# Chapter 2
# Kernelization

*Kernelization is a systematic approach to study polynomial-time preprocessing algorithms. It is an important tool in the design of parameterized algorithms. In this chapter we explain basic kernelization techniques such as crown decomposition, the expansion lemma, the sunflower lemma, and linear programming. We illustrate these techniques by obtaining kernels for* Vertex Cover, Feedback Arc Set in Tournaments, Edge Clique Cover, Maximum Satisfiability, *and* $d$-Hitting Set.

Preprocessing (data reduction or kernelization) is used universally in almost every practical computer implementation that aims to deal with an NP-hard problem. The goal of a preprocessing subroutine is to solve efficiently the “easy parts” of a problem instance and reduce it (shrink it) to its computationally difficult “core” structure (the *problem kernel* of the instance). In other words, the idea of this method is to reduce (but not necessarily solve) the given problem instance to an equivalent “smaller sized” instance in time polynomial in the input size. A slower exact algorithm can then be run on this smaller instance.

How can we measure the effectiveness of such a preprocessing subroutine? Suppose we define a useful preprocessing algorithm as one that runs in polynomial time and replaces an instance $I$ with an equivalent instance that is at least one bit smaller. Then the existence of such an algorithm for an NP-hard problem would imply P= NP, making it unlikely that such an algorithm can be found. For a long time, there was no other suggestion for a formal definition of useful preprocessing, leaving the mathematical analysis of polynomial-time preprocessing algorithms largely neglected. But in the language of parameterized complexity, we can formulate a definition of useful preprocessing by demanding that large instances with a small parameter should be shrunk, while instances that are small compared to their parameter



M. Cygan et al., *Parameterized Algorithms*,
DOI 10.1007/978-3-319-21275-3_2

do not have to be processed further. These ideas open up the "lost continent" of polynomial-time algorithms called kernelization.

In this chapter we illustrate some commonly used techniques to design kernelization algorithms through concrete examples. The next section, Section 2.1, provides formal definitions. In Section 2.2 we give kernelization algorithms based on so-called natural reduction rules. Section 2.3 introduces the concepts of crown decomposition and the expansion lemma, and illustrates it on MAXIMUM SATISFIABILITY. Section 2.5 studies tools based on linear programming and gives a kernel for VERTEX COVER. Finally, we study the sunflower lemma in Section 2.6 and use it to obtain a polynomial kernel for $d$-HITTING SET.

## 2.1 Formal definitions

We now turn to the formal definition that captures the notion of kernelization. A *data reduction rule*, or simply, reduction rule, for a parameterized problem $Q$ is a function $\phi\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance $(I, k)$ of $Q$ to an equivalent instance $(I', k')$ of $Q$ such that $\phi$ is computable in time polynomial in $|I|$ and $k$. We say that two instances of $Q$ are *equivalent* if $(I, k) \in Q$ if and only if $(I', k') \in Q$; this property of the reduction rule $\phi$, that it translates an instance to an equivalent one, is sometimes referred to as the *safeness* or *soundness* of the reduction rule. In this book, we stick to the phrases: *a rule is safe* and *the safeness of a reduction rule*.

The general idea is to design a *preprocessing algorithm* that consecutively applies various data reduction rules in order to shrink the instance size as much as possible. Thus, such a preprocessing algorithm takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of $Q$, works in polynomial time, and returns an equivalent instance $(I', k')$ of $Q$. In order to formalize the requirement that the output instance has to be small, we apply the main principle of Parameterized Complexity: The complexity is measured in terms of the parameter. Consequently, the *output size* of a preprocessing algorithm $\mathcal{A}$ is a function $\mathrm{size}_{\mathcal{A}}\colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined as follows:

$$\mathrm{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' \;:\; (I', k') = \mathcal{A}(I, k),\ I \in \Sigma^*\}.$$

In other words, we look at all possible instances of $Q$ with a fixed parameter $k$, and measure the supremum of the sizes of the output of $\mathcal{A}$ on these instances. Note that this supremum may be infinite; this happens when we do not have any bound on the size of $\mathcal{A}(I, k)$ in terms of the input parameter $k$ only. *Kernelization algorithms* are exactly these preprocessing algorithms whose output size is finite and bounded by a computable function of the parameter.

**Definition 2.1 (Kernelization, kernel).** A *kernelization algorithm*, or simply a *kernel*, for a parameterized problem $Q$ is an algorithm $\mathcal{A}$ that, given an instance $(I, k)$ of $Q$, works in polynomial time and returns an equivalent instance $(I', k')$ of $Q$. Moreover, we require that $\text{size}_{\mathcal{A}}(k) \leq g(k)$ for some computable function $g\colon \mathbb{N} \to \mathbb{N}$.

The size requirement in this definition can be reformulated as follows: There exists a computable function $g(\cdot)$ such that whenever $(I', k')$ is the output for an instance $(I, k)$, then it holds that $|I'| + k' \leq g(k)$. If the upper bound $g(\cdot)$ is a polynomial (linear) function of the parameter, then we say that $Q$ admits a *polynomial (linear) kernel*. We often abuse the notation and call the output of a kernelization algorithm the "reduced" equivalent instance, also a kernel.

In the course of this chapter, we will often encounter a situation when in some boundary cases we are able to completely resolve the considered problem instance, that is, correctly decide whether it is a yes-instance or a no-instance. Hence, for clarity, we allow the reductions (and, consequently, the kernelization algorithm) to return a yes/no answer instead of a reduced instance. Formally, to fit into the introduced definition of a kernel, in such cases the kernelization algorithm should instead return a constant-size trivial yes-instance or no-instance. Note that such instances exist for every parameterized language except for the empty one and its complement, and can be therefore hardcoded into the kernelization algorithm.

Recall that, given an instance $(I, k)$ of $Q$, the size of the kernel is defined as the number of *bits* needed to encode the reduced equivalent instance $I'$ plus the parameter value $k'$. However, when dealing with problems on graphs, hypergraphs, or formulas, often we would like to emphasize other aspects of output instances. For example, for a graph problem $Q$, we could say that $Q$ admits a kernel with $\mathcal{O}(k^3)$ vertices and $\mathcal{O}(k^5)$ edges to emphasize the upper bound on the number of vertices and edges in the output instances. Similarly, for a problem defined on formulas, we could say that the problem admits a kernel with $\mathcal{O}(k)$ variables.

It is important to mention here that the early definitions of kernelization required that $k' \leq k$. On an intuitive level this makes sense, as the parameter $k$ measures the complexity of the problem — thus the larger the $k$, the harder the problem. This requirement was subsequently relaxed, notably in the context of lower bounds. An advantage of the more liberal notion of kernelization is that it is robust with respect to polynomial transformations of the kernel. However, it limits the connection with practical preprocessing. All the kernels mentioned in this chapter respect the fact that the output parameter is at most the input parameter, that is, $k' \leq k$.

While usually in Computer Science we measure the efficiency of an algorithm by estimating its running time, the central measure of the efficiency of a kernelization algorithm is a bound on its output size. Although the actual running time of a kernelization algorithm is often very important for practical applications, in theory a kernelization algorithm is only required to run in polynomial time.

If we have a kernelization algorithm for a problem for which there is some algorithm (with any running time) to decide whether $(I, k)$ is a yes-instance, then clearly the problem is FPT, as the size of the reduced instance $I$ is simply a function of $k$ (and independent of the input size $n$). However, a surprising result is that the converse is also true.

**Lemma 2.2.** *If a parameterized problem $Q$ is* FPT *then it admits a kernelization algorithm.*

*Proof.* Since $Q$ is FPT, there is an algorithm $\mathcal{A}$ deciding if $(I, k) \in Q$ in time $f(k) \cdot |I|^c$ for some computable function $f$ and a constant $c$. We obtain a kernelization algorithm for $Q$ as follows. Given an input $(I, k)$, the kernelization algorithm runs $\mathcal{A}$ on $(I, k)$, for at most $|I|^{c+1}$ steps. If it terminates with an answer, use that answer to return either that $(I, k)$ is a yes-instance or that it is a no-instance. If $\mathcal{A}$ does not terminate within $|I|^{c+1}$ steps, then return $(I, k)$ itself as the output of the kernelization algorithm. Observe that since $\mathcal{A}$ did not terminate in $|I|^{c+1}$ steps, we have that $f(k) \cdot |I|^c > |I|^{c+1}$, and thus $|I| < f(k)$. Consequently, we have $|I| + k \leq f(k) + k$, and we obtain a kernel of size at most $f(k) + k$; note that this upper bound is computable as $f(k)$ is a computable function. □

Lemma 2.2 implies that a decidable problem admits a kernel if and only if it is fixed-parameter tractable. Thus, in a sense, kernelization can be another way of defining fixed-parameter tractability.

However, kernels obtained by this theoretical result are usually of exponential (or even worse) size, while problem-specific data reduction rules often achieve quadratic ($g(k) = \mathcal{O}(k^2)$) or even linear-size ($g(k) = \mathcal{O}(k)$) kernels. So a natural question for any concrete FPT problem is whether it admits a problem kernel that is bounded by a polynomial function of the parameter ($g(k) = k^{\mathcal{O}(1)}$). In this chapter we give polynomial kernels for several problems using some elementary methods. In Chapter 9, we give more advanced methods for obtaining kernels.

## 2.2 Some simple kernels

In this section we give kernelization algorithms for Vertex Cover and Feedback Arc Set in Tournaments (FAST) based on a few natural reduction rules.

### 2.2.1 Vertex Cover

Let $G$ be a graph and $S \subseteq V(G)$. The set $S$ is called a *vertex cover* if for every edge of $G$ at least one of its endpoints is in $S$. In other words, the graph $G - S$ contains no edges and thus $V(G) \setminus S$ is an *independent set*. In the Vertex Cover problem, we are given a graph $G$ and a positive integer $k$ as input, and the objective is to check whether there exists a vertex cover of size at most $k$.

The first reduction rule is based on the following simple observation. For a given instance $(G, k)$ of Vertex Cover, if the graph $G$ has an isolated vertex, then this vertex does not cover any edge and thus its removal does not change the solution. This shows that the following rule is safe.

**Reduction VC.1.** If $G$ contains an isolated vertex $v$, delete $v$ from $G$. The new instance is $(G - v, k)$.

The second rule is based on the following natural observation:

> If $G$ contains a vertex $v$ of degree more than $k$, then $v$ should be in every vertex cover of size at most $k$.

Indeed, this is because if $v$ is not in a vertex cover, then we need at least $k + 1$ vertices to cover edges incident to $v$. Thus our second rule is the following.

**Reduction VC.2.** If there is a vertex $v$ of degree at least $k + 1$, then delete $v$ (and its incident edges) from $G$ and decrement the parameter $k$ by 1. The new instance is $(G - v, k - 1)$.

Observe that exhaustive application of reductions VC.1 and VC.2 completely removes the vertices of degree 0 and degree at least $k + 1$. The next step is the following observation.

> If a graph has maximum degree $d$, then a set of $k$ vertices can cover at most $kd$ edges.

This leads us to the following lemma.

**Lemma 2.3.** *If $(G, k)$ is a yes-instance and none of the reduction rules VC.1, VC.2 is applicable to $G$, then $|V(G)| \leq k^2 + k$ and $|E(G)| \leq k^2$.*

*Proof.* Because we cannot apply Reductions VC.1 anymore on $G$, $G$ has no isolated vertices. Thus for every vertex cover $S$ of $G$, every vertex of $G - S$ should be adjacent to some vertex from $S$. Since we cannot apply Reductions VC.2, every vertex of $G$ has degree at most $k$. It follows that

$|V(G-S)| \leq k|S|$ and hence $|V(G)| \leq (k+1)|S|$. Since $(G,k)$ is a yes-instance, there is a vertex cover $S$ of size at most $k$, so $|V(G)| \leq (k+1)k$. Also every edge of $G$ is covered by some vertex from a vertex cover and every vertex can cover at most $k$ edges. Hence if $G$ has more than $k^2$ edges, this is again a no-instance. □

Lemma 2.3 allows us to claim the final reduction rule that explicitly bounds the size of the kernel.

**Reduction VC.3.** Let $(G,k)$ be an input instance such that Reductions VC.1 and VC.2 are not applicable to $(G,k)$. If $k<0$ and $G$ has more than $k^2+k$ vertices, or more than $k^2$ edges, then conclude that we are dealing with a no-instance.

Finally, we remark that all reduction rules are trivially applicable in linear time. Thus, we obtain the following theorem.

**Theorem 2.4.** VERTEX COVER *admits a kernel with* $\mathcal{O}(k^2)$ *vertices and* $\mathcal{O}(k^2)$ *edges.*

### *2.2.2* FEEDBACK ARC SET IN TOURNAMENTS

In this section we discuss a kernel for the FEEDBACK ARC SET IN TOURNAMENTS problem. A *tournament* is a directed graph $T$ such that for every pair of vertices $u,v \in V(T)$, exactly one of $(u,v)$ or $(v,u)$ is a directed edge (also often called an *arc*) of $T$. A set of edges $A$ of a directed graph $G$ is called a *feedback arc set* if every directed cycle of $G$ contains an edge from $A$. In other words, the removal of $A$ from $G$ turns it into a directed acyclic graph. Very often, acyclic tournaments are called *transitive* (note that then $E(G)$ is a transitive relation). In the FEEDBACK ARC SET IN TOURNAMENTS problem we are given a tournament $T$ and a nonnegative integer $k$. The objective is to decide whether $T$ has a feedback arc set of size at most $k$.

For tournaments, the deletion of edges results in directed graphs which are not tournaments anymore. Because of that, it is much more convenient to use the characterization of a feedback arc set in terms of "reversing edges". We start with the following well-known result about *topological orderings* of directed acyclic graphs.

**Lemma 2.5.** *A directed graph $G$ is acyclic if and only if it is possible to order its vertices in such a way such that for every directed edge $(u,v)$, we have $u<v$.*

We leave the proof of Lemma 2.5 as an exercise; see Exercise 2.1. Given a directed graph $G$ and a subset $F \subseteq E(G)$ of edges, we define $G \circledast F$ to be the directed graph obtained from $G$ by reversing all the edges of $F$. That is, if $\mathrm{rev}(F) = \{(u,v) \ : \ (v,u) \in F\}$, then for $G \circledast F$ the vertex set is $V(G)$

and the edge set $E(G \circledast F) = (E(G) \cup \text{rev}(F)) \setminus F$. Lemma 2.5 implies the following.

**Observation 2.6.** Let $G$ be a directed graph and let $F$ be a subset of edges of $G$. If $G \circledast F$ is a directed acyclic graph then $F$ is a feedback arc set of $G$.

The following lemma shows that, in some sense, the opposite direction of the statement in Observation 2.6 is also true. However, the minimality condition in Lemma 2.7 is essential, see Exercise 2.2.

**Lemma 2.7.** *Let $G$ be a directed graph and $F$ be a subset of $E(G)$. Then $F$ is an inclusion-wise minimal feedback arc set of $G$ if and only if $F$ is an inclusion-wise minimal set of edges such that $G \circledast F$ is an acyclic directed graph.*

*Proof.* We first prove the forward direction of the lemma. Let $F$ be an inclusion-wise minimal feedback arc set of $G$. Assume to the contrary that $G \circledast F$ has a directed cycle $C$. Then $C$ cannot contain only edges of $E(G) \setminus F$, as that would contradict the fact that $F$ is a feedback arc set. Let $f_1, f_2, \cdots, f_\ell$ be the edges of $C \cap \text{rev}(F)$ in the order of their appearance on the cycle $C$, and let $e_i \in F$ be the edge $f_i$ reversed. Since $F$ is inclusion-wise minimal, for every $e_i$, there exists a directed cycle $C_i$ in $G$ such that $F \cap C_i = \{e_i\}$. Now consider the following closed walk $W$ in $G$: we follow the cycle $C$, but whenever we are to traverse an edge $f_i \in \text{rev}(F)$ (which is not present in $G$), we instead traverse the path $C_i - e_i$. By definition, $W$ is a closed walk in $G$ and, furthermore, note that $W$ does not contain any edge of $F$. This contradicts the fact that $F$ is a feedback arc set of $G$.

The minimality follows from Observation 2.6. That is, every set of edges $F$ such that $G \circledast F$ is acyclic is also a feedback arc set of $G$, and thus, if $F$ is not a minimal set such that $G \circledast F$ is acyclic, then it will contradict the fact that $F$ is a minimal feedback arc set.

For the other direction, let $F$ be an inclusion-wise minimal set of edges such that $G \circledast F$ is an acyclic directed graph. By Observation 2.6, $F$ is a feedback arc set of $G$. Moreover, $F$ is an inclusion-wise minimal feedback arc set, because if a proper subset $F'$ of $F$ is an inclusion-wise minimal feedback arc set of $G$, then by the already proved implication of the lemma, $G \circledast F'$ is an acyclic directed graph, a contradiction with the minimality of $F$. □

We are ready to give a kernel for Feedback Arc Set in Tournaments.

**Theorem 2.8.** Feedback Arc Set in Tournaments *admits a kernel with at most $k^2 + 2k$ vertices.*

*Proof.* Lemma 2.7 implies that a tournament $T$ has a feedback arc set of size at most $k$ if and only if it can be turned into an acyclic tournament by reversing directions of at most $k$ edges. We will use this characterization for the kernel.

In what follows by a *triangle* we mean a directed cycle of length three. We give two simple reduction rules.

**Reduction FAST.1.** If an edge $e$ is contained in at least $k+1$ triangles, then reverse $e$ and reduce $k$ by 1.

**Reduction FAST.2.** If a vertex $v$ is not contained in any triangle, then delete $v$ from $T$.

The rules follow similar guidelines as in the case of Vertex Cover. In Reduction FAST.1, we greedily take into a solution an edge that participates in $k+1$ otherwise disjoint forbidden structures (here, triangles). In Reduction FAST.2, we discard vertices that do not participate in any forbidden structure, and should be irrelevant to the problem.

However, a formal proof of the safeness of Reduction FAST.2 is not immediate: we need to verify that deleting $v$ and its incident edges does not make make a yes-instance out of a no-instance.

Note that after applying any of the two rules, the resulting graph is again a tournament. The first rule is safe because if we do not reverse $e$, we have to reverse at least one edge from each of $k+1$ triangles containing $e$. Thus $e$ belongs to every feedback arc set of size at most $k$.

Let us now prove the safeness of the second rule. Let $X = N^+(v)$ be the set of heads of directed edges with tail $v$ and let $Y = N^-(v)$ be the set of tails of directed edges with head $v$. Because $T$ is a tournament, $X$ and $Y$ is a partition of $V(T) \setminus \{v\}$. Since $v$ is not a part of any triangle in $T$, we have that there is no edge from $X$ to $Y$ (with head in $Y$ and tail in $X$). Consequently, for any feedback arc set $A_1$ of tournament $T[X]$ and any feedback arc set $A_2$ of tournament $T[Y]$, the set $A_1 \cup A_2$ is a feedback arc set of $T$. As the reverse implication is trivial (for any feedback arc set $A$ in $T$, $A \cap E(T[X])$ is a feedback arc set of $T[X]$, and $A \cap E(T[Y])$ is a feedback arc set of $T[Y]$), we have that $(T, k)$ is a yes-instance if and only if $(T - v, k)$ is.

Finally, we show that every reduced yes-instance $T$, an instance on which none of the presented reduction rules are applicable, has at most $k(k+2)$ vertices. Let $A$ be a feedback arc set of a reduced instance $T$ of size at most $k$. For every edge $e \in A$, aside from the two endpoints of $e$, there are at most $k$ vertices that are in triangles containing $e$ — otherwise we would be able to apply Reduction FAST.1. Since every triangle in $T$ contains an edge of $A$ and every vertex of $T$ is in a triangle, we have that $T$ has at most $k(k+2)$ vertices.

Thus, given $(T, k)$ we apply our reduction rules exhaustively and obtain an equivalent instance $(T', k')$. If $T'$ has more than $k'^2 + k'$ vertices, then the algorithm returns that $(T, k)$ is a no-instance, otherwise we get the desired kernel. This completes the proof of the theorem. ☐

### *2.2.3* Edge Clique Cover

Not all FPT problems admit polynomial kernels. In the Edge Clique Cover problem, we are given a graph $G$ and a nonnegative integer $k$, and the goal is to decide whether the edges of $G$ can be covered by at most $k$ cliques. In this section we give an exponential kernel for Edge Clique Cover. In Theorem 14.20 of Section *14.3.3, we remark that this simple kernel is essentially optimal.

Let us recall the reader that we use $N(v) = \{u : uv \in E(G)\}$ to denote the neighborhood of vertex $v$ in $G$, and $N[v] = N(v) \cup \{v\}$ to denote the closed neighborhood of $v$. We apply the following data reduction rules in the given order (i.e., we always use the lowest-numbered rule that modifies the instance).

**Reduction ECC.1.** Remove isolated vertices.

**Reduction ECC.2.** If there is an isolated edge $uv$ (a connected component that is just an edge), delete it and decrease $k$ by 1. The new instance is $(G - \{u, v\}, k - 1)$.

**Reduction ECC.3.** If there is an edge $uv$ whose endpoints have exactly the same closed neighborhood, that is, $N[u] = N[v]$, then delete $v$. The new instance is $(G - v, k)$.

The crux of the presented kernel for Edge Clique Cover is an observation that two true twins (vertices $u$ and $v$ with $N[u] = N[v]$) can be treated in exactly the same way in some optimum solution, and hence we can reduce them. Meanwhile, the vertices that are contained in exactly the same set of cliques in a feasible solution *have to* be true twins. This observation bounds the size of the kernel.

The safeness of the first two reductions is trivial, while the safeness of Reduction ECC.3 follows from the observation that a solution in $G - v$ can be extended to a solution in $G$ by adding $v$ to all the cliques containing $u$ (see Exercise 2.3).

**Theorem 2.9.** Edge Clique Cover *admits a kernel with at most* $2^k$ *vertices.*

*Proof.* We start with the following claim.

*Claim.* If $(G, k)$ is a reduced yes-instance, on which none of the presented reduction rules can be applied, then $|V(G)| \leq 2^k$.

*Proof.* Let $C_1, \ldots, C_k$ be an edge clique cover of $G$. We claim that $G$ has at most $2^k$ vertices. Targeting a contradiction, let us assume that $G$ has more than $2^k$ vertices. We assign to each vertex $v \in V(G)$ a binary vector $b_v$ of length $k$, where bit $i$, $1 \leq i \leq k$, is set to 1 if and only if $v$ is contained in clique $C_i$. Since there are only $2^k$ possible vectors, there must be $u \neq v \in V(G)$ with $b_u = b_v$. If $b_u$ and $b_v$ are zero vectors, the first rule applies; otherwise, $u$ and $v$ are contained in the same cliques. This means that $u$ and $v$ are adjacent and have the same neighborhood; thus either Reduction ECC.2 or Reduction ECC.3 applies. Hence, if $G$ has more than $2^k$ vertices, at least one of the reduction rules can be applied to it, which is a contradiction to the initial assumption that $G$ is reduced. This completes the proof of the claim. □

The kernelization algorithm works as follows. Given an instance $(G, k)$, it applies Reductions ECC.1, ECC.2, and ECC.3 exhaustively. If the resulting graph has more than $2^k$ vertices the kernelization algorithm outputs that the input instance is a no-instance, else it outputs the reduced instance. □

## 2.3 Crown decomposition

Crown decomposition is a general kernelization technique that can be used to obtain kernels for many problems. The technique is based on the classical matching theorems of Kőnig and Hall.

Recall that for disjoint vertex subsets $U, W$ of a graph $G$, a matching $M$ is called *a matching of $U$ into $W$* if every edge of $M$ connects a vertex of $U$ and a vertex of $W$ and, moreover, every vertex of $U$ is an endpoint of some edge of $M$. In this situation, we also say that $M$ *saturates* $U$.

**Definition 2.10 (Crown decomposition).** A *crown decomposition* of a graph $G$ is a partitioning of $V(G)$ into three parts $C$, $H$ and $R$, such that

1. $C$ is nonempty.
2. $C$ is an independent set.
3. There are no edges between vertices of $C$ and $R$. That is, $H$ separates $C$ and $R$.
4. Let $E'$ be the set of edges between vertices of $C$ and $H$. Then $E'$ contains a matching of size $|H|$. In other words, $G$ contains a matching of $H$ into $C$.

The set $C$ can be seen as a crown put on head $H$ of the remaining part $R$, see Fig. 2.1. Note that the fact that $E'$ contains a matching of size $|H|$ implies that there is a matching of $H$ into $C$. This is a matching in the subgraph $G'$, with the vertex set $C \cup H$ and the edge set $E'$, saturating all the vertices of $H$.

For finding a crown decomposition in polynomial time, we use the following well known structural and algorithmic results. The first is a mini-max theorem due to Kőnig.

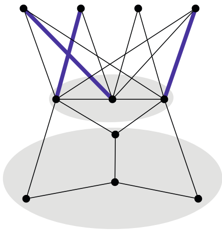


Fig. 2.1: Example of a crown decomposition. Set $C$ is an independent set, $H$ separates $C$ and $R$, and there is a matching of $H$ into $C$

**Theorem 2.11 (Kőnig's theorem, [303]).** *In every undirected bipartite graph the size of a maximum matching is equal to the size of a minimum vertex cover.*

Let us recall that a matching *saturates* a set of vertices $S$ when every vertex in $S$ is incident to an edge in the matching. The second classic result states that in bipartite graphs, a trivial necessary condition for the existence of a matching is also sufficient.

**Theorem 2.12 (Hall's theorem, [256]).** *Let $G$ be an undirected bipartite graph with bipartition $(V_1, V_2)$. The graph $G$ has a matching saturating $V_1$ if and only if for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$.*

The following theorem is due to Hopcroft and Karp [268]. The proof of the (nonstandard) second claim of the theorem is deferred to Exercise 2.21.

**Theorem 2.13 (Hopcroft-Karp algorithm, [268]).** *Let $G$ be an undirected bipartite graph with bipartition $V_1$ and $V_2$, on $n$ vertices and $m$ edges. Then we can find a maximum matching as well as a minimum vertex cover of $G$ in time $\mathcal{O}(m\sqrt{n})$. Furthermore, in time $\mathcal{O}(m\sqrt{n})$ either we can find a matching saturating $V_1$ or an inclusion-wise minimal set $X \subseteq V_1$ such that $|N(X)| < |X|$.*

The following lemma is the basis for kernelization algorithms using crown decomposition.

**Lemma 2.14 (Crown lemma).** *Let $G$ be a graph without isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that either*

- *finds a matching of size $k + 1$ in $G$; or*
- *finds a crown decomposition of $G$.*

*Proof.* We first find an inclusion-maximal matching $M$ in $G$. This can be done by a greedy algorithm. If the size of $M$ is $k + 1$, then we are done.

Hence, we assume that $|M| \leq k$, and let $V_M$ be the endpoints of $M$. We have $|V_M| \leq 2k$. Because $M$ is a maximal matching, the remaining set of vertices $I = V(G) \setminus V_M$ is an independent set.

Consider the bipartite graph $G_{I,V_M}$ formed by edges of $G$ between $V_M$ and $I$. We compute a minimum-sized vertex cover $X$ and a maximum sized matching $M'$ of the bipartite graph $G_{I,V_M}$ in polynomial time using Theorem 2.13. We can assume that $|M'| \leq k$, for otherwise we are done. Since $|X| = |M'|$ by Kőnig's theorem (Theorem 2.11), we infer that $|X| \leq k$.

If no vertex of $X$ is in $V_M$, then $X \subseteq I$. We claim that $X = I$. For a contradiction assume that there is a vertex $w \in I \setminus X$. Because $G$ has no isolated vertices there is an edge, say $wz$, incident to $w$ in $G_{I,V_M}$. Since $G_{I,V_M}$ is bipartite, we have that $z \in V_M$. However, $X$ is a vertex cover of $G_{I,V_M}$ such that $X \cap V_M = \emptyset$, which implies that $w \in X$. This is contrary to our assumption that $w \notin X$, thus proving that $X = I$. But then $|I| \leq |X| \leq k$, and $G$ has at most

$$|I| + |V_M| \leq k + 2k = 3k$$

vertices, which is a contradiction.

Hence, $X \cap V_M \neq \emptyset$. We obtain a crown decomposition $(C, H, R)$ as follows. Since $|X| = |M'|$, every edge of the matching $M'$ has exactly one endpoint in $X$. Let $M^*$ denote the subset of $M'$ such that every edge from $M^*$ has exactly one endpoint in $X \cap V_M$ and let $V_{M^*}$ denote the set of endpoints of edges in $M^*$. We define head $H = X \cap V_M = X \cap V_{M^*}$, crown $C = V_{M^*} \cap I$, and the remaining part $R = V(G) \setminus (C \cup H) = V(G) \setminus V_{M^*}$. In other words, $H$ is the set of endpoints of edges of $M^*$ that are present in $V_M$ and $C$ is the set of endpoints of edges of $M^*$ that are present in $I$. Obviously, $C$ is an independent set and by construction, $M^*$ is a matching of $H$ into $C$. Furthermore, since $X$ is a vertex cover of $G_{I,V_M}$, every vertex of $C$ can be adjacent only to vertices of $H$ and thus $H$ separates $C$ and $R$. This completes the proof. □

The crown lemma gives a relatively strong structural property of graphs with a small vertex cover (equivalently, a small maximum matching). If in a studied problem the parameter upper bounds the size of a vertex cover (maximum matching), then there is a big chance that the structural insight given by the crown lemma would help in developing a small kernel — quite often with number of vertices bounded linearly in the parameter.

We demonstrate the application of crown decompositions on kernelization for Vertex Cover and Maximum Satisfiability.

### *2.3.1* Vertex Cover

Consider a Vertex Cover instance $(G, k)$. By an exhaustive application of Reduction VC.1, we may assume that $G$ has no isolated vertices. If $|V(G)| > 3k$, we may apply the crown lemma to the graph $G$ and integer $k$, obtaining either a matching of size $k+1$, or a crown decomposition $V(G) = C \cup H \cup R$. In the first case, the algorithm concludes that $(G, k)$ is a no-instance.

In the latter case, let $M$ be a matching of $H$ into $C$. Observe that the matching $M$ witnesses that, for every vertex cover $X$ of the graph $G$, $X$ contains at least $|M| = |H|$ vertices of $H \cup C$ to cover the edges of $M$. On the other hand, the set $H$ covers all edges of $G$ that are incident to $H \cup C$. Consequently, there exists a minimum vertex cover of $G$ that contains $H$, and we may reduce $(G, k)$ to $(G-H, k-|H|)$. Note that in the instance $(G-H, k-|H|)$, the vertices of $C$ are isolated and will be reduced by Reduction VC.1.

As the crown lemma promises us that $H \neq \emptyset$, we can always reduce the graph as long as $|V(G)| > 3k$. Thus, we obtain the following.

**Theorem 2.15.** Vertex Cover *admits a kernel with at most* $3k$ *vertices.*

### *2.3.2* Maximum Satisfiability

For a second application of the crown decomposition, we look at the following parameterized version of Maximum Satisfiability. Given a CNF formula $F$, and a nonnegative integer $k$, decide whether $F$ has a truth assignment satisfying at least $k$ clauses.

**Theorem 2.16.** Maximum Satisfiability *admits a kernel with at most* $k$ *variables and* $2k$ *clauses.*

*Proof.* Let $\varphi$ be a CNF formula with $n$ variables and $m$ clauses. Let $\psi$ be an arbitrary assignment to the variables and let $\neg\psi$ be the assignment obtained by complementing the assignment of $\psi$. That is, if $\psi$ assigns $\delta \in \{\top, \bot\}$ to some variable $x$ then $\neg\psi$ assigns $\neg\delta$ to $x$. Observe that either $\psi$ or $\neg\psi$ satisfies at least $m/2$ clauses, since every clause is satisfied by $\psi$ or $\neg\psi$ (or by both). This means that, if $m \geq 2k$, then $(\varphi, k)$ is a yes-instance. In what follows we give a kernel with $n < k$ variables.

Let $G_\varphi$ be the *variable-clause* incidence graph of $\varphi$. That is, $G_\varphi$ is a bipartite graph with bipartition $(X, Y)$, where $X$ is the set of the variables of $\varphi$ and $Y$ is the set of clauses of $\varphi$. In $G_\varphi$ there is an edge between a variable $x \in X$ and a clause $c \in Y$ if and only if either $x$, or its negation, is in $c$. If there is a matching of $X$ into $Y$ in $G_\varphi$, then there is a truth assignment satisfying at least $|X|$ clauses: we can set each variable in $X$ in such a way that the clause matched to it becomes satisfied. Thus at least $|X|$ clauses are satisfied. Hence, in this case, if $k \leq |X|$, then $(\varphi, k)$ is a yes-instance. Otherwise,

$k > |X| = n$, and we get the desired kernel. We now show that, if $\varphi$ has at least $n \geq k$ variables, then we can, in polynomial time, either reduce $\varphi$ to an equivalent smaller instance, or find an assignment to the variables satisfying at least $k$ clauses (and conclude that we are dealing with a yes-instance).

Suppose $\varphi$ has at least $k$ variables. Using Hall's theorem and a polynomial-time algorithm computing a maximum-size matching (Theorems 2.12 and 2.13), we can in polynomial time find either a matching of $X$ into $Y$ or an inclusion-wise minimal set $C \subseteq X$ such that $|N(C)| < |C|$. As discussed in the previous paragraph, if we found a matching, then the instance is a yes-instance and we are done. So suppose we found a set $C$ as described. Let $H$ be $N(C)$ and $R = V(G_\varphi) \setminus (C \cup H)$. Clearly, $N(C) \subseteq H$, there are no edges between vertices of $C$ and $R$ and $G[C]$ is an independent set. Select an arbitrary $x \in C$. We have that there is a matching of $C \setminus \{x\}$ into $H$ since $|N(C')| \geq |C'|$ for every $C' \subseteq C \setminus \{x\}$. Since $|C| > |H|$, we have that the matching from $C \setminus \{x\}$ to $H$ is in fact a matching of $H$ into $C$. Hence $(C, H, R)$ is a crown decomposition of $G_\varphi$.

We prove that all clauses in $H$ are satisfied in every assignment satisfying the maximum number of clauses. Indeed, consider any assignment $\psi$ that does not satisfy all clauses in $H$. Fix any variable $x \in C$. For every variable $y$ in $C \setminus \{x\}$ set the value of $y$ so that the clause in $H$ matched to $y$ is satisfied. Let $\psi'$ be the new assignment obtained from $\psi$ in this manner. Since $N(C) \subseteq H$ and $\psi'$ satisfies all clauses in $H$, more clauses are satisfied by $\psi'$ than by $\psi$. Hence $\psi$ cannot be an assignment satisfying the maximum number of clauses.

The argument above shows that $(\varphi, k)$ is a yes-instance to MAXIMUM SATISFIABILITY if and only if $(\varphi \setminus H, k - |H|)$ is. This gives rise to the following simple reduction.

**Reduction MSat.1.** Let $(\varphi, k)$ and $H$ be as above. Then remove $H$ from $\varphi$ and decrease $k$ by $|H|$. That is, $(\varphi \setminus H, k - |H|)$ is the new instance.

Repeated applications of Reduction MSat.1 and the arguments described above give the desired kernel. This completes the proof of the theorem. □

## 2.4 Expansion lemma

In the previous subsection, we described crown decomposition techniques based on the classical Hall's theorem. In this section, we introduce a powerful variation of Hall's theorem, which is called the expansion lemma. This lemma captures a certain property of neighborhood sets in graphs and can be used to obtain polynomial kernels for many graph problems. We apply this result to get an $\mathcal{O}(k^2)$ kernel for FEEDBACK VERTEX SET in Chapter 9.

A *q-star*, $q \geq 1$, is a graph with $q+1$ vertices, one vertex of degree $q$, called the *center*, and all other vertices of degree 1 adjacent to the center. Let $G$ be a bipartite graph with vertex bipartition $(A, B)$. For a positive integer $q$, a set of edges $M \subseteq E(G)$ is called by a *$q$-expansion of $A$ into $B$* if

- every vertex of $A$ is incident to exactly $q$ edges of $M$;
- $M$ saturates exactly $q|A|$ vertices in $B$.

Let us emphasize that a $q$-expansion saturates all vertices of $A$. Also, for every $u, v \in A$, $u \neq v$, the set of vertices $E_u$ adjacent to $u$ by edges of $M$ does not intersect the set of vertices $E_v$ adjacent to $v$ via edges of $M$, see Fig. 2.2. Thus every vertex $v \in A$ could be thought of as the center of a star with its $q$ leaves in $B$, with all these $|A|$ stars being vertex-disjoint. Furthermore, a collection of these stars is also a family of $q$ edge-disjoint matchings, each saturating $A$.

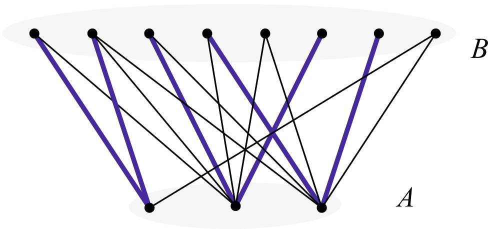


Fig. 2.2: Set $A$ has a 2-expansion into $B$

Let us recall that, by Hall's theorem (Theorem 2.12), a bipartite graph with bipartition $(A, B)$ has a matching of $A$ into $B$ if and only if $|N(X)| \geq |X|$ for all $X \subseteq A$. The following lemma is an extension of this result.

**Lemma 2.17.** *Let $G$ be a bipartite graph with bipartition $(A, B)$. Then there is a $q$-expansion from $A$ into $B$ if and only if $|N(X)| \geq q|X|$ for every $X \subseteq A$. Furthermore, if there is no $q$-expansion from $A$ into $B$, then a set $X \subseteq A$ with $|N(X)| < q|X|$ can be found in polynomial time.*

*Proof.* If $A$ has a $q$-expansion into $B$, then trivially $|N(X)| \geq q|X|$ for every $X \subseteq A$.

For the opposite direction, we construct a new bipartite graph $G'$ with bipartition $(A', B)$ from $G$ by adding $(q - 1)$ copies of all the vertices in $A$. For every vertex $v \in A$ all copies of $v$ have the same neighborhood in $B$ as $v$. We would like to prove that there is a matching $M$ from $A'$ into $B$ in $G'$. If we prove this, then by identifying the endpoints of $M$ corresponding to the copies of vertices from $A$, we obtain a $q$-expansion in $G$. It suffices to check that the assumptions of Hall's theorem are satisfied in $G'$. Assume otherwise, that there is a set $X \subseteq A'$ such that $|N_{G'}(X)| < |X|$. Without loss of generality, we can assume that if $X$ contains some copy of a vertex $v$, then it contains all the copies of $v$, since including all the remaining copies increases $|X|$ but does not change $|N_{G'}(X)|$. Hence, the set $X$ in $A'$ naturally corresponds to the set $X_A$ of size $|X|/q$ in $A$, the set of vertices whose copies are in $X$. But then $|N_G(X_A)| = |N_{G'}(X)| < |X| = q|X_A|$, which is a contradiction. Hence $A'$ has a matching into $B$ and thus $A$ has a $q$-expansion into $B$.

For the algorithmic claim, note that, if there is no $q$-expansion from $A$ into $B$, then we can use Theorem 2.13 to find a set $X \subseteq A'$ such that $|N_{G'}(X)| < |X|$, and the corresponding set $X_A$ satisfies $|N_G(X_A)| < q|X_A|$. □

Finally, we are ready to prove a lemma analogous to Lemma 2.14.

**Lemma 2.18. (Expansion lemma)** *Let $q \geq 1$ be a positive integer and $G$ be a bipartite graph with vertex bipartition $(A, B)$ such that*

(*i*) *$|B| \geq q|A|$, and*
(*ii*) *there are no isolated vertices in $B$.*

*Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that*

- *there is a $q$-expansion of $X$ into $Y$, and*
- *no vertex in $Y$ has a neighbor outside $X$, that is, $N(Y) \subseteq X$.*

*Furthermore, the sets $X$ and $Y$ can be found in time polynomial in the size of $G$.*

Note that the sets $X$, $Y$ and $V(G) \setminus (X \cup Y)$ form a crown decomposition of $G$ with a stronger property — every vertex of $X$ is not only matched into $Y$, but there is a $q$-expansion of $X$ into $Y$. We proceed with the proof of expansion lemma.

*Proof.* We proceed recursively, at every step decreasing the cardinality of $A$. When $|A| = 1$, the claim holds trivially by taking $X = A$ and $Y = B$.

We apply Lemma 2.17 to $G$. If $A$ has a $q$-expansion into $B$, then we are done as we may again take $X = A$ and $Y = B$. Otherwise, we can in polynomial time find a (nonempty) set $Z \subseteq A$ such that $|N(Z)| < q|Z|$. We construct the graph $G'$ by removing $Z$ and $N(Z)$ from $G$. We claim that $G'$ satisfies the assumptions of the lemma. Indeed, because we removed less than $q$ times more vertices from $B$ than from $A$, we have that (*i*) holds for $G'$. Moreover, every vertex from $B \setminus N(Z)$ has no neighbor in $Z$, and thus (*ii*) also holds for $G'$. Note that $Z \neq A$, because otherwise $N(A) = B$ (there are no isolated vertices in $B$) and $|B| \geq q|A|$. Hence, we recurse on the graph $G'$ with bipartition $(A \setminus Z, B \setminus N(Z))$, obtaining nonempty sets $X \subseteq A \setminus Z$ and $Y \subseteq B \setminus N(Z)$ such that there is a $q$-expansion of $X$ into $Y$ and such that $N_{G'}(Y) \subseteq X$. Because $Y \subseteq B \setminus N(Z)$, we have that no vertex in $Y$ has a neighbor in $Z$. Hence, $N_{G'}(Y) = N_G(Y) \subseteq X$ and the pair $(X, Y)$ satisfies all the required properties. □

The expansion lemma is useful when the matching saturating the head part $H$ in the crown lemma turns out to be not sufficient for a reduction, and we would like to have a few vertices of the crown $C$ matched to a single vertex of the head $H$. For example, this is the case for the FEEDBACK VERTEX SET kernel presented in Section 9.1, where we need the case $q = 2$.

## 2.5 Kernels based on linear programming

In this section we design a $2k$-vertex kernel for VERTEX COVER exploiting the solution to a linear programming formulation of VERTEX COVER.

Many combinatorial problems can be expressed in the language of INTEGER LINEAR PROGRAMMING (ILP). In an INTEGER LINEAR PROGRAMMING instance, we are given a set of integer-valued variables, a set of linear inequalities (called *constraints*) and a linear *cost function*. The goal is to find an (integer) evaluation of the variables that satisfies all constraints, and minimizes or maximizes the value of the cost function.

Let us give an example on how to encode a VERTEX COVER instance $(G, k)$ as an INTEGER LINEAR PROGRAMMING instance. We introduce $n = |V(G)|$ variables, one variable $x_v$ for each vertex $v \in V(G)$. Setting variable $x_v$ to 1 means that $v$ is in the vertex cover, while setting $x_v$ to 0 means that $v$ is not in the vertex cover. To ensure that every edge is covered, we can introduce constraints $x_u + x_v \geq 1$ for every edge $uv \in E(G)$. The size of the vertex cover is given by $\sum_{v \in V(G)} x_v$. In the end, we obtain the following ILP formulation:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V(G)} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every } uv \in E(G), \\ & 0 \leq x_v \leq 1 & \text{for every } v \in V(G), \\ & x_v \in \mathbb{Z} & \text{for every } v \in V(G). \end{array} \tag{2.1}$$

Clearly, the optimal value of (2.1) is at most $k$ if and only if $G$ has a vertex cover of size at most $k$.

As we have just seen, INTEGER LINEAR PROGRAMMING is at least as hard as VERTEX COVER, so we do not expect it to be polynomial-time solvable. In fact, it is relatively easy to express many NP-hard problems in the language of INTEGER LINEAR PROGRAMMING. In Section 6.2 we discuss FPT algorithms for INTEGER LINEAR PROGRAMMING and their application in proving fixed-parameter tractability of other problems.

Here, we proceed in a different way: we relax the integrality requirement of INTEGER LINEAR PROGRAMMING, which is the main source of the hardness of this problem, to obtain LINEAR PROGRAMMING. That is, in LINEAR

Programming the instance looks exactly the same as in Integer Linear Programming, but the variables are allowed to take arbitrary real values, instead of just integers.

In the case of Vertex Cover, we relax (2.1) by dropping the constraint $x_v \in \mathbb{Z}$ for every $v \in V(G)$. In other words, we obtain the following Linear Programming instance. For a graph $G$, we call this relaxation LPVC($G$).

$$\begin{array}{lll} \text{minimize} & \sum_{v\in V(G)} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every } uv \in E(G), \\ & 0 \leq x_v \leq 1 & \text{for every } v \in V(G). \end{array} \tag{2.2}$$

Note that constraints $x_v \leq 1$ can be omitted because every optimal solution of LPVC($G$) satisfies these constraints.

Observe that in LPVC($G$), a variable $x_v$ can take fractional values in the interval $[0, 1]$, which corresponds to taking "part of the vertex $v$" into a vertex cover. Consider an example of $G$ being a triangle. A minimum vertex cover of a triangle is of size 2, whereas in LPVC($G$) we can take $x_v = \frac{1}{2}$ for every $v \in V(G)$, obtaining a feasible solution of cost $\frac{3}{2}$. Thus, LPVC($G$) does not express exactly the Vertex Cover problem on graph $G$, but its optimum solution can still be useful to learn something about minimum vertex covers in $G$.

The main source of utility of Linear Programming comes from the fact that Linear Programming can be solved in polynomial time, even in some general cases where there are exponentially many constraints, accessed through an oracle. For this reason, Linear Programming has found abundant applications in approximation algorithms (for more on this topic, we refer to the book of Vazirani [427]). In this section, we use LP to design a small kernel for Vertex Cover. In Section 3.4, we will use LPVC($G$) to obtain an FPT branching algorithm for Vertex Cover.

Let us now have a closer look at the relaxation LPVC($G$). Fix an optimal solution $(x_v)_{v\in V(G)}$ of LPVC($G$). In this solution the variables corresponding to vertices of $G$ take values in the interval $[0, 1]$. We partition $V(G)$ according to these values into three sets as follows.

- $V_0 = \{v \in V(G) \ : \ x_v < \frac{1}{2}\}$,
- $V_{\frac{1}{2}} = \{v \in V(G) \ : \ x_v = \frac{1}{2}\}$,
- $V_1 = \{v \in V(G) \ : \ x_v > \frac{1}{2}\}$.

**Theorem 2.19 (Nemhauser-Trotter theorem).** *There is a minimum vertex cover $S$ of $G$ such that*

$$V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}.$$

*Proof.* Let $S^* \subseteq V(G)$ be a minimum vertex cover of $G$. Define

$$S = (S^* \setminus V_0) \cup V_1.$$

By the constraints of (2.2), every vertex of $V_0$ can have a neighbor only in $V_1$ and thus $S$ is also a vertex cover of $G$. Moreover, $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$. It suffices to show that $S$ is a *minimum* vertex cover. Assume the contrary, i.e., $|S| > |S^*|$. Since $|S| = |S^*| - |V_0 \cap S^*| + |V_1 \setminus S^*|$ we infer that

$$|V_0 \cap S^*| < |V_1 \setminus S^*|. \tag{2.3}$$

Let us define

$$\varepsilon = \min\{|x_v - \tfrac{1}{2}| \ : \ v \in V_0 \cup V_1\}.$$

We decrease the fractional values of vertices from $V_1 \setminus S^*$ by $\varepsilon$ and increase the values of vertices from $V_0 \cap S^*$ by $\varepsilon$. In other words, we define a vector $(y_v)_{v \in V(G)}$ as

$$y_v = \begin{cases} x_v - \varepsilon & \text{if } v \in V_1 \setminus S^*, \\ x_v + \varepsilon & \text{if } v \in V_0 \cap S^*, \\ x_v & \text{otherwise.} \end{cases}$$

Note that $\varepsilon > 0$, because otherwise $V_0 = V_1 = \emptyset$, a contradiction with (2.3). This, together with (2.3), implies that

$$\sum_{v \in V(G)} y_v < \sum_{v \in V(G)} x_v. \tag{2.4}$$

Now we show that $(y_v)_{v \in V(G)}$ is a feasible solution, i.e., it satisfies the constraints of LPVC$(G)$. Since $(x_v)_{v \in V(G)}$ is a feasible solution, by the definition of $\varepsilon$ we get $0 \leq y_v \leq 1$ for every $v \in V(G)$. Consider an arbitrary edge $uv \in E(G)$. If none of the endpoints of $uv$ belong to $V_1 \setminus S^*$, then both $y_u \geq x_u$ and $y_v \geq x_v$, so $y_u + y_v \geq x_u + x_v \geq 1$. Otherwise, by symmetry we can assume that $u \in V_1 \setminus S^*$, and hence $y_u = x_u - \varepsilon$. Because $S^*$ is a vertex cover, we have that $v \in S^*$. If $v \in V_0 \cap S^*$, then

$$y_u + y_v = x_u - \varepsilon + x_v + \varepsilon = x_u + x_v \geq 1.$$

Otherwise, $v \in (V_{\frac{1}{2}} \cup V_1) \cap S^*$. Then $y_v \geq x_v \geq \frac{1}{2}$. Note also that $x_u - \varepsilon \geq \frac{1}{2}$ by the definition of $\varepsilon$. It follows that

$$y_u + y_v = x_u - \varepsilon + y_v \geq \frac{1}{2} + \frac{1}{2} = 1.$$

Thus $(y_v)_{v \in V(G)}$ is a feasible solution of LPVC$(G)$ and hence (2.4) contradicts the optimality of $(x_v)_{v \in V(G)}$. □

Theorem 2.19 allows us to use the following reduction rule.

**Reduction VC.4.** Let $(x_v)_{v \in V(G)}$ be an optimum solution to LPVC$(G)$ in a Vertex Cover instance $(G, k)$ and let $V_0$, $V_1$ and $V_{\frac{1}{2}}$ be defined as above. If $\sum_{v \in V(G)} x_v > k$, then conclude that we are dealing with a no-instance. Otherwise, greedily take into the vertex cover the vertices of $V_1$. That is, delete all vertices of $V_0 \cup V_1$, and decrease $k$ by $|V_1|$.

Let us now formally verify the safeness of Reduction VC.4.

**Lemma 2.20.** *Reduction VC.4 is safe.*

*Proof.* Clearly, if $(G, k)$ is a yes-instance, then an optimum solution to LPVC$(G)$ is of cost at most $k$. This proves the correctness of the step if we conclude that $(G, k)$ is a no-instance.

Let $G' = G - (V_0 \cup V_1) = G[V_{\frac{1}{2}}]$ and $k' = k - |V_1|$. We claim that $(G, k)$ is a yes-instance of Vertex Cover if and only if $(G', k')$ is. By Theorem 2.19, we know that $G$ has a vertex cover $S$ of size at most $k$ such that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$. Then $S' = S \cap V_{\frac{1}{2}}$ is a vertex cover in $G'$ and the size of $S'$ is at most $k - |V_1| = k'$.

For the opposite direction, let $S'$ be a vertex cover in $G'$. For every solution of LPVC$(G)$, every edge with an endpoint from $V_0$ should have an endpoint in $V_1$. Hence, $S = S' \cup V_1$ is a vertex cover in $G$ and the size of this vertex cover is at most $k' + |V_1| = k$. □

Reduction VC.4 leads to the following kernel for Vertex Cover.

**Theorem 2.21.** Vertex Cover *admits a kernel with at most* $2k$ *vertices.*

*Proof.* Let $(G, k)$ be an instance of Vertex Cover. We solve LPVC$(G)$ in polynomial time, and apply Reduction VC.4 to the obtained solution $(x_v)_{v \in V(G)}$, either concluding that we are dealing with a no-instance or obtaining an instance $(G', k')$. Lemma 2.20 guarantees the safeness of the reduction. For the size bound, observe that

$$|V(G')| = |V_{\frac{1}{2}}| = \sum_{v \in V_{\frac{1}{2}}} 2x_v \leq 2 \sum_{v \in V(G)} x_v \leq 2k.$$

□

While it is possible to solve linear programs in polynomial time, usually such solutions are less efficient than combinatorial algorithms. The specific structure of the LP-relaxation of the vertex cover problem (2.2) allows us to solve it by reducing to the problem of finding a maximum-size matching in a bipartite graph.

**Lemma 2.22.** *For a graph $G$ with $n$ vertices and $m$ edges, the optimal (fractional) solution to the linear program* LPVC$(G)$ *can be found in time* $\mathcal{O}(m\sqrt{n})$.

*Proof.* We reduce the problem of solving LPVC$(G)$ to a problem of finding a minimum-size vertex cover in the following bipartite graph $H$. Its vertex set consists of two copies $V_1$ and $V_2$ of the vertex set of $G$. Thus, every vertex $v \in V(G)$ has two copies $v_1 \in V_1$ and $v_2 \in V_2$ in $H$. For every edge $uv \in E(H)$, we have edges $u_1v_2$ and $v_1u_2$ in $H$.

Using the Hopcroft-Karp algorithm (Theorem 2.13), we can find a minimum vertex cover $S$ of $H$ in time $\mathcal{O}(m\sqrt{n})$. We define a vector $(x_v)_{v\in V(G)}$ as follows: if both vertices $v_1$ and $v_2$ are in $S$, then $x_v = 1$. If exactly one of the vertices $v_1$ and $v_2$ is in $S$, we put $x_v = \frac{1}{2}$. We put $x_v = 0$ if none of the vertices $v_1$ and $v_2$ are in $S$. Thus

$$\sum_{v\in V(G)} x_v = \frac{|S|}{2}.$$

Since $S$ is a vertex cover in $H$, we have that for every edge $uv \in E(G)$ at least two vertices from $\{u_1, u_2, v_1, v_2\}$ should be in $S$. Thus $x_u + x_v \geq 1$ and vector $(x_v)_{v\in V(G)}$ satisfies the constraints of LPVC($G$).

To show that $(x_v)_{v\in V(G)}$ is an optimal solution of LPVC($G$), we argue as follows. Let $(y_v)_{v\in V(G)}$ be an optimal solution of LPVC($G$). For every vertex $v_i$, $i \in \{1,2\}$, of $H$, we assign the weight $\mathbf{w}(v_i) = y_v$. This weight assignment is a fractional vertex cover of $H$, i.e., for every edge $v_1u_2 \in E(H)$, $\mathbf{w}(v_1) + \mathbf{w}(u_2) \geq 1$. We have that

$$\sum_{v\in V(G)} y_v = \frac{1}{2} \sum_{v\in V(G)} (\mathbf{w}(v_1) + \mathbf{w}(v_2)).$$

On the other hand, the value $\sum_{v\in V(H)} \mathbf{w}(v)$ of any fractional solution of LPVC($H$) is at least the size of a maximum matching $M$ in $H$. A reader familiar with linear programming can see that this follows from weak duality; we also ask you to verify this fact in Exercise 2.24.

By Kőnig's theorem (Theorem 2.11), $|M| = |S|$. Hence

$$\sum_{v\in V(G)} y_v = \frac{1}{2} \sum_{v\in V(G)} (\mathbf{w}(v_1) + \mathbf{w}(v_2)) = \frac{1}{2} \sum_{v\in V(H)} \mathbf{w}(v) \geq \frac{|S|}{2} = \sum_{v\in V(G)} x_v.$$

Thus $(x_v)_{v\in V(G)}$ is an optimal solution of LPVC($G$). $\square$

We immediately obtain the following.

**Corollary 2.23.** *For a graph $G$ with $n$ vertices and $m$ edges, the kernel of Theorem 2.21 can be found in time $\mathcal{O}(m\sqrt{n})$.*

The following proposition is another interesting consequence of the proof of Lemma 2.22.

**Proposition 2.24.** *Let $G$ be a graph on $n$ vertices and $m$ edges. Then* LPVC($G$) *has a half-integral optimal solution, i.e., all variables have values in the set $\{0, \frac{1}{2}, 1\}$. Furthermore, we can find a half-integral optimal solution in time $\mathcal{O}(m\sqrt{n})$.*

In short, we have proved properties of LPVC($G$). There exists a half-integral optimal solution $(x_v)_{v\in V(G)}$ to LPVC($G$), and it can be found efficiently. We can look at this solution as a partition of $V(G)$ into parts $V_0$, $V_{\frac{1}{2}}$, and $V_1$ with the following message: greedily take $V_1$ into a solution, do not take any vertex of $V_0$ into a solution, and in $V_{\frac{1}{2}}$, we do not know what to do and that is the hard part of the problem. However, as an optimum solution pays $\frac{1}{2}$ for every vertex of $V_{\frac{1}{2}}$, the hard part — the kernel of the problem — cannot have more than $2k$ vertices.

## 2.6 Sunflower lemma

In this section we introduce a classical result of Erdős and Rado and show some of its applications in kernelization. In the literature it is known as the sunflower lemma or as the Erdős-Rado lemma. We first define the terminology used in the statement of the lemma. A *sunflower* with $k$ *petals* and a *core* $Y$ is a collection of sets $S_1, \dots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are petals and we require *none of them to be empty.* Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

**Theorem 2.25 (Sunflower lemma).** *Let $\mathcal{A}$ be a family of sets (without duplicates) over a universe $U$, such that each set in $\mathcal{A}$ has cardinality exactly $d$. If $|\mathcal{A}| > d!(k-1)^d$, then $\mathcal{A}$ contains a sunflower with $k$ petals and such a sunflower can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and $k$.*

*Proof.* We prove the theorem by induction on $d$. For $d = 1$, i.e., for a family of singletons, the statement trivially holds. Let $d \geq 2$ and let $\mathcal{A}$ be a family of sets of cardinality at most $d$ over a universe $U$ such that $|\mathcal{A}| > d!(k-1)^d$.

Let $\mathcal{G} = \{S_1, \dots, S_\ell\} \subseteq \mathcal{A}$ be an inclusion-wise maximal family of pairwise disjoint sets in $\mathcal{A}$. If $\ell \geq k$ then $\mathcal{G}$ is a sunflower with at least $k$ petals. Thus we assume that $\ell < k$. Let $S = \bigcup_{i=1}^{\ell} S_i$. Then $|S| \leq d(k-1)$. Because $\mathcal{G}$ is maximal, every set $A \in \mathcal{A}$ intersects at least one set from $\mathcal{G}$, i.e., $A \cap S \neq \emptyset$. Therefore, there is an element $u \in U$ contained in at least

$$\frac{|\mathcal{A}|}{|S|} > \frac{d!(k-1)^d}{d(k-1)} = (d-1)!(k-1)^{d-1}$$

sets from $\mathcal{A}$. We take all sets of $\mathcal{A}$ containing such an element $u$, and construct a family $\mathcal{A}'$ of sets of cardinality $d-1$ by removing from each set the element $u$. Because $|\mathcal{A}'| > (d-1)!(k-1)^{d-1}$, by the induction hypothesis, $\mathcal{A}'$ contains a sunflower $\{S'_1, \dots, S'_k\}$ with $k$ petals. Then $\{S'_1 \cup \{u\}, \dots, S'_k \cup \{u\}\}$ is a sunflower in $\mathcal{A}$ with $k$ petals.

The proof can be easily transformed into a polynomial-time algorithm, as follows. Greedily select a maximal set of pairwise disjoint sets. If the size of this set is at least $k$, then return this set. Otherwise, find an element $u$ contained in the maximum number of sets in $\mathcal{A}$, and call the algorithm recursively on sets of cardinality $d-1$, obtained from deleting $u$ from the sets containing $u$. □

### 2.6.1 $d$-Hitting Set

As an application of the sunflower lemma, we give a kernel for $d$-Hitting Set. In this problem, we are given a family $\mathcal{A}$ of sets over a universe $U$, where each set in the family has cardinality at most $d$, and a positive integer $k$. The objective is to decide whether there is a subset $H \subseteq U$ of size at most $k$ such that $H$ contains at least one element from each set in $\mathcal{A}$.

**Theorem 2.26.** $d$-Hitting Set *admits a kernel with at most* $d!k^d$ *sets and at most* $d!k^d \cdot d^2$ *elements.*

*Proof.* The crucial observation is that if $\mathcal{A}$ contains a sunflower

$$S = \{S_1, \dots, S_{k+1}\}$$

of cardinality $k+1$, then every hitting set $H$ of $\mathcal{A}$ of cardinality at most $k$ intersects the core $Y$ of the sunflower $S$. Indeed, if $H$ does not intersect $Y$, it should intersect each of the $k+1$ disjoint petals $S_i \setminus Y$. This leads to the following reduction rule.

**Reduction HS.1.** Let $(U, \mathcal{A}, k)$ be an instance of $d$-Hitting Set and assume that $\mathcal{A}$ contains a sunflower $S = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core $Y$. Then return $(U', \mathcal{A}', k)$, where $\mathcal{A}' = (\mathcal{A} \setminus S) \cup \{Y\}$ is obtained from $\mathcal{A}$ by deleting all sets $\{S_1, \dots, S_{k+1}\}$ and by adding a new set $Y$ and $U' = \bigcup_{X \in \mathcal{A}'} X$.

Note that when deleting sets we do not delete the elements contained in these sets but only those which do not belong to any set. Then the instances $(U, \mathcal{A}, k)$ and $(U', \mathcal{A}', k)$ are equivalent, i.e. $(U, \mathcal{A})$ contains a hitting set of size $k$ if and only if $(U, \mathcal{A}')$ does.

The kernelization algorithm is as follows. If for some $d' \in \{1, \dots, d\}$ the number of sets in $\mathcal{A}$ of size exactly $d'$ is more than $d'!k^{d'}$, then the kernelization algorithm applies the sunflower lemma to find a sunflower of size $k+1$, and applies Reduction HS.1 on this sunflower. It applies this procedure exhaustively, and obtains a new family of sets $\mathcal{A}'$ of size at most $d!k^d \cdot d$. If $\emptyset \in \mathcal{A}'$ (that is, at some point a sunflower with an empty core has been discovered), then the algorithm concludes that there is no hitting set of size at most $k$ and returns that the given instance is a no-instance. Otherwise, every set contains at most $d$ elements, and thus the number of elements in the kernel is at most $d!k^d \cdot d^2$. □

## Exercises

**2.1 (🍃).** Prove Lemma 2.5: A digraph is acyclic if and only if it is possible to order its vertices in such a way such that for every arc $(u, v)$, we have $u < v$.

**2.2 (🍃).** Give an example of a feedback arc set $F$ in a tournament $G$, such that $G \circledast F$ is not acyclic.

**2.3 (🍃).** Show that Reductions ECC.1, ECC.2, and ECC.3 are safe.

**2.4 (🍃).** In the Point Line Cover problem, we are given a set of $n$ points in the plane and an integer $k$, and the goal is to check if there exists a set of $k$ lines on the plane that contain all the input points. Show a kernel for this problem with $\mathcal{O}(k^2)$ points.

**2.5.** A graph $G$ is a *cluster graph* if every connected component of $G$ is a clique. In the Cluster Editing problem, we are given as input a graph $G$ and an integer $k$, and the objective is to check whether one can edit (add or delete) at most $k$ edges in $G$ to obtain a cluster graph. That is, we look for a set $F \subseteq \binom{V(G)}{2}$ of size at most $k$, such that the graph $(V(G), (E(G) \setminus F) \cup (F \setminus E(G)))$ is a cluster graph.

1. Show that a graph $G$ is a cluster graph if and only if it does not have an induced path on three vertices (sequence of three vertices $u, v, w$ such that $uv$ and $vw$ are edges and $uw \notin E(G)$).
2. Show a kernel for Cluster Editing with $\mathcal{O}(k^2)$ vertices.

**2.6.** In the Set Splitting problem, we are given a family of sets $\mathcal{F}$ over a universe $U$ and a positive integer $k$, and the goal is to test whether there exists a coloring of $U$ with two colors such that at least $k$ sets in $\mathcal{F}$ are nonmonochromatic (that is, they contain vertices of both colors). Show that the problem admits a kernel with at most $2k$ sets and $\mathcal{O}(k^2)$ universe size.

**2.7.** In the Minimum Maximal Matching problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to decide whether there exists a maximal matching in $G$ on at most $k$ edges. Obtain a polynomial kernel for the problem (parameterized by $k$).

**2.8.** In the Min-Ones-2-SAT problem, we are given a 2-CNF formula $\phi$ and an integer $k$, and the objective is to decide whether there exists a satisfying assignment for $\phi$ with at most $k$ variables set to true. Show that Min-Ones-2-SAT admits a polynomial kernel.

**2.9.** In the $d$-Bounded-Degree Deletion problem, we are given an undirected graph $G$ and a positive integer $k$, and the task is to find at most $k$ vertices whose removal decreases the maximum vertex degree of the graph to at most $d$. Obtain a kernel of size polynomial in $k$ and $d$ for the problem. (Observe that Vertex Cover is the case of $d = 0$.)

**2.10.** Show a kernel with $\mathcal{O}(k^2)$ vertices for the following problem: given a graph $G$ and an integer $k$, check if $G$ contains a subgraph with exactly $k$ edges, whose vertices are all of odd degree in the subgraph.

**2.11.** A set of vertices $D$ in an undirected graph $G$ is called a *dominating set* if $N[D] = V(G)$. In the Dominating Set problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to test whether there exists a dominating set of size at most $k$. Show that Dominating Set admits a polynomial kernel on graphs where the length of the shortest cycle is at least 5. (What would you do with vertices with degree more than $k$? Note that unlike for the Vertex Cover problem, you cannot delete a vertex once you pick it in the solution.)

**2.12.** Show that Feedback Vertex Set admits a kernel with $\mathcal{O}(k)$ vertices on undirected regular graphs.

**2.13.** We say that an $n$-vertex digraph is *well-spread* if every vertex has indegree at least $\sqrt{n}$. Show that Directed Feedback Arc Set, restricted to well-spread digraphs, is FPT by obtaining a polynomial kernel for this problem. Does the problem remain FPT if we replace the lower bound on indegree by any monotonically increasing function of $n$ (like $\log n$ or $\log\log\log n$)? Does the assertion hold if we replace indegree with outdegree? What about Directed Feedback Vertex Set?

**2.14.** In the Connected Vertex Cover problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to decide whether there exists a vertex cover $C$ of $G$ such that $|C| \leq k$ and $G[C]$ is connected.

1. Explain where the kernelization procedure described in Theorem 2.4 for Vertex Cover breaks down for the Connected Vertex Cover problem.
2. Show that the problem admits a kernel with at most $2^k + \mathcal{O}(k^2)$ vertices.
3. Show that if the input graph $G$ does not contain a cycle of length 4 as a subgraph, then the problem admits a kernel with at most $\mathcal{O}(k^2)$ vertices.

**2.15 (☠).** Extend the argument of the previous exercise to show that, for every fixed $d \geq 2$, Connected Vertex Cover admits a kernel of size $\mathcal{O}(k^d)$ if restricted to graphs that do not contain the biclique $K_{d,d}$ as a subgraph.

**2.16 (☠).** A graph $G$ is chordal if it contains no induced cycles of length more than 3, that is, every cycle of length at least 4 has a chord. In the Chordal Completion problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to decide whether we can add at most $k$ edges to $G$ so that it becomes a chordal graph. Obtain a polynomial kernel for Chordal Completion (parameterized by $k$).

**2.17 (☠).** In the Edge Disjoint Cycle Packing problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to test whether $G$ has $k$ pairwise edge disjoint cycles. Obtain a polynomial kernel for Edge Disjoint Cycle Packing (parameterized by $k$).

**2.18 (☠).** A *bisection* of a graph $G$ with an even number of vertices is a partition of $V(G)$ into $V_1$ and $V_2$ such that $|V_1| = |V_2|$. The size of $(V_1, V_2)$ is the number of edges with one endpoint in $V_1$ and the other in $V_2$. In the Maximum Bisection problem, we are given an undirected graph $G$ with an even number of vertices and a positive integer $k$, and the objective is to test whether there exists a bisection of size at least $k$.

1. Show that every graph with $m$ edges has a bisection of size at least $\lceil \frac{m}{2} \rceil$. Use this to show that Maximum Bisection admits a kernel with $2k$ edges.
2. Consider the following "above guarantee" variant of Maximum Bisection, where we are given an undirected graph $G$ and a positive integer $k$, but the objective is to test whether there exists a bisection of size at least $\lceil \frac{m}{2} \rceil + k$. Show that the problem admits a kernel with $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^3)$ edges.

**2.19 (✎).** Byteland, a country of area exactly $n$ square miles, has been divided by the government into $n$ regions, each of area exactly one square mile. Meanwhile, the army of Byteland divided its area into $n$ military zones, each of area again exactly one square mile. Show that one can build $n$ airports in Byteland, such that each region and each military zone contains one airport.

**2.20.** A magician and his assistant are performing the following magic trick. A volunteer from the audience picks five cards from a standard deck of 52 cards and then passes the deck to the assistant. The assistant shows to the magician, one by one in some order, four cards from the chosen set of five cards. Then, the magician guesses the remaining fifth card. Show that this magic trick can be performed without any help of magic.

**2.21.** Prove the second claim of Theorem 2.13.

**2.22.** In the DUAL-COLORING problem, we are given an undirected graph $G$ on $n$ vertices and a positive integer $k$, and the objective is to test whether there exists a proper coloring of $G$ with at most $n - k$ colors. Obtain a kernel with $\mathcal{O}(k)$ vertices for this problem using crown decomposition.

**2.23 (☠).** In the MAX-INTERNAL SPANNING TREE problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to test whether there exists a spanning tree with at least $k$ internal vertices. Obtain a kernel with $\mathcal{O}(k)$ vertices for MAX-INTERNAL SPANNING TREE.

**2.24 (✐).** Let $G$ be an undirected graph, let $(x_v)_{v \in V(G)}$ be any feasible solution to LPVC($G$), and let $M$ be a matching in $G$. Prove that $|M| \leq \sum_{v \in V(G)} x_v$.

**2.25 (☠).** Let $G$ be a graph and let $(x_v)_{v \in V(G)}$ be an optimum solution to LPVC($G$) (not necessarily a half-integral one). Define a vector $(y_v)_{v \in V(G)}$ as follows:

$$y_v = \begin{cases} 0 & \text{if } x_v < \frac{1}{2} \\ \frac{1}{2} & \text{if } x_v = \frac{1}{2} \\ 1 & \text{if } x_v > \frac{1}{2}. \end{cases}$$

Show that $(y_v)_{v \in V(G)}$ is also an optimum solution to LPVC($G$).

**2.26 (☠).** In the MIN-ONES-2-SAT, we are given a CNF formula, where every clause has exactly two literals, and an integer $k$, and the goal is to check if there exists a satisfying assignment of the input formula with at most $k$ variables set to true. Show a kernel for this problem with at most $2k$ variables.

**2.27 (✐).** Consider a restriction of $d$-HITTING SET, called E$d$-HITTING SET, where we require every set in the input family $\mathcal{A}$ to be of size *exactly* $d$. Show that this problem is not easier than the original $d$-HITTING SET problem, by showing how to transform a $d$-HITTING SET instance into an equivalent E$d$-HITTING SET instance without changing the number of sets.

**2.28.** Show a kernel with at most $f(d)k^d$ sets (for some computable function $f$) for the E$d$-HITTING SET problem, defined in the previous exercise.

**2.29.** In the $d$-SET PACKING problem, we are given a family $\mathcal{A}$ of sets over a universe $U$, where each set in the family has cardinality at most $d$, and a positive integer $k$. The objective is to decide whether there are sets $S_1, \ldots, S_k \in \mathcal{A}$ that are pairwise disjoint. Use the sunflower lemma to obtain a kernel for $d$-SET PACKING with $f(d)k^d$ sets, for some computable function $d$.

**2.30.** Consider a restriction of $d$-SET PACKING, called E$d$-SET PACKING, where we require every set in the input family $\mathcal{A}$ to be of size *exactly* $d$. Show that this problem is not easier than the original $d$-SET PACKING problem, by showing how to transform a $d$-SET PACKING instance into an equivalent E$d$-SET PACKING instance without changing the number of sets.

**2.31.** A *split graph* is a graph in which the vertices can be partitioned into a clique and an independent set. In the VERTEX DISJOINT PATHS problem, we are given an undirected graph $G$ and $k$ pairs of vertices $(s_i, t_i)$, $i \in \{1, \ldots, k\}$, and the objective is to decide whether there exists paths $P_i$ joining $s_i$ to $t_i$ such that these paths are pairwise vertex disjoint. Show that VERTEX DISJOINT PATHS admits a polynomial kernel on split graphs (when parameterized by $k$).

**2.32.** Consider now the VERTEX DISJOINT PATHS problem, defined in the previous exercise, restricted, for a fixed integer $d \geq 3$, to a class of graphs that does not contain a $d$-vertex path as an induced subgraph. Show that in this class the VERTEX DISJOINT PATHS problem admits a kernel with $\mathcal{O}(k^{d-1})$ vertices and edges.

**2.33.** In the CLUSTER VERTEX DELETION problem, we are given as input a graph $G$ and a positive integer $k$, and the objective is to check whether there exists a set $S \subseteq V(G)$ of size at most $k$ such that $G - S$ is a cluster graph. Show a kernel for CLUSTER VERTEX DELETION with $\mathcal{O}(k^3)$ vertices.

**2.34.** An undirected graph $G$ is called *perfect* if for every induced subgraph $H$ of $G$, the size of the largest clique in $H$ is same as the chromatic number of $H$. In the ODD CYCLE TRANSVERSAL problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to find at most $k$ vertices whose removal makes the resulting graph bipartite. Obtain a kernel with $\mathcal{O}(k^2)$ vertices for ODD CYCLE TRANSVERSAL on perfect graphs.

**2.35.** In the SPLIT VERTEX DELETION problem, we are given an undirected graph $G$ and a positive integer $k$ and the objective is to test whether there exists a set $S \subseteq V(G)$ of size at most $k$ such that $G - S$ is a split graph (see Exercise 2.31 for the definition).

1. Show that a graph is split if and only if it has no induced subgraph isomorphic to one of the following three graphs: a cycle on four or five vertices, or a pair of disjoint edges.
2. Give a kernel with $\mathcal{O}(k^5)$ vertices for SPLIT VERTEX DELETION.

**2.36 (☠).** In the SPLIT EDGE DELETION problem, we are given an undirected graph $G$ and a positive integer $k$, and the objective is to test whether $G$ can be transformed into a split graph by deleting at most $k$ edges. Obtain a polynomial kernel for this problem (parameterized by $k$).

**2.37 (☠).** In the RAMSEY problem, we are given as input a graph $G$ and an integer $k$, and the objective is to test whether there exists in $G$ an independent set or a clique of size at least $k$. Show that RAMSEY is FPT.

**2.38 (☠).** A directed graph $D$ is called *oriented* if there is no directed cycle of length at most 2. Show that the problem of testing whether an oriented digraph contains an induced directed acyclic subgraph on at least $k$ vertices is FPT.

# Hints

**2.4** Consider the following reduction rule: if there exists a line that contains more than $k$ input points, delete the points on this line and decrease $k$ by 1.

**2.5** Consider the following natural reduction rules:

1. delete a vertex that is not a part of any $P_3$ (induced path on three vertices);
2. if an edge $uv$ is contained in at least $k+1$ different $P_3$s, then delete $uv$;
3. if a non-edge $uv$ is contained in at least $k+1$ different $P_3$s, then add $uv$.

Show that, after exhaustive application of these rules, a yes-instance has $\mathcal{O}(k^2)$ vertices.

**2.6** First, observe that one can discard any set in $\mathcal{F}$ that is of size at most 1. Second, observe that if every set in $\mathcal{F}$ is of size at least 2, then a random coloring of $U$ has at least

$|\mathcal{F}|/2$ nonmonochromatic sets on average, and an instance with $|\mathcal{F}| \geq 2k$ is a yes-instance. Moreover, observe that if we are dealing with a yes-instance and $F \in \mathcal{F}$ is of size at least $2k$, then we can always tweak the solution coloring to color $F$ nonmonochromatically: fix two differently colored vertices for $k-1$ nonmonochromatic sets in the solution, and color some two uncolored vertices of $F$ with different colors. Use this observation to design a reduction rule that handles large sets in $\mathcal{F}$.

**2.7** Observe that the endpoints of the matching in question form a vertex cover of the input graph. In particular, every vertex of degree larger than $2k$ needs to be an endpoint of a solution matching. Let $X$ be the set of these large-degree vertices. Argue, similarly as in the case of $\mathcal{O}(k^2)$ kernel for Vertex Cover, that in a yes-instance, $G \setminus X$ has only few edges. Design a reduction rule to reduce the number of isolated vertices of $G \setminus X$.

**2.8** Proceed similarly as in the $\mathcal{O}(k^2)$ kernel for Vertex Cover.

**2.9** Proceed similarly as in the case of Vertex Cover. Argue that the vertices of degree larger than $d+k$ need to be included in the solution. Moreover, observe that you may delete isolated vertices, as well as edges connecting two vertices of degree at most $d$. Argue that, if no rule is applicable, then a yes-instance is of size bounded polynomially in $d+k$.

**2.10** The important observation is that a matching of size $k$ is a good subgraph. Hence, we may restrict ourselves to the case where we are additionally given a vertex cover $X$ of the input graph of size at most $2k$. Moreover, assume that $X$ is inclusion-wise minimal. To conclude, prove that, if a vertex $v \in X$ has at least $k$ neighbors in $V(G) \setminus X$, then $(G,k)$ is a yes-instance.

**2.11** The main observation is that, since there is no 3-cycle nor 4-cycle in the graph, if $x, y \in N(v)$, then only $v$ can dominate both $x$ and $y$ at once. In particular, every vertex of degree larger than $k$ needs to be included in the solution.

However, you cannot easily delete such a vertex. Instead, mark it as "obligatory" and mark its neighbors as "dominated". Note now that you can delete a "dominated" vertex, as long as it has no unmarked neighbor and its deletion does not drop the degree of an "obligatory" vertex to $k$.

Prove that, in a yes-instance, if no rule is applicable, then the size is bounded polynomially in $k$. To this end, show that

1. any vertex can dominate at most $k$ unmarked vertices, and, consequently, there are at most $k^2$ unmarked vertices;
2. there are at most $k$ "obligatory" vertices;
3. every remaining "dominated" vertex can be charged to one unmarked or obligatory vertex in a manner that each unmarked or obligatory vertex is charged at most $k+1$ times.

**2.12** Let $(G,k)$ be a Feedback Vertex Set instance and assume $G$ is $d$-regular. If $d \leq 2$, then solve $(G,k)$ in polynomial time. Otherwise, observe that $G$ has $dn/2$ edges and, if $(G,k)$ is a yes-instance and $X$ is a feedback vertex set of $G$ of size at most $k$, then at most $dk$ edges of $G$ are incident to $X$ and $G-X$ contains less than $n-k$ edges (since it is a forest). Consequently, $dn/2 \leq dk + n - k$, which gives $n = \mathcal{O}(k)$ for $d \geq 3$.

**2.13** Show, using greedy arguments, that if every vertex in a digraph $G$ has indegree at least $d$, then $G$ contains $d$ pairwise edge-disjoint cycles.

For the vertex-deletion variant, design a simple reduction that boosts up the indegree of every vertex without actually changing anything in the solution space.

**2.14** Let $X$ be the set of vertices of $G$ of degree larger than $k$. Clearly, any connected vertex cover of $G$ of size at most $k$ needs to contain $X$. Moreover, as in the case of Vertex Cover, in a yes-instance there are only $\mathcal{O}(k^2)$ edges in $G-X$. However, we cannot easily discard the isolated vertices of $G-X$, as they may be used to make the solution connected.

To obtain an exponential kernel, note that in a yes-instance, $|X| \leq k$, and if we have two vertices $u, v$ that are isolated in $G - X$, and $N_G(u) = N_G(v)$ (note that $N_G(u) \subseteq X$ for every $u$ that is isolated in $G - X$), then we need only one of the vertices $u, v$ in a Connected Vertex Cover solution. Hence, in a kernel, we need to keep:

1. $G[X]$, and all edges and non-isolated vertices of $G - X$;
2. for every $x \in X$, some $k + 1$ neighbors of $x$;
3. for every $Y \subseteq X$, one vertex $u$ that is isolated in $G - X$ and $N_G(u) = Y$ (if there exists any such vertex).

For the last part of the exercise, note that in the presence of this assumption, no two vertices of $X$ share more than one neighbor and, consequently, there are only $\mathcal{O}(|X|^2)$ sets $Y \subseteq X$ for which there exist $u \notin X$ with $N_G(u) = Y$.

**2.15** We repeat the argument of the previous exercise, and bound the number of sets $Y \subseteq X$ for which we need to keep a vertex $u \in V(G) \setminus X$ with $N_G(u) = Y$. First, there are $\mathcal{O}(d|X|^{d-1})$ sets $Y$ of size smaller than $d$. Second, charge every set $Y$ of size at least $d$ to one of its subset of size $d$. Since $G$ does not contain $K_{d,d}$ as a subgraph, every subset $X$ of size $d$ is charged less than $d$ times. Consequently, there are at most $(d-1)\binom{|X|}{d}$ vertices $u \in V(G) \setminus X$ such that $N_G(u) \subseteq X$ and $|N_G(u)| \geq d$.

**2.16** The main observation is as follows: an induced cycle of length $\ell$ needs exactly $\ell - 3$ edges to become chordal. In particular, if a graph contains an induced cycle of length larger than $k + 3$, then the input instance is a no-instance, as we need more than $k$ edges to triangulate the cycle in question.

First, prove the safeness of the following two reduction rules:

1. Delete any vertex that is not contained in any induced cycle in $G$.
2. A vertex $x$ is a *friend* of a non-edge $uv$, if $u, x, v$ are three consecutive vertices of some induced cycle in $G$. If $uv \notin E(G)$ has more than $2k$ friends, then add the edge $uv$ and decrease $k$ by one.

Second, consider the following procedure. Initiate $A$ to be the vertex set of any inclusion-wise maximal family of pairwise vertex-disjoint induced cycles of length at most 4 in $G$. Then, as long as there exists an induced cycle of length at most 4 in $G$ that contains two consecutive vertices in $V(G) \setminus A$, move these two vertices to $A$. Show, using a charging argument, that, in a yes-instance, the size of $A$ remains $\mathcal{O}(k)$. Conclude that the size of a reduced yes-instance is bounded polynomially in $k$.

**2.17** Design reduction rules that remove vertices of degree at most 2 (you may obtain a multigraph in the process). Prove that every $n$-vertex multigraph of minimum degree at least 3 has a cycle of length $\mathcal{O}(\log n)$. Use this to show a greedy argument that an $n$-vertex multigraph of minimum degree 3 has $\Omega(n^{\varepsilon})$ pairwise edge-disjoint cycles for some $\varepsilon > 0$.

**2.18** Consider the following argument. Let $|V(G)| = 2n$ and pair the vertices of $G$ arbitrarily: $V(G) = \{x_1, y_1, x_2, y_2, \ldots, x_n, y_n\}$. Consider the bisection $(V_1, V_2)$ where, in each pair $(x_i, y_i)$, one vertex goes to $V_1$ and the other goes to $V_2$, where the decision is made uniformly at random and independently of other pairs. Prove that, in expectation, the obtained bisection is of size at least $(m + \ell)/2$, where $\ell$ is the number of pairs $(x_i, y_i)$ where $x_i y_i \in E(G)$.

Use the arguments in the previous paragraph to show not only the first point of the exercise, but also that the input instance is a yes-instance if it admits a matching of size $2k$. If this is not the case, then let $X$ be the set of endpoints of a maximal matching in $G$; note that $|X| \leq 4k$.

First, using a variation of the argument of the first paragraph, prove that, if there exists $x \in X$ that has at least $2k$ neighbors and at least $2k$ non-neighbors outside $X$, then the input instance is a yes-instance. Second, show that in the absence of such a vertex, all but

$\mathcal{O}(k^2)$ vertices of $V(G) \setminus X$ have exactly the same neighborhood in $X$, and design a way to reduce them.

**2.19** Construct the following bipartite graph: on one side there are regions of Byteland, on the second side there are military zones, and a region $R$ is adjacent to a zone $Z$ if $R \cap Z \neq \emptyset$. Show that this graph satisfies the condition of Hall's theorem and, consequently, contains a perfect matching.

**2.20** Consider the following bipartite graph: on one side there are all $\binom{52}{5}$ sets of five cards (possibly chosen by the volunteer), and on the other side there are all $52 \cdot 51 \cdot 50 \cdot 49$ tuples of pairwise different four cards (possibly shown by the assistant). A set $S$ is adjacent to a tuple $T$ if all cards of $T$ belong to $S$. Using Hall's theorem, show that this graph admits a matching saturating the side with all sets of five cards. This matching induces a strategy for the assistant and the magician.

We now show a relatively simple explicit strategy, so that you can impress your friends and perform this trick at some party. In every set of five cards, there are two cards of the same color, say $a$ and $b$. Moreover, as there are 13 cards of the same color, the cards $a$ and $b$ differ by at most 6, that is, $a+i=b$ or $b+i=a$ for some $1 \leq i \leq 6$, assuming some cyclic order on the cards of the same color. Without loss of generality, assume $a+i=b$. The assistant first shows the card $a$ to the magician. Then, using the remaining three cards, and some fixed total order on the whole deck of cards, the assistant shows the integer $i$ (there are $3! = 6$ permutations of remaining three cards). Consequently, the magician knows the card $b$ by knowing its color (the same as the first card show by the assistant) and the value of the card $a$ and the number $i$.

**2.21** Let $M$ be a maximum matching, which you can find using the Hopcroft-Karp algorithm (the first part of Theorem 2.13). If $M$ saturates $V_1$, then we are done. Otherwise, pick any $v \in V_1 \setminus V(M)$ (i.e., a vertex $v \in V_1$ that is not an endpoint of an edge of $M$) and consider all vertices of $G$ that are reachable from $v$ using alternating paths. (A path $P$ is *alternating* if every second edge of $P$ belongs to $M$.) Show that all vertices from $V_1$ that are reachable from $v$ using alternating paths form an inclusion-wise minimal set $X$ with $|N(X)| < |X|$.

**2.22** Apply the crown lemma to $\bar{G}$, *the edge complement of* $G$ ($\bar{G}$ has vertex set $V(G)$ and $uv \in E(\bar{G})$ if and only if $uv \notin E(G)$) and the parameter $k-1$. If it returns a matching $M_0$ of size $k$, then note that one can color the endpoints of each edge of $M_0$ with the same color, obtaining a coloring of $G$ with $n-k$ colors. Otherwise, design a way to greedily color the head and the crown of the obtained crown decomposition.

**2.23** Your main tool is the following variation of the crown lemma: if $V(G)$ is sufficiently large, then you can find either a matching of size $k+1$, or a crown decomposition $V(G) = C \cup H \cup R$, such that $G[H \cup C]$ admits a spanning tree where all vertices of $H$ and $|H|-1$ vertices of $C$ are of degree at least two. Prove it, and use it for the problem in question.

**2.24** Observe that for every $uv \in M$ we have $x_u + x_v \geq 1$ and, moreover, all these inequalities for all edges of $M$ contain different variables. In other words,

$$\sum_{v \in V(G)} \mathbf{w}(x_v) \geq \sum_{v \in V(M)} \mathbf{w}(x_v) = \sum_{vu \in M} (\mathbf{w}(x_v) + \mathbf{w}(x_u)) \geq \sum_{vu \in M} 1 = |M|.$$

**2.25** Let $V_\delta = \{v \in V(G) : 0 < x_v < \frac{1}{2}\}$ and $V_{1-\delta} = \{v \in V(G) : \frac{1}{2} < x_v < 1\}$. For sufficiently small $\varepsilon > 0$, consider two operations: first, an operation of adding $\varepsilon$ to all variables $x_v$ for $v \in V_\delta$ and subtracting $\varepsilon$ from $x_v$ for $v \in V_{1-\delta}$, and second, an operation of adding $\varepsilon$ to all variables $x_v$ for $v \in V_{1-\delta}$ and subtracting $\varepsilon$ from $x_v$ for $v \in V_\delta$. Show that both these operations lead to feasible solutions to LPVC($G$), as long as $\varepsilon$ is small enough. Conclude that $|V_\delta| = |V_{1-\delta}|$, and that both operations lead to other *optimal* solutions to LPVC($G$). Finally, observe that, by repeatedly applying the second operation, one can empty sets $V_\delta$ and $V_{1-\delta}$, and reach the vector $(y_v)_{v\in V(G)}$.

**2.26** First, design natural reduction rules that enforce the following: for every variable $x$ in the input formula $\varphi$, there exists a truth assignment satisfying $\varphi$ that sets $x$ to false, and a truth assignment satisfying $\varphi$ that sets $x$ to true. In other words, whenever a value of some variable is fixed in any satisfying assignment, fix this value and propagate it in the formula.

Then, consider the following *closure* of the formula $\varphi$: for every two-literal clause $C$ that is satisfied in every truth assignment satisfying $\varphi$, add $C$ to $\varphi$. Note that testing whether $C$ is such a clause can be done in polynomial time: force two literals in $C$ to be false and check if $\varphi$ remains satisfiable. Moreover, observe that the sets of satisfying assignments for $\varphi$ and $\varphi'$ are equal.

Let $\varphi'$ be the closure of $\varphi$. Consider the following auxiliary graph $H$: $V(H)$ is the set of variables of $\varphi'$, and $xy \in E(H)$ iff the clause $x \vee y$ belongs to $\varphi'$. Clearly, if we take any truth assignment $\psi$ satisfying $\varphi$, then $\psi^{-1}(\top)$ is a vertex cover of $H$. A somewhat surprising fact is that a partial converse is true: for every inclusion-wise minimal vertex cover $X$ of $H$, the assignment $\psi$ defined as $\psi(x) = \top$ if and only if $x \in X$ satisfies $\varphi'$ (equivalently, $\varphi$). Note that such a claim would solve the exercise: we can apply the LP-based kernelization algorithm to Vertex Cover instance $(H, k)$, and translate the reductions it makes back to the formula $\varphi$.

Below we prove the aforementioned claim in full detail. We encourage you to try to prove it on your own before reading.

Let $X$ be a minimal vertex cover of $H$, and let $\psi$ be defined as above. Take any clause $C$ in $\varphi'$ and consider three cases. If $C = x \vee y$, then $xy \in E(H)$, and, consequently, either $x$ or $y$ belongs to $X$. It follows from the definition of $\psi$ that $\psi(x) = \top$ or $\psi(y) = \top$, and $\psi$ satisfies $C$.

In the second case, $C = x \vee \neg y$. For a contradiction, assume that $\psi$ does not satisfy $C$ and, consequently, $x \notin X$ and $y \in X$. Since $X$ is a minimal vertex cover, there exists $z \in N_H(y)$ such that $z \notin X$ and the clause $C' = y \vee z$ belongs to $\varphi'$. If $z = x$, then any satisfying assignment to $\varphi'$ sets $y$ to true, a contradiction to our first preprocessing step. Otherwise, the presence of $C$ and $C'$ implies that in any assignment $\psi'$ satisfying $\varphi'$ we have $\psi'(x) = \top$ or $\psi'(z) = \top$. Thus, $x \vee z$ is a clause of $\varphi'$, and $xz \in E(H)$. However, neither $x$ nor $z$ belongs to $X$, a contradiction.

In the last case, $C = \neg x \vee \neg y$ and, again, we assume that $\psi$ does not satisfy $C$, that is, $x, y \in X$. Since $X$ is a minimal vertex cover, there exist $s \in N_H(x)$, $t \in N_H(y)$ such that $s, t \notin X$. It follows from the definition of $H$ that the clauses $C_x = x \vee s$ and $C_y = y \vee t$ are present in $\varphi'$. If $s = t$, then the clauses $C$, $C_x$ and $C_y$ imply that $t$ is set to true in any truth assignment satisfying $\varphi'$, a contradiction to our first preprocessing step. If $s \neq t$, then observe that the clauses $C$, $C_x$ and $C_y$ imply that either $s$ or $t$ is set to true in any truth assignment satisfying $\varphi'$ and, consequently, $s \vee t$ is a clause of $\varphi'$ and $st \in E(H)$. However, $s, t \notin X$, a contradiction.

**2.27** If $\emptyset \in \mathcal{A}$, then conclude that we are dealing with a no-instance. Otherwise, for every set $X \in \mathcal{A}$ of size $|X| < d$, create $d - |X|$ new elements and add them to $X$.

**2.28** There are two ways different ways to solve this exercise. First, you can treat the input instance as a $d$-Hitting Set instance, proceed as in Section 2.6.1, and at the end apply the solution of Exercise 2.27 to the obtained kernel, in order to get an E$d$-Hitting Set instance.

In a second approach, try to find a sunflower with $k+2$ sets, instead of $k+1$ as in Section 2.6.1. If a sunflower is found, then discard one of the sets: the remaining $k+1$ sets still ensure that the core needs to be hit in any solution of size at most $k$.

**2.29** Show that in a $(dk+2)$-sunflower, one set can be discarded without changing the answer to the problem.

**2.30** Proceed as in Exercise 2.27: pad every set $X \in \mathcal{A}$ with $d-|X|$ newly created elements.

**2.31** Show that, in a split graph, every path can be shortened to a path on at most four vertices. Thus, for every $1 \leq i \leq k$, we have a family $\mathcal{F}_i$ of vertex sets of possible paths between $s_i$ and $t_i$, and this family is of size $\mathcal{O}(n^4)$. Interpret the problem as a $d$-Set Packing instance for some constant $d$ and family $\mathcal{F} = \bigcup_{i=1}^{k} \mathcal{F}_i$. Run the kernelization algorithm from the previous exercise, and discard all vertices that are not contained in any set in the obtained kernel.

**2.32** Show that, in a graph excluding a $d$-vertex path as an induced subgraph, every path in a solution can shortened to a path on at most $d-1$ vertices. Proceed then as in Exercise 2.31.

**2.33** Let $\mathcal{A}$ be a family of all vertex sets of a $P_3$ (induced path on three vertices) in $G$. In this manner, Cluster Vertex Deletion becomes a 3-Hitting Set problem on family $\mathcal{A}$, as we need to hit all induced $P_3$s in $G$. Reduce $\mathcal{A}$, but not exactly as in the $d$-Hitting Set case: repeatedly find a $k+2$ sunflower and delete one of its elements from $\mathcal{A}$. Show that this reduction is safe for Cluster Vertex Deletion. Moreover, show that, if $\mathcal{A}'$ is the family after the reduction is exhaustively applied, then $(G[\bigcup \mathcal{A}'], k)$ is the desired kernel.

**2.34** Use the following observation: a perfect graph is bipartite if and only if it does not contain a triangle. Thus, the problem reduces to hitting all triangles in the input graph, which is a 3-Hitting Set instance.

**2.35** Proceed as in the case of Cluster Vertex Deletion: interpret a Split Vertex Deletion instance as a 5-Hitting Set instance.

**2.36** Let $\{C_4, C_5, 2K_2\}$ be the set of *forbidden induced subgraphs* for split graphs. That is, a graph is a split graph if it contains none of these three graphs as an induced subgraph.

You may need (some of) the following reduction rules. (Note that the safeness of some of them is not so easy.)

1. The "standard" sunflower-like: if more than $k$ forbidden induced subgraphs share a single edge (and otherwise are pairwise edge-disjoint), delete the edge in question.
2. The irrelevant vertex rule: if a vertex is not part of any forbidden induced subgraph, then delete the vertex in question. (Safeness is not obvious here!)
3. If two adjacent edges $uv$ and $uw$ are contained in more than $k$ induced $C_4$s, then delete $uv$ and $uw$, and replace them with edges $va$ and $wb$, where $a$ and $b$ are new degree-1 vertices.
4. If two adjacent edges $uv$ and $uw$ are contained in more than $k$ pairwise edge-disjoint induced $C_5$s, then delete $uv$ and $uw$, and decrease $k$ by 2.
5. If the edges $v_1v_2$, $v_2v_3$ and $v_3v_4$ are contained in more than $k$ induced $C_5$s, delete $v_2v_3$ and decrease $k$ by 1.

**2.37** By induction, show that every graph on at least $4^k$ vertices has either a clique or an independent set on $k$ vertices. Observe that this implies a kernel of exponential size for the problem.

**2.38** Show that every tournament on $n$ vertices has a transitive subtournament on $\mathcal{O}(\log n)$ vertices. Then, use this fact to show that every oriented directed graph on $n$ vertices has an induced directed acyclic subgraph on $\log n$ vertices. Finally, obtain an exponential kernel for the considered problem.

## Bibliographic notes

The history of preprocessing, such as that of applying reduction rules to simplify truth functions, can be traced back to the origins of computer science—the 1950 work of Quine [391]. A modern example showing the striking power of efficient preprocessing is the commercial integer linear program solver CPLEX. The name "lost continent of polynomial-time preprocessing" is due to Mike Fellows [175].

Lemma 2.2 on equivalence of kernelization and fixed-parameter tractability is due to Cai, Chen, Downey, and Fellows [66]. The reduction rules for Vertex Cover discussed in this chapter are attributed to Buss in [62] and are often referred to as Buss kernelization in the literature. A more refined set of reduction rules for Vertex Cover was introduced in [24]. The kernelization algorithm for FAST in this chapter follows the lines provided by Dom, Guo, Hüffner, Niedermeier and Truß [140]. The improved kernel with $(2+\varepsilon)k$ vertices, for $\varepsilon > 0$, is obtained by Bessy, Fomin, Gaspers, Paul, Perez, Saurabh, and Thomassé in [32]. The exponential kernel for Edge Clique Cover is taken from [234], see also Gyárfás [253]. Cygan, Kratsch, Pilipczuk, Pilipczuk and Wahlström [114] showed that Edge Clique Cover admits no kernel of polynomial size unless NP $\subseteq$ coNP/ poly (see Exercise 15.4, point 16). Actually, as we will see in Chapter 14 (Exercise 14.10), one can prove a stronger result: no subexponential kernel exists for this problem unless P $=$ NP.

Kőnig's theorem (found also independently in a more general setting of weighted graphs by Egerváry) and Hall's theorem [256, 303] are classic results from graph theory, see also the book of Lovász and Plummer [338] for a general overview of matching theory. The crown rule was introduced by Chor, Fellows and Juedes in [94], see also [174]. Implementation issues of kernelization algorithms for vertex cover are discussed in [4]. The kernel for Maximum Satisfiability (Theorem 2.16) is taken from [323]. Abu-Khzam used crown decomposition to obtain a kernel for $d$-Hitting Set with at most $(2d-1)k^{d-1}+k$ elements [3] and for $d$-Set Packing with $\mathcal{O}(k^{d-1})$ elements [2]. Crown Decomposition and its variations were used to obtain kernels for different problems by Wang, Ning, Feng and Chen [431], Prieto and Sloper [388, 389], Fellows, Heggernes, Rosamond, Sloper and Telle [178], Moser [369], Chlebík and Chlebíková [93]. The expansion lemma, in a slightly different form, appears in the PhD thesis of Prieto [387], see also Thomassé [420, Theorem 2.3] and Halmos and Vaughan [257].

The Nemhauser-Trotter theorem is a classical result from combinatorial optimization [375]. Our proof of this theorem mimics the proof of Khuller from [289]. The application of the Nemhauser-Trotter theorem in kernelization was observed by Chen, Kanj and Jia [81]. The sunflower lemma is due to Erdős and Rado [167]. Our kernelization for $d$-Hitting Set follows the lines of [189].

Exercise 2.11 is from [392], Exercise 2.15 is from [121, 383], Exercise 2.16 is from [281], Exercise 2.18 is from [251] and Exercise 2.23 is from [190]. An improved kernel for the above guarantee variant of Maximum Bisection, discussed in Exercise 2.18, is obtained by Mnich and Zenklusen [364]. A polynomial kernel for Split Edge Deletion (Exercise 2.36) was first shown by Guo [240]. An improved kernel, as well as a smaller kernel for Split Vertex Deletion, was shown by Ghosh, Kolay, Kumar, Misra, Panolan, Rai, and Ramanujan [228]. It is worth noting that the very simple kernel of Exercise 2.4 is probably optimal by the result of Kratsch, Philip, and Ray [308].

http://www.springer.com/978-3-319-21274-6

Parameterized Algorithms
Cygan, M.; Fomin, F.V.; Kowalik, Ł.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; Saurabh, S.
2015, XVII, 613 p. 84 illus., 25 illus. in color., Hardcover
ISBN: 978-3-319-21274-6