOC*, pages 207–217. ACM Press, 1981.

[298] X. Muñoz, W. Unger, and I. Vrt'o. One sided crossing minimization is NP-hard for sparse graphs. In Mutzel et al. [299], pages 115–123.

[299] P. Mutzel, M. Jünger, and S. Leipert, editors. *9th International Symp. on Graph Drawing GD'01*, volume 2265 of *LNCS*. Springer, 2002.

[300] Petra Mutzel. An alternative method to crossing minimization on hierarchical graphs. In Stephen C. North, editor, *Graph Drawing*, volume 1190 of *LNCS*, pages 318–333, 1997.

[301] H. Nagamochi. An improved approximation to the one-sided bilayer drawing. In G. Liotta, editor, *Graph Drawing, 11th International Symposium GD 2003*, volume 2912 of *LNCS*, pages 406–418. Springer, 2004.

[302] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001.

[303] K. S. Natarajan and L. J. White. Optimum domination in weighted trees. *Information Processing Letters*, 7:261–265, 1978.

[304] G. L. Nemhauser and Jr. L. E. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[305] T. A. J. Nicholson. Permutation procedure for minimizing the number of crossings in a network. *Proc IEE*, 115:21–26, 1968.

[306] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Habilitationsschrift, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany, October 2002.

[307] R. Niedermeier. Ubiquitous parameterization—invitation to fixed-parameter algorithms. In *Mathematical Foundations of Computer Science MFCS 2004*, volume 3153 of *LNCS*, pages 84–103. Springer, 2004.

[308] R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1:89–102, 2003.

[309] R. Niedermeier and P. Rossmanith. On efficient fixed parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47:63–77, 2003.

[310] N. Nishimura, P. Ragde, and D. Thilikos. Smaller kernels for hitting set problems of constant arity. In R. Downey, M. Fellows, and F. Dehne,

editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 121–126. Springer, 2004.

[311] R. Z. Norman and M. O. Rabin. An algorithm for a minimum cover of a graph. *Proceedings of the American Mathematical Society*, 10:315–319, 1959.

[312] O. Obst. Using model-based diagnosis to build hypotheses about spatial environments. Technical Report Fachberichte Informatik Nr. 5, Universität Koblenz-Landau, 19.

[313] M. Okun. On approximation of the vertex cover problem in hypergraphs. *Discrete Optimization*, 2:101–111, 2005.

[314] O. Ore. *Theory of Graphs*, volume XXXVIII of *Colloquium Publications*. American Mathematical Society, 1962.

[315] P. R. J. Östergård and W. D. Weakley. Values of domination numbers of the queen's graph. *Electronic J. Combin.*, 8:#R29, 19pp., 2001.

[316] S.-I. Oum and P. Seymour. Approximating clique-width and branchwidth. see `http://www.math.princeton.edu/~sangil/rwdfpt.pdf`, 2005.

[317] J. Pach and G. Tóth. Which crossing number is it anyway? *Journal of Combinatorial Theory, Series B*, 80:225–246, 2000.

[318] A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, Roman domination and other dominating set variants. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Networking and Mobile Computing, IFIP 17th World Computer Congress — TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science IFIP TCS*, pages 280–291. Kluwer, 2002. Also available as Technical Report 365, ETH Zürich, Institute of Theoretical Computer Science, 10/2001.

[319] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[320] V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, 1997.

[321] J. Petit. Experiments on the minimum linear arrangement problem. *J. Exp. Algorithmics*, 8, 2003.

[322] E. Prieto. The method of extremal structure on the k-maximum cut problem. In M. Atkinson and F. Dehne, editors, *Eleventh Computing: The Australasian Theory Symposium (CATS2005)*, volume 41 of *CRPIT*, pages 119–126, Newcastle, Australia, 2005. ACS (Australian Computer Society).

[323] E. Prieto. *Systematic Kernelization in FPT Algorithm Design.* PhD thesis, The University of Newcastle, Australia, 2005.

[324] E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing FPT-algorithms—the case of k-internal spanning tree. In *Proceedings of WADS 2003, Workshop on Algorithms and Data Structures*, volume 2748 of *LNCS*, pages 465–483. Springer, 2003.

[325] E. Prieto and C. Sloper. Looking at the stars. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 138–148. Springer, 2004.

[326] H. J. Prömel and A. Steger. *The Steiner Tree Problem; a Tour through Graphs, Algorithms, and Complexity.* Vieweg, 2002.

[327] D. Ratner and M. K. Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990.

[328] S. S. Ravi and H. B. Hunt. An application of the planar separator theorem to counting problems. *Information Processing Letters*, 25(6):317–322, 1987.

[329] B. Reed. Paths, stars, and the number three. *Combinatorics, Probability and Computing*, 5:277–295, 1996.

[330] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32:299–301, 2004.

[331] M. Reichling. A simplified solution of the n queens' problem. *Information Processing Letters*, 25:253–255, 1987.

[332] G. Reinelt. *The Linear Ordering Problem: Algorithms and Applications.* Heldermann, 1985.

[333] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[334] C. S. ReVelle and K. E. Rosing. Defendens imperium Romanum: A classical problem in military strategy. *American Mathematical Monthly*, 107:585–594, 2000.

[335] G. Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abhandlungen des Mathematischen Seminars der Universität Hamburg*, 29:107–117, 1965.

[336] I. Rivin, I. Vardi, and P. Zimmerman. The n-queens problem. *American Mathematical Monthly*, 101(7):629–639, 1994.

[337] I. Rivin and R. Zabih. A dynamic programming solution to the n-queens problem. *Information Processing Letters*, 41:253–256, 1992.

[338] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas. The four colour theorem. *Journal of Combinatorial Theory, Series B*, 70:2–44, 1997.

[339] N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62:323–348, 1994.

[340] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[341] N. Robertson and P. D. Seymour. Graph minors IV: tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48:227–254, 1990.

[342] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, September 1986.

[343] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux I, Talence, January 2001.

[344] N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optimization*, 1:215–228, 2004.

[345] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Ann. ACM Symp. Theory of Computing STOC*, pages 216–226. ACM Press, 1978.

[346] U. Schöning. Algorithms in exponential time. In V. Diekert and B. Durand, editors, *Symposium on Theoretical Aspects of Computer Science STACS 2005*, volume 3404 of *LNCS*, pages 36–43. Springer, 2005.

[347] H. Schröder, A. E. May, I. Vr´to, and O. Sýkora. Approximation algorithms for the vertex bipartization problem. In *SOFSEM '97: Proceedings of the 24th Seminar on Current Trends in Theory and Practice of Informatics*, volume 1338 of *LNCS*, pages 547–554. Springer, 1997.

[348] S. E. Schuster. Multiple word/bit line redundancy for semiconductor memories. *IEEE J. Solid-State Circuits*, 13(5):698–703, 1978.

[349] F. Shahrokhi, O. Sýkora, L. Székely, and I. Vrťo. Crossing numbers: bounds and applications, 1997.

[350] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vr´to. On bipartite drawings and the linear arrangement problem. *SIAM Journal Comput.*, 30(6):1773–1789, 2001.

[351] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. In L. Kučera, editor, *Graph-Theoretic Concepts in Computer Science WG 2002*, volume 2573 of *LNCS*, pages 379–390. Springer, 2002.

[352] Y. Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal Comput.*, 8:15–32, 1979.

[353] D. A. Spielman and S.-H. Teng. Disk packings and planar separators. In *SCG 96: 12th Annual ACM Symposium on Computational Geometry*, pages 349–358, 1996.

[354] U. Stege, I. van Rooij, A. Hertel, and P. Hertel. An $O(pn + 1.151p)$-algorithm for p-profit cover and its practical implications for vertex cover. In P. Bose and P. Morin, editors, *Algorithms and Computation, 13th International Symposium, ISAAC 2002*, volume 2518 of *LNCS*, pages 249–261. Springer, 2002.

[355] F. Steimann, P. Fröhlich, and W. Nejdl. Model-based diagnosis for open systems fault management. *AI Communications*, 12:5–17, 1999.

[356] I. Stewart. Defend the Roman Empire. *Scientific American*, pages 136,137,139, December 1999.

[357] M. Suderman and S. Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. In G. Liotta, editor, *Graph Drawing GD*, volume 2912 of *LNCS*, pages 345–356. Springer, 2003.

[358] S. Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *Journal of Computer and System Sciences*, 69:656–674, 2004.

[359] S. Szeider. On fixed-parameter tractable parameterizations of sat. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 188–202. Springer, 2004.

[360] L. A. Székely. A successful concept for measuring non-planarity of graphs: the crossing number. *Discrete Mathematics*, 276:331–352, 2004.

[361] L. A. Székely. Progress on crossing number problems. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM*, volume 3381 of *LNCS*, pages 53–61. Springer, 2005.

[362] R. E. Tarjan and A. E. Trojanowski. Finding a maximum independent set. *SIAM Journal Comput.*, 6:537–546, 1977.

[363] J. A. Telle. Complexity of domination-type problems in graphs. *Nordic J. of Comp.*, 1:157–171, 1994.

[364] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In F. Dehne et al., editors, *Algorithms and Data Structures, Proc. 3rd WADS'93*, volume 709 of *LNCS*, pages 610–621, 1993.

[365] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J[ournal of] Discrete Math[ematics]*, 10(4):529–550, 1997.

[366] W. F. D. Theron and G. Geldenhuys. Domination by queens on a square beehive. *Discrete Mathematics*, 178:213–220, 1998.

[367] D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth. In F. Meyer auf der Heide, editor, *9th Annual European Symposium on Algorithms ESA*, volume 2161 of *LNCS*, pages 380–390, 2001.

[368] C. Thomassen. Grötzsch's 3-color theorem and its counterparts for the torus and the projective plane. *Journal of Combinatorial Theory, Series B*, 62:268–279, 1994.

[369] M. Thorup. Structured programs have small tree-width and good register allocation. In R. Möhring, editor, *Proc. 23rd Intern. Workshop Graph-Theor. Concepts in Comp. Sci. WG'97*, volume 1335 of *LNCS*, pages 318–332, 1997.

[370] S. Tsukiyama, H. Ariuoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal Comput.*, 16:505–517, 1977.

[371] R. Uehara. *Probabilistic Algorithms and Complexity Classes.* PhD thesis, Department of Computer Science and Information Mathematics, The University of Electro-Communications, Japan, March 1998.

[372] L. Volkmann. On graphs with equal domination and edge independence numbers. *Ars Combinatoria*, 41:45–56, 1995.

[373] M. Wahlström. Exact algorithms for finding minimum transversals in rank-3 hypergraphs. *Journal of Algorithms*, 51:107–121, 2004.

[374] L. Wang, Y. Li, D. Wijesekera, and S. Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. In *Computer Security — European Symposium on Research in Computer Security ESORICS 2003*, volume 2808 of *LNCS*, pages 100–115. Springer, 2003.

[375] K. Weihe. Covering trains by stations or the power of data reduction. In R. Battiti and A. A. Bertossi, editors, *Algorithms and Experiments ALEX 98*, pages 1–8. `http://rtm.science.unitn.it/alex98/proceedings.html`, 1998.

[376] S. Wernicke. On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems. Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, September 2003.

[377] M. Weyer. Bounded fixed-parameter tractability: the case $2^{\mathrm{poly}(k)}$. In R. Downey, M. Fellows, and F. Dehne, editors, *International Workshop on Parameterized and Exact Computation IWPEC 2004*, volume 3162 of *LNCS*, pages 49–60. Springer, 2004.

[378] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In M. Juenger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka! You shrink!*, volume 2570 of *LNCS*, pages 185–207. Springer, 2003.

[379] F. Wotawa. A variant of Reiter's hitting set algorithm. *Information Processing Letters*, 79:45–51, 2001.

[380] A. Yamaguchi and A. Sugimoto. An approximation algorithm for the two-layered graph drawing problem. In T. Asano et al., editors, *Proc. COCOON'99*, volume 1627 of *LNCS*, pages 81–91, 1999. A journal version appeared in Japanese in the *Journal of the Information Processing Society of Japan*, vol. 40, no. 10.

[381] A. Yamaguchi and H. Toh. Visualization of genetic networks: edge crossing minimization of a graph drawing with vertex pairs. *Genome Informatics*, 11:245–246, 2000.

[382] A. Yamaguchi and H. Toh. Two-layered genetic network drawings with minimum edge crossings. *Genome Informatics*, 12:456–457, 2001.

[383] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, June 1980.

[384] L. Youngs and S. Paramanandam. Mapping and repairing embedded-memory defects. *IEEE Design and Test*, 14(1):18–24, 1997.

Index