Chapter 2
Treewidth

The objective of this chapter is to present the basics techniques on graphs of bounded treewidth. After providing examples of algorithms on graphs of bounded treewidth, we discuss Courcelle’s theorem, an algorithmic meta-theorem describing a large class of problems applicable to this approach. Most of the applications of treewidth requires a tree decomposition, we give a simple FPT-approximation for Treewidth. The chapter is concluded with applications of treewidth on planar graphs via shifting, bidimensionality, and irrelative vertex techniques.

2.1 Trees and separators

We want to make a party and to invite some of our colleagues. Our goal is to maximize the total fun factor of the invited people and we want everyone to have fun. However, there is not much fun when your direct boss is invited. We model this problem as follows. We have a rooted tree $T$ representing job relationships, vertices of the tree have weights $w(v), v \in V(T)$, representing the amount of fun of a particular person. The task is to find the maximum weight of an independent set, that is a set of pairwise nonadjacent vertices. We refer to this problem as to Weighted Independent Set.

This problem can be easily solved on trees by making use of dynamic programming. We solve a large number of subproblems that depend on each other. The answer is a single subproblem. We pick up an arbitrary vertex of $T$ and fix it as a root. For a vertex $v$ of $T$, let $T_v$ be the subtree rooted at $v$ and let $A[v]$ be the maximum weight of an independent set in $T_v$. Then our goal is to determine $A[r]$ for the root $r$. We define an auxiliary value $B[v]$ to be the maximum weight of an independent set in $T_v$ that does not contain $v$.

The computations of the values $A[v]$ and $B[v]$ for leaves of $T$ are trivial, and for other vertices the values are calculated in a bottom-up order. Assume that $v_1, \ldots, v_k$ are the children of $v$. Then we can use the following recurrence
relations

\[ B[v] = \sum_{i=1}^{k} A[v_i] \]

and

\[ A[v] = \max\{B[v], w(v) + \sum_{i=1}^{k} B[v_i]\}. \]

Let us remark that with a slight modification the algorithm can find not only the maximum weight but the corresponding independent set as well. The whole procedure can be clearly implemented in polynomial time.

We continue with Weighted Independent Set, now we try to solve the problem for an arbitrary graph \( G \). The idea is again to perform dynamic programming, but this time instead of going from leaves to the root, we go via separating set of vertices.

Let us fix an ordering \( \sigma = (v_1, v_2, \ldots, v_n) \) of the vertices of \( G \). For \( j \in \{1, 2, \ldots, n\} \), we use \( V_j \) to denote the set of the first \( j \) vertices of \( \sigma \). Thus \( G[V_n] = G \). The boundary of \( V_j \) is the subset of vertices of \( V_j \) adjacent to vertices outside \( V_j \). We denote the boundary of \( V_j \) by \( \partial(V_j) \). For example, in Fig. 2.1, for \( \sigma = (a, b, c, d, e, f, g, h, i) \) and \( j = 5 \), we have \( V_j = \{a, b, c, d, e\} \) and \( \partial(V_j) = \{d, e\} \). Let us note that every path from \( v \in V_j \) to \( u \notin V_j \) contains an edge with one endpoint in \( V_j \) and the other in \( V(G) \setminus V_j \), and hence every \( uv \)-path contains a vertex from \( \partial(V_j) \). In other words, \( \partial(V_j) \) separates \( V_j \) from the other vertices of \( G \).

Consistency of \( \partial(V_j) \) with other chapters.

Coming back to maximum independent sets in \( G \), our plan is to perform dynamic programming over sets \( V_j \). To compute different partial solutions for \( V_j \), we use only the information computed for \( \partial(V_j) \).

More formally. For every \( j \in \{1, \ldots, n\} \) and every subset \( S \) of \( \partial(V_j) \), we keep the value \( c[S, j] \), which is the maximum weight of an independent set \( I \) in \( G[V_j] \) such that \( I \cap \partial(V_j) = S \). If \( S \) is not an independent set in \( G[V_j] \), we put \( c[S, j] = -\infty \). In the example in Fig. 2.1, for \( j = 5 \) and \( \partial(V_j) = \{d, e\} \), assuming all vertices are of unit weights, we have the following table of values for \( c[S, j] \)

| \( c[\emptyset, j] \) | 1 |
| \( c[\{d\}, j] \) | 2 |
| \( c[\{e\}, j] \) | 2 |
| \( c[\{d, e\}, j] \) | -\infty |

Because the boundary of \( V_n \) is the empty set, the solution to the problem can be found by computing the maximum value of \( c[\emptyset, n] \).

What remains, is to explain how to obtain a table for \( \partial(V_{j+1}) \) from a table for \( \partial(V_j) \). By definition, \( V_{j+1} = V_j \cup \{v_{j+1}\} \) but it is not necessary
that $\partial(V_{j+1}) = \partial(V_j) \cup \{v_{j+1}\}$. However, $\partial(V_{j+1}) \subseteq \partial(V_j) \cup \{v_{j+1}\}$ and every vertex $u \in \partial(V_j) \setminus \partial(V_{j+1})$ is adjacent to $v_{j+1}$.

Let $S \subseteq \partial(V_{j+1})$. If $S$ is not an independent set, then we put $c[S, j + 1] = -\infty$. From now on we assume that $S$ is an independent set. Let $I$ be an independent set of maximum weight such that $I \cap \partial(V_{j+1}) = S$. If $v_{j+1} \in S$, then because every vertex of $\partial(V_j) \setminus \partial(V_{j+1})$ is adjacent to $v_{j+1}$, we have that $I \cap \partial(V_j) = S \setminus \{v_{j+1}\}$. In this situation,

$$c[S, j + 1] = w(v_{j+1}) + c[S \setminus \{v_{j+1}\}, j].$$

If $v_{j+1} \in \partial(V_{j+1})$ and $v_{j+1} \notin S$, then by the definition of set $I$, $v_{j+1} \notin I$. Also if $v_{j+1}$ has a neighbor in $S$, then $S \cup \{v_{j+1}\}$ is not an independent set, hence $v_{j+1} \notin I$. Thus in these two cases, we have

$$c[S, j + 1] = \max_{S \subseteq S' \subseteq \partial(V_j)} c[S', j]. \quad (2.1)$$

Finally, the only missing case is when $v_{j+1} \notin \partial(V_{j+1})$ and $v_{j+1}$ has no neighbor in $S$. Here, because $v_{j+1}$ has no neighbors outside $V_{j+1}$ and because all neighbors of $v_{j+1}$ in $V_{j+1}$ are in $\partial(V_j)$, every maximal independent set extending $S$, and thus set $I$ in particular, should contain $v_{j+1}$. Hence in this case

$$c[S, j + 1] = w(v_{j+1}) + \max_{S \subseteq S' \subseteq \partial(V_j)} c[S', j]. \quad (2.2)$$

Let us remark that as far as the tables $c[S, j]$ are constructed, an independent set of maximum weight can be found from these tables.

Concerning the running time of the algorithm. For an ordering $\sigma$, let

$$t_\sigma = \max_{1 \leq j \leq n} |\partial(V_j)|.$$ 

Then for every $i \in \{1, \ldots, n\}$, we keep a table with $O(2^{t_\sigma})$ entries. The most time consuming steps are the steps $(2.1)$ and $(2.2)$, where for every $\partial(V_j)$ and $i \geq 0$, we go through subsets of $\partial(V_j)$ of size $i$. Each of theses steps can be implemented in time $t_\sigma^{O(1)}$. The running time is then

$$\sum_{i=1}^{t_\sigma} \left( \frac{t_\sigma}{i} \right) 2^i \cdot t_\sigma^{O(1)} = 3^{t_\sigma} \cdot t_\sigma^{O(1)}.$$

Since we repeat computations for each $j$ from $1$ to $n$, the total running time of our algorithm is $3^{t_\sigma} \cdot t_\sigma^{O(1)}. n$. We do not discuss here the exponent of the polynomial $t_\sigma^{O(1)}$. It depends on how fast we can organize a search in the tables, how we check that a set $S$ is independent, and thus on our assumptions concerning the model of computation, the data structure we use. While for implementations of dynamic programming algorithms optimization of $t_\sigma^{O(1)}$ is an important and a non-trivial issue, this goes out of the scope of this
Let us define the following graph parameter, vertex separation number of a graph $G$

$$\text{vsn}(G) = \min\{t_\sigma \mid \sigma \text{ is a permutation of } V(G)\}.$$ 

What we have shown is that we can find a maximum independent set in a graph in time $3^{\text{vsn}(G)} \cdot \text{vsn}(G)^{O(1)} \cdot n$ if the corresponding permutation is given.

Two important questions are not answered so far

- How to find a good permutation?
- While the vertex separation number of a tree can be arbitrarily large, the dynamic programs we used on trees and on graphs with small separation numbers, are quite similar. Is it possible to combine both approaches?

In what follow we provide answers to both questions. The answer to the questions will be given by making use of tree decompositions and treewidth. We will show that trees and graphs with small vertex separation numbers have small treewidth and explain how the ideas of dynamic programming can be adapted to tree decompositions. And we give a parameterized approximation algorithm to compute the treewidth of a graph, thus instead of the vertex separation, we can approximate the treewidth, and use this machinery to solve the problem on hands.

### 2.2 Path and tree decompositions

A path decomposition of graph $G$ is a sequence of bags $X_i \subseteq V(G)$, $i \in \{1, \ldots, r\}$,

$$(X_1, X_2, \ldots, X_r)$$

such that

(P1) $\bigcup_{1 \leq i \leq r} X_i = V(G)$. In other words, every vertex of $G$ is in some (maybe several) bags.

(P2) For every $vw \in E(G)$, there exists $i \in \{1, \ldots, r\}$ such that bag $X_i$ contains both $v$ and $w$.

(P3) For every $v \in V(G)$, let $i$ be the minimum and $j$ be the maximum indices of the bags containing $v$. Then for every $k$, $i \leq k \leq j$, we have $v \in X_k$. In other words, the indices of the bags containing $v$ form an interval.

See Fig. 2.1 for an example of path decomposition. The width of a path decomposition $(X_1, X_2, \ldots, X_r)$ is $\max_{1 \leq i \leq r} |X_i| - 1$. The pathwidth of a graph $G$, denoted by $\text{pw}(G)$, is the minimum width of a path decomposition of $G$. 
Another way to interpret the bags of a path decomposition as nodes of a path and two consecutive bags correspond to two adjacent nodes of the path. This interpretation will become handy when we introduce tree decomposition.

For us the most crucial property of path decompositions is the following separation property.

**Lemma 2.1.** Let $(X_1, X_2, \ldots, X_r)$ be a path decomposition. Then for every $j \in \{1, \ldots, r-1\}$, $\partial(X_1 \cup X_2 \cdots \cup X_j) \subseteq X_j \cap X_{j+1}$. In other words, $X_j \cap X_{j+1}$ separates $X_1 \cup X_2 \cdots \cup X_j$ from the other vertices of $G$.

**Proof.** Let $V_j = X_1 \cup X_2 \cdots \cup X_j$. Targeting towards a contradiction, let us assume that there is an edge $uv \in E(G)$ such that $u \in V_j$, $v \notin V_j$ but $u \notin X_j \cap X_{j+1}$. Let $i$ be the largest index such that $u \in X_i$ and $k$ be the smallest index such that $v \in X_k$. Because $u \in V_j$ and $u \notin X_j \cap X_{j+1}$, (P3) implies that $i \leq j$. Since $v \notin V_j$, $k \geq j+1$. Thus $i < k$. On the other hand, by (P2), there should be a bag $X_{k+}$ containing both $u$ and $v$. We obtain that $\ell \leq i < k \leq \ell$, which is a contradiction. \qed

By Lemma 2.1, an intersection of two consecutive bags separates the vertices to the “left” from the vertices to the “right”. This makes the pathwidth to look very similar to the vertex separation number. Let us formalize these similarities.

It is more convenient to work with nice decompositions. A path decomposition $(X_1, X_2, \ldots, X_r)$ of a graph $G$ is **nice** if $|X_1| = |X_r| = 1$, and for every
where

\[ \text{Tree decomposition.} \]

does not exceed

\[ \text{bag after} \]

@\[\introduce \text{nodes} \]

introduce nodes of

\[ \text{position of width} \]

\[ \text{i} \]

holds and hence \( (P3) \), every vertex of \( G \) belongs to consecutive sets of bags, and thus the number of bags in a nice path decomposition is at most twice the number of vertices of \( G \). In Exercise 2.43, the reader is asked to construct an algorithm that for a given path decomposition of width \( p \) constructs a nice path decomposition of width \( p \) in time \( O(p^2 n) \).

In the previous section, we consider the parameter vertex separation number of a graph. It appears, that this is exactly the pathwidth of a graph.

**Lemma 2.2.** For any graph \( G \), \( \text{vsn}(G) = \text{pw}(G) \).

*Proof.* Let \( \sigma = (v_1, v_2, \ldots, v_n) \) be an ordering of the vertices of \( G \). We define vertex sets \( X_1, X_2, \ldots, X_n \) as follows. We put \( X_1 = \{v_1\} \) and for \( i > 1, X_i = \partial(V_{i-1}) \cup \{v_1\} \). We claim that \( (X_1, X_2, \ldots, X_n) \) is a path decomposition. Property \( (P1) \) holds trivially. For every edge \( v_i v_j \) of \( G \), \( i < j \), both its endpoints are in bag \( X_j \). To show that \( (P3) \) holds, let \( v_i \) be a vertex of \( G \). If all neighbors of \( v_i \) appear before \( v_i \) in \( \sigma \), then \( v_i \) appears only in one bag, namely \( X_i \). Otherwise, let \( j \) be the largest index of a neighbor of \( v_i \). Then \( v_i \) belongs to each of the bags from \( \{X_i, X_{i+1}, \ldots, X_j\} \). Thus \( (P3) \) holds and hence \( (X_1, X_2, \ldots, X_n) \) is a path decomposition. Because for every \( i \in \{1, \ldots, n\}, |X_i| \leq |\partial(V_{i-1})| + 1 \) it follows that the width of the constructed path decomposition does not exceed \( t_\sigma = \max_{1 \leq j \leq n} |\partial(V_j)| \).

For the opposite direction. Let \( P = (X_1, X_2, \ldots, X_r) \) be a path decomposition of width \( p \). We can assume that \( P \) is a nice path decomposition. There are exactly \( n \) introduce nodes (or bags) in \( P \). Let \( (X'_1, X'_2, \ldots, X'_r) \) be the introduce nodes of \( P \) given in the same ordering as in \( P \). We construct vertex ordering \( \sigma = (v_1, v_2, \ldots, v_n) \) of \( G \) such that \( v_i \) is the vertex introduced in the introduce node \( X'_i \). By Lemma 2.1, we have that for every \( i \in \{1, \ldots, n\}, \partial(V_i) \subseteq X'_i \). Moreover, if \( X'_i \) is a bag with \( p + 1 \) vertices, then in \( P \) the next bag after \( X'_i \) is a forget node with \( p \) vertices. Thus \( t_\sigma = \max_{1 \leq j \leq n} |\partial(V_j)| \) does not exceed \( p \), and we have that \( t_\sigma \leq p \).

**Tree decomposition.** A tree decomposition of a graph \( G \) is a pair \( T = (T, \chi) \), where \( T \) is a tree and mapping \( \chi \) assigns to every node \( t \) of \( T \) a vertex subset \( X_t \) (called a bag) such that

\[
\text{(T1)} \quad \bigcup_{t \in V(T)} X_t = V(G).
\]

\[
\text{(T2)} \quad \text{For every } uvw \in E(G), \text{ there exists a node } t \text{ of } T \text{ such that } \text{bag } \chi(t) = X_t \text{ contains both } v \text{ and } w.
\]

\[
\text{(T3)} \quad \text{For every } v \in V(G), \text{ the set } \chi^{-1}(v), \text{ i.e. the set of nodes } T_v = \{ t \in V(T) \mid v \in X_t \} \text{ forms a connected subgraph (subtree) of } T.
\]
The width of tree decomposition $\mathcal{T} = (T, \chi)$ equals $\max_{t \in V(T)} |X_t| - 1$, i.e. the maximum size of its bag minus one. The treewidth of a graph $G$, $\text{tw}(G)$, is the minimum width of a tree decomposition of $G$.

To distinguish the vertices of the decomposition tree $T$ and the vertices of the graph $G$, we will refer to the vertices of $T$ as to the nodes. Let us note that the path decomposition of a graph can be defined as a tree decomposition with tree $T$ being a path.

In Lemma 2.1, we have proved that for every pair of adjacent nodes of the path of a path decomposition, the intersection of the corresponding bags is a separator. The following lemma establishes similar separation properties of bags of a tree decomposition. Its proof is similar to the proof of Lemma 2.1 and is left as an exercise (Exercise 2.54).

**Lemma 2.3.** Let $(T, \chi)$ be a tree decomposition and $st$ be an edge of $T$. The forest $T - st$ obtained from $T$ by deleting $st$ contains two connected components $T_s$ and $T_t$. Let $V_s$ and $V_t$ be the vertices of $G$ contained in bags of $T_s$ and $T_t$, correspondingly. Then $\partial(V_s) \subseteq X_s \cap X_t$. Thus $X_s \cap X_t$ separates $V_s$ and $V_t$. In other words, every path in $G$ with one endpoint in $V_s$ and one in $V_t$ contains a vertex from $X_s \cap X_t$.

In the remaining part of this section we discuss the relation of the width parameters with graph searching and chordal graphs. While these results are not directly related to the topics discussed in the book, they can be useful when working with treewidth.

### 2.2.1 Graph searching, interval and chordal graphs

Suppose that $G$ is a graph representing a system of tunnels where an agile and omniscient fugitive with unbounded speed is hidden. An alternative interpretation of the same problem is to view it as the task of cleaning the tunnels from some poisonous gas, or a computer network infected by a virus, which can spread very fast. Initially the whole graph is contaminated. The object of the game is to clear all edges, using one or more searchers. A move of a searcher can be one of the following: placement of the searcher on a vertex $v$ or removal of the searcher from a vertex $v$. A search program is a sequence of moves of searchers. An edge is declared clean when both its endpoints are occupied by searchers. A cleared edge become contaminated at some moment of searching if at that moment it can be connected by a path without searchers with a contaminated edge. In terms of searching it means that the fugitive can run to already cleared area through a vertex containing no searchers. If this occur, we say that the edge is recontaminated. The node search number, of a graph $G$ is the minimum number of searchers required to clear $G$. 

An important question in graph searching is if \( k \) searchers can clear a graph, can they do it in a monotone way, i.e. in such a way that no recontamination occur? The proof of the following theorem of LaPaugh [145] on the monotonicity of graph searching is beyond the scope of this book.

**Theorem 2.4.** Recontamination does not help to search a graph. In other words, for any graph, if \( k \) searchers can clear the graph, then \( k \) searchers can clear the graph such that no edge is recontaminated.

A graph \( G \) is an interval graph, if and only if one can associate with each vertex \( v \in V(G) \) an interval \( I_v = [l_v, r_v] \) on the real line, such that for all \( v, w \in V(G), \ v \neq w: vw \in E(G) \) if and only if \( I_v \cap I_w \neq \emptyset \). The set of intervals \( I = \{I_v\}_{v \in V} \) is called an interval representation of \( G \). It is easy to check that every interval graph has an interval representation in which the left endpoints are distinct integers \( 1, 2, \ldots, n \). Such a representation is said to be canonical.

A graph \( G \) is a supergraph of the graph \( G' \) if \( V(G') = V(G) \) and \( E(G') \subseteq E(G) \). Let \( G \) be an interval graph and let \( I = \{I_v\}_{v \in V(G)} \) be a canonical representation of \( G \). We define the interval width of \( G \), as the minimum of the maximum clique size, where minimum is taken over all interval supergraphs of \( G \).

**Theorem 2.5.** For any graph \( G \), the following are equivalent

(i) The vertex separation number of \( G \) is at most \( k \);
(ii) The node search number of \( G \) is at most \( k + 1 \);
(iii) The interval width of \( G \) is at most \( k + 1 \);
(iv) The pathwidth of \( G \) is at most \( k \).

**Proof.** (i) \( \Rightarrow \) (ii). Let \( \sigma = (v_1, v_2, \ldots, v_n) \) be an ordering of \( V(G) \) such that for every \( i, |\partial(V_i)| \leq k \). We consider a search program with \( n \) steps, where at each step the searchers occupy the set \( \partial(V_i) \cup \{v_i\} \). This program uses at most \( k + 1 \) searchers. For every edge \( v_i, v_j, j > i \), both its endpoints at the \( j \)-th step of searching are occupied by searchers, hence every edge was cleared at some step. Let us note that no recontamination occurs. Indeed, if at some step \( i \) for the first time a cleared edge is recontaminated, it means that there is a path \( v_p, v_q, v_i \) such that edge \( v_p, v_q \) is cleared and edge \( v_q, v_i \) is contaminated. Thus recontamination occurs through vertex \( v_q \) and this vertex is not occupied by a searcher at this step. Because \( v_p, v_q \) is cleared, we have that \( q \leq i \), and because edge \( v_q, v_i \) is contaminated and was not cleared before — this is the first time recontamination occurs — we have that \( t > i \). But then \( v_q \in \partial(V_i) \cup \{v_i\} \) and thus by the definition of our search program, \( v_q \) is occupied by a searcher.

(ii) \( \Rightarrow \) (iii). Let \((Z_1, Z_2, \ldots, Z_n)\) be a monotone search program using \( k + 1 \) searchers, where for every step \( i \), the searchers occupy vertices \( Z_i \subseteq V(G) \). For every vertex \( v \), we define \( \ell(v) \) as the first step and \( r(v) \) as the last step when \( v \) was occupied by a searcher. Because the program monotone, we can assume that for every \( i \in \{\ell(v), r(v)\} \), vertex \( v \) is occupied during the \( i \)-th
step. For each vertex $v$, we associate interval $I_v = [\ell(v), r(v)]$. The intersection graph $G_I$ of intervals $\mathcal{I} = \{I_v\}_{v \in V}$ is, of course, an interval graph. This graph is also a supergraph of $G$—since every edge is cleared, there is a step for each edge $uv$ such that both its endpoints are occupied by searchers, and thus the corresponding intervals intersect. Finally, the maximum size of a clique in $G_I$, is the maximum number of intervals intersecting in one point, which is at most $\max_{1 \leq i \leq n} |Z_i| \leq k + 1$.

$(iii) \Rightarrow (iv)$. Let $G_I$ be an interval supergraph of $G$ with the maximum clique-size at most $k + 1$ and let $\mathcal{I} = \{I_v\}_{v \in V}$ be the canonical representation of $G_I$. For $i \in \{1, \ldots, n\}$, we define $X_i$ as the set of vertices $v$ of $G$ whose corresponding intervals $I_v$ contain $i$. Then $|X_i| \leq k + 1$. We claim that $(X_1, \ldots, X_n)$ is a path decomposition of $G$. Indeed, property (P1) holds trivially. For (P2), because $G_I$ is a supergraph of $G$, for every edge $uv \in E(G)$, we have $I_v \cap I_u \neq \emptyset$ and thus there is $X_i$ containing $u$ and $v$. By the construction of sets $X_i$, for every vertex $v$, the sets containing $v$ form an interval and thus (P3) also holds.

$(iv) \Rightarrow (i)$. Is already proven in Lemma 2.2

A similar set of equivalent characterizations is known for the treewidth. The role of interval graphs is now played by chordal graphs. A graph is chordal if it does not contain an induced cycle of length more than three, i.e. every cycle of length more than three has a chord. Sometimes chordal graphs are called triangulated. Let us observe that interval graphs are chordal, see Exercise 2.58.

We define the chordal width of $G$ as the minimum of the maximum clique size, where minimum is taken over all chordal supergraphs of $G$. Before stating the proof of the equivalence of the chordal width and the treewidth, let us introduce the notion of graph searching relevant to the treewidth.

The rules of the search game for treewidth are exactly the same as for the node searching. At every move a searcher can be placed or removed from a vertex of graph $G$. The difference is that at every move of the game, the searchers placed on a vertex set $S$, know in which connected component of $G - S$ the fugitive is located. In this setting the game has the following interpretation. Searchers (let us call them cops to distinguish from the node search game) fly in helicopters. At every move of the game, each of the cops can be in a vertex of the graph or in a helicopter, that is, temporarily removed from the graph. The task of the cops is to land on the vertex containing the robber. The robber can run arbitrarily fast, and seeing a cop landing at a vertex can run to another vertex of the graph via a path free of cops. It is easy to see that when cops do not see the robber, this is exactly the node search game. However, if they see the robber, they can gain a lot. For example, two cops can catch a robber on any tree but catching an invisible fugitive on an $n$-vertex can require $\log_3 n$ searchers, see Exercise 2.42.

We say that two subsets $A$ and $B$ of $V(G)$ touch if either they have a vertex in common, or there is an edge with endpoints in $A$ and in $B$. A bramble is

---

To check if chordal graphs were not defined in branching in Chapter 1.
a set of mutually touching connected vertex sets in $G$. A subset $C \subseteq V(G)$ covers a bramble $B$ if it meets every element of $B$. The least number of vertices covering bramble $B$ is the order of $B$. It is not difficult to see that if $G$ has a bramble $B$ of order $k + 1$, then $k$ cops cannot catch the robber. Indeed, for every position of cops in $G$, there is an element $X$ of $B$ containing no cops. Robber selects a vertex of this unoccupied set. If in the next move of cops, one of the cops lands on a vertex from $X$, there is another element of $B$, say $Y$, unoccupied by cops. Because $X$ and $Y$ touch, the robber runs to a vertex of $Y$. The following deep result of Seymour and Thomas [189] provide a “min-max” characterization of the treewidth by making use of brambles. We state this result without proof.

**Theorem 2.6.** For every $k \geq 0$ and graph $G$, the treewidth of $G$ is at least $k$ if and only if it contains a bramble of order at least $k + 1$.

The following theorem is the analogue of Theorem 2.5 for treewidth. The main steps of its proof are discussed in Exercise 2.59.

**Theorem 2.7.** For any graph $G$, the following are equivalent

(i) The treewidth of $G$ is at most $k$;
(ii) The chordal width of $G$ is at most $k + 1$;
(iii) $k + 1$ cops can catch a visible robber.

### 2.3 Dynamic programming on graphs of bounded treewidth

In Section 2.1, we gave an algorithm solving Weighted Independent Set by making use of dynamic programming over boundaries of sets. As we also have shown (Lemma 2.2), the vertex separation number is exactly the pathwidth of a graph. We start this section by explaining how similar dynamic programming can be done by making use of a nice path decomposition of $G$. While this follows almost the same steps as for the vertex separation number, it is important to repeat these steps; it will brings us naturally to dynamic programming on graph of bounded treewidth.

Let $(X_1, X_2, \ldots, X_r)$ be a nice path decomposition of graph $G$ of width $p$. For $j \in \{1, 2, \ldots, r\}$, we put

$$V_j = \bigcup_{i=1}^{j} X_i.$$ 

Thus $G[V_r] = G$. For each $j$ and $S \subseteq X_j$, we compute $c[S, j]$, which is the maximum weight of an independent set $I$ in $V_j$ extending $S$, i.e. such that $I \cap X_j = S$. For $j = 1$, $X_1$ consists of one vertex, say $v$, and we have only
two values $c[\emptyset, 1] = 0$ and $c[X_1, 1] = w(v)$. If $S$ is not an independent set, we put $c[S, j] = -\infty$. For $j > 1$, computations of $c[S, j]$ depend on the type of $X_j$. Because we are working with a nice path decomposition, there are two types of nodes, introduce and forget.

**Introduce node.** If $X_j = X_{j-1} \cup \{v\}$, let $I$ be an optimal independent set of $V_j$ extending $S$. By Lemma 2.1, every neighbor of $v$ in $V_j$ is in $X_{j-1} \cap X_j$. Thus the set $I \cap \{v\}$ is an optimal extension of $S \cap \{v\}$ in $V_{j-1}$. Therefore

$$c[S, j] = \begin{cases} w(v) + c[S \setminus \{v\}, j - 1], & \text{if } v \in S, \\ c[S, j - 1], & \text{otherwise}. \end{cases}$$

**Forget node.** When $X_j = X_{j-1} \setminus \{v\}$, then

$$c[S, j] = \max\{c[S \cup \{v\}, j - 1], c[S, j - 1]\}.$$ 

Concerning the running time, for every $j$, we perform $2^{|X_j|} \cdot |X_i|^\mathcal{O}(1)$ operations to compute all values $c[S, i]$. However, to estimate the running time we have to be careful. The reason is that in several steps of the algorithm, one of the basic operations we have to perform is the adjacency check for a pair of vertices (we need it, for example, to verify that a set $S$ is independent). However, because graph $G$ is of treewidth $t$, it is possible to construct a data structure in time $t^\mathcal{O}(1)n$ such that all required adjacency tests can be done in time $\mathcal{O}(t)$. Thus for every $j$, it takes time $2^{|X_j|} \cdot |X_i|^\mathcal{O}(1)$ to compute all values $c[S, i]$. Since the number of bags in the nice path decomposition is $\mathcal{O}(n)$, the total running time of the algorithm is $2^p \cdot p^\mathcal{O}(1) \cdot n$, where $p$ is the width of the path decomposition. Summing up, we obtain the following lemma.

**Lemma 2.8.** Let $G$ be an $n$-vertex graph given together with its path decomposition of width $p$. Then Weighted Independent Set is solvable in time $2^p \cdot p^\mathcal{O}(1) \cdot n$.

The algorithm of Lemma 2.8 is quite simple, and it is very natural to ask if it running time can be improved. As we will see in Chapter 8, Theorem 9.37, it can be argued that the bound of Lemma 2.8 is tight.

Now we proceed to generalize the algorithm for path decompositions to tree decompositions. Again, it is convenient to work with nice decompositions. A rooted tree decomposition $T = (T, \chi)$ of a graph $G$ is nice if each of its nodes is one of the following four types.

- **Leaf node:** a node $i$ with $|X_i| = 1$ and no children.
- **Introduce node:** a node $i$ with exactly one child $j$ such that $X_i = X_j \cup \{v\}$ for some vertex $v \notin X_j$; we say that $v$ is introduced in $i$. 

Every tree decomposition of width $t$ can be turned into a nice tree decomposition of width $t$ in time $O(t^2 \cdot n)$, see Exercise 2.57.

In what follows, we give several examples of dynamic programming on graphs of bounded treewidth. In this section we will always assume that we are given a nice tree decomposition $T = (T, \chi)$ of width $t$. In each of the examples, we compute tables with partial solutions from leaves to the root of the tree in the nice tree decomposition. Every time we compute the tables for a new node from the tables of its children according to the type of the node. We will use the following notations. The root of $T$ is denoted by $r$. For node $i$ of $T$, we use $T_i$ to denote the subtree of $T$ rooted in $i$ and by $V_i$ the set of vertices of $G$ contained in the bags of $T_i$. Thus $G[V_r] = G$.

In all examples we show how to compute the (maximum) minimum size of a solution. A solution of the optimum size can be easily constructed by additional bookkeeping and going backwards through stored answers to subproblems.

### 2.3.1 Maximum Independent Set

We continue with Weighted Independent Set. As in the path decomposition example, for each $i \in V(T)$ and $S \subseteq X_i$, we compute $c[S, i]$ which is the maximum weight of an independent set $I$ in $V_i$ such that $I \cap X_i = S$.

**Leaf node.** Each leaf node contains one vertex and here the computation of $c[S, i]$ is trivial.

**Introduce node.** Here there is no difference with the arguments we gave for path decompositions except that we now use Lemma 2.3. Let $j$ be the child of an introduce node $i$, thus $X_i = X_j \cup \{v\}$ for some $v \notin X_j$. Then

$$c[S, i] = \begin{cases} w(v) + c[S \setminus \{v\}, j], & \text{if } v \in S, \\ c[S, j], & \text{otherwise.} \end{cases}$$

**Forget node.** Let $j$ be the child of a forget node $i$, that is $X_i = X_j \setminus \{v\}$. As in the case with path decomposition, we have

$$c[S, i] = \max\{c[S \cup \{v\}, j], c[S, j]\}.$$
2.3 Dynamic programming on graphs of bounded treewidth

Join node. Let $j$ and $k$ be the children of the join node $i$. Thus $X_i = X_j = X_k$. Then every maximum weight independent set $I$ of $G[V_i]$ such that $I \cap X_i = S$, if the union of two maximum weight independent sets $I_j$ in $G[V_j]$ and $I_k$ in $G[V_k]$, intersecting $X_i$ exactly in $S$. The weight of $I$ is the sum of the weights of $I_j$ and $I_k$ minus the weights of the vertices contained in both sets, which is $S$. In this case we have

\[ c[S, i] = c[S, j] + c[S, k] - w(S). \]

The number of nodes in $T$ is $O(n)$ and for each node we keep $O(2^t)$ different values $c[S, i]$. As in the case of pathwidth (Lemma 2.8 and Exercise 2.52), we preprocess the graph in time $O(tn)$ in such a way that adjacency testing can be done in time $O(t)$. Then the update time for each node is $2^t \cdot t^{O(1)}$.

**Theorem 2.9.** Let $G$ be an $n$-vertex graph given together with its tree decomposition of width $t$. Then Weighted Independent Set is solvable in time $2^t \cdot t^{O(1)} \cdot n$.

Since a graph has a vertex cover of size at most $k$ if and only if it has an independent set of size at least $n - k$, we immediately obtain the following corollary.

**Corollary 2.10.** Let $G$ be an $n$-vertex graph given together with its tree decomposition of width $t$. Then Vertex Cover is solvable in time $2^t \cdot t^{O(1)} \cdot n$.

### 2.3.2 Minimum Dominating Set

Our next example is Dominating Set. Let us remind that a set of vertices $D$ is a dominating set in graph $G$, if $V(G) = N[D]$.

Here we use a refined variant of nice tree decomposition. In the “standard” nice decomposition that we defined in the previous subsection, whenever a vertex is introduced in the introduce node, also all the edges between this vertex and other vertices of the current bag are introduced. In our algorithm for Dominating Set, we refine the notion of nice tree decomposition. Now we have both introduce vertex nodes and introduce edge nodes. Introduce vertex node corresponds to the introduce node in the standard definition, while introduce edge node is defined as follows.

**Introduce edge node:** a node $i$, labelled with edge $uv \in E(G)$ with exactly one child $j$ such that $X_i = X_j$ and $u, v \in X_i$; we say that $uv$ is introduced in $i$. 


We additionally require that every edge in \( E(G) \) is introduced exactly once. Leaf node, forget node and join node are defined just as previously. Given a standard nice tree decomposition, it can be easily extended by the introduce edge nodes, as follows: for every edge \( uv \in E(G) \) add an introduce edge node above the first node with respect to the in-order traversal of the tree that contains \( u \) and \( v \).

We also associate with each node \( i \) of the tree decomposition a subgraph of \( G \) as follows:

\[
G_i = \left( V_i, E_i = \{ e \mid e \text{ is introduced in a descendant of } i \} \right).
\]

For independent set, at every node \( i \) of the tree some partial solutions in \( (V_i, G[V_i]) \) were computed, while now we compute partial solutions in \( G_i \). Moreover, for independent set, we were computing partial solutions according to how independent set intersects a bag of tree decomposition. For domination the situation is more complicated. Here we have to distinguish not only if a vertex is in the dominating set or not, but also if it is dominated. A coloring of bag \( X_i \) is a mapping \( f : X_i \to \{0, \hat{0}, 1\} \) assigning three different colors to vertices of the bag.

- **black**, represented by 1, the meaning is that all black vertices are in the dominating set of the partial solution;
- **white**, represented by 0, means that all white vertices are dominated by the partial solution;
- **grey**, represented by \( \hat{0} \), the vertices which are not dominated by the partial solution and thus to be dominated in the future steps of the algorithm.

Let us note that there are \( 3^{|X_i|} \) colorings of \( X_i \). For a coloring \( f \) of \( X_i \) we denote by \( c[f,i] \) the minimum size of a set \( D \subseteq V_i \) such that

- \( D \cap X_i = f^{-1}(1) \), which is the set of vertices of \( X_i \) colored black;
- Every vertex of \( V_i \setminus X_i \) is either in \( D \) or is adjacent in \( G_i \) to a vertex of \( D \). That is, \( D \) dominates all vertices of \( V_i \) in graph \( G_i \) except some vertices of \( X_i \);
- Each white vertex of \( X_i \), \( f^{-1}(0) \), is adjacent in \( G_i \) to a vertex of \( D \);
- Each grey vertex \( f^{-1}(\hat{0}) \), has no neighbor from \( D \) in graph \( G_i \) and thus has to be dominated in the “future”.

We call such a set \( D \) by a **minimum compatible set** for \( f \) and \( i \). If no minimum compatible set for \( f \) and \( i \) exists, we put \( c[f,i] = +\infty \). Let us note that a minimum compatible set \( D \) for \( f \) and \( i \) is a dominating set of \( G_i \) if no vertex of \( X_i \) is colored grey. Thus the size of a minimum dominating set if \( G \) is the minimum of \( c[f,r] \), where minimum is taken over all colorings \( f \) of the root bag \( X_r \) such that no vertex is colored grey.

It will be convenient to use the following notation. For a subset \( X \subseteq V(G) \), consider a coloring \( f : X \to \{0, \hat{0}, 1\} \). For a vertex \( v \in V(G) \) and a color \( \alpha \in \{0, \hat{0}, 1\} \) we define a new coloring \( f_{v \to \alpha} : X \cup \{v\} \to \{0, \hat{0}, 1\} \) as follows:
\[ f_{v \to \alpha}(x) = \begin{cases} f(x), & \text{when } x \neq v, \\ \alpha, & \text{when } x = v. \end{cases} \]

Computing the values \( c \) for leaf nodes is trivial and we skip it.

**Forget node.** Let \( j \) be the child of a forget node \( i \), that is \( X_i = X_j \setminus \{v\} \).

Note that we cannot forget a grey vertex, because by Lemma 2.3, \( X_i \cap X_j \) separates \( v \) from the vertices still to be processed, and thus there will be no way to dominate \( v \) in the future. Thus

\[
c[f,i] = \min\{c[f_{v \to 1},j], c[f_{v \to 0},j]\}.
\]

**Introduce vertex node.** Let \( j \) be the child of an introduce node \( i \) and \( X_i = X_j \cup \{v\} \) for some \( v \not\in X_j \). Since this node does not introduce edges to \( G_j \), this node is also easy. We just need to be sure not to introduce a white vertex. For a coloring \( f \) of \( X \) and \( Y \subset X \), we use \( f|_Y \) to denote the coloring induced by \( f \) on \( Y \). For every coloring \( f \) of \( X_i \) we put

\[
c[f,i] = \begin{cases} +\infty, & \text{when } f(v) = 0, \\ c[f|_{X_i},j], & \text{when } f(v) = \hat{0}, \\ 1 + c[f|_{X_i},j], & \text{when } f(v) = 1. \end{cases}
\]

**Introduce edge node.** Let \( j \) be the child of an introduce edge node \( i \) labelled with an edge \( uv \). Let \( f \) be a coloring of \( X_i \). Here, if \( u \) is colored black, \( v \) cannot be grey (and vice versa). Moreover, if \( u \) is white and \( v \) is black then in a minimum compatible set either \( u \) is dominated only by \( v \), and we put \( c[f,i] = c[f_{u \to \hat{0}},j] \), or \( u \) is dominated by some other vertex, and then we put \( c[f,i] = c[f,j] \). The situation is analogous if \( v \) is white and \( u \) is black. In the remaining cases the introduced edge does not influence anything, so \( c[f,i] = c[f,j] \). To sum up, we use formula

\[
c[f,i] = \begin{cases} +\infty, & \text{when } (f(u), f(v)) \in \{(1,\hat{0}), (\hat{0},1)\}, \\ \min\{c[f_{u \to \hat{0}},j], c[f,j]\}, & \text{when } (f(u), f(v)) = (0,1), \\ \min\{c[f_{v \to \hat{0}},j], c[f,j]\}, & \text{when } (f(u), f(v)) = (1,0), \\ c[f,j], & \text{otherwise.} \end{cases}
\]

**Join node.** Let \( j \) and \( k \) be the children of the join node \( i \). Thus \( X_i = X_j = X_k \). We say that colorings \( g \) of \( X_j \) and \( h \) of \( X_k \) are consistent with coloring \( f \) of \( X_i \), if for every \( v \in X_i \),

(i) \( f(v) = 1 \) if and only if \( g(v) = h(v) = 1 \),
(ii) \( f(v) = 0 \) if and only if \( (g(v), h(v)) \in \{(0,0), (\hat{0},0), (0,\hat{0})\} \),
(iii) \( f(v) = \hat{0} \) if and only if \( g(v) = h(v) = \hat{0} \).
It is easy to check that every minimum compatible set $D$ for $f$ and $i$ is the union of a minimum compatible set $D_j$ for $g$ and $j$ and $D_k$ for $h$ and $k$, where $g$ and $h$ are colorings consistent with $f$. Moreover, every element of $D$ is in exactly one of the following three sets: $D_j \cap X_i$, $D_k \cap X_i$ or $D \setminus X_i$. By $(i)$, $D \setminus X_i = D_j \setminus X_i = D_k \setminus X_i = f^{-1}(1)$. In other words, $|D| = |D_j| + |D_k| - |f^{-1}(1)|$. Thus

$$c[f, i] = \min_{g, h} \{c[g, j] + c[h, k] - |f^{-1}(1)|\},$$

where the minimum is taken over all colorings $g, h$ consistent with $f$.

Now let us analyze the running time. Clearly, the time needed to process leaf node, introduce vertex/edge node or forget node is $3t \cdot t^{O(1)}$. However, computing the table $c$ in a join node is more time-consuming. The computation can be implemented as follows: for every coloring $f$ we generate all pairs of colorings $g, h$ consistent with $f$. Note that if the pair $g, h$ is consistent with $f$, then for every $v \in X_i$ we have

$$(f(v), g(v), h(v)) \in \{(1, 1, 1), (0, 0, 0), (0, 0, \hat{0}), (0, \hat{0}, 0), (\hat{0}, \hat{0}, \hat{0})\}. \quad (2.3)$$

It follows that for every coloring $f$ there is at least one consistent pair $g, h$, and using the above formula we can easily generate all the consistent pairs in the time proportional to their number (for every vertex $v \in X_i$ we branch to either one (when $f(v) \in \{1, \hat{0}\}$) or three (if $f(v) = 0$) possible values of $(g(v), h(v))$. The formula also easily implies that the total number of triples of colorings $(f, g, h)$ where $g$ and $h$ are consistent with $f$ is exactly $5^{|X_i|} \leq 5^t$.

It follows that the algorithm spends $5^t \cdot t^{O(1)}$ time in every join node. Since the number of nodes in a nice tree decomposition is $O(tn)$ (see Exercise 2.62), we derive the following theorem.

**Theorem 2.11.** Let $G$ be an $n$-vertex graph given together with its tree decomposition of with $t$. Then DOMINATING SET is solvable in time $5^t \cdot t^{O(1)} \cdot n$.

Let us note that the running time of the theorem can be improved to $4^t \cdot t^{O(1)} \cdot n$ by a more careful analyses of the join node case (see Exercise 2.63). In Chapter 5, we show how more clever techniques can reduce the exponential dependence on the treewidth to $3^t$.

### 2.3.3 Minimum Steiner Tree

Our last example is STEINER TREE. We are given an undirected graph $G$, a set of vertices $K \subseteq V$, called terminals. The goal is to find a tree $T \subseteq G$ of minimum size (that is, the minimum number of edges) connecting all the terminals.

While the exponential dependence on the treewidth in the algorithms we discussed so far is single-exponential, for STEINER TREE the situation is
different. In what follows, we give an algorithm of running time $t^O(t)n$. While a single-exponential algorithm for Steiner Tree is possible, it requires new techniques which will be discussed in Chapter 5.

Let $S$ be a Steiner tree and $X_i$ be a bag of $T = (T, \chi)$. The intersection of $S$ with $G[V_i]$ is either empty, or consists of a forest with several connected components, see Fig. 2.2.

In the latter case, each connected component intersects $X_i$ . Also every terminal vertex from $K \cap V_i$ should belong to some connected component of $F$. We try to encode this information by keeping for all possible partitions of $X_i$ the size of a forest in $G[V_i]$ spanning all terminal vertices from $V_i$ and which connected components correspond to the partition of $X_i$. When introducing a new vertex and joining solutions, we should keep track that the result of these operations are again a forest. Also in the join operation, we need a procedure avoiding double-counting of edges from both solutions.

More precisely, we introduce the following function. For a bag $X_i$, a set $X \subseteq X_i$ (a set of vertices untouched by Steiner tree), a forest $H$ in $X_i$ (mainly required to handle double-counting in join operation), and a partition $P = (P_1, P_2, \ldots, P_\ell)$ of $X_i \setminus X$, the value $c[X, H, P, i]$ is the minimum number of edges in a forest $F$ of $G[V_i]$ such that

- $E(F) \cap E(G[X_i]) = E(H)$, that is, forest $F$ is an extension of forest $H$;
- $F$ has exactly $\ell$ connected components $C_1, \ldots, C_\ell$ and for each $s \in \{1, \ldots, \ell\}$, $P_s \subseteq V(C_s)$. Thus partitioning $P$ correspond to connected components of $F$;
- $X \cap V(F) = \emptyset$. Vertices $X$ do not touch a Steiner tree extending $F$;
- Every terminal vertex from $K \cap V_i$ is in $V(F)$.

The size of an optimal Steiner tree is then $\min c[X, H, P, r]$, where minimum is taken over all subsets $X_r \subseteq X$ and degenerate partition $P = (X_r \setminus X)$ corresponding to the case when $F$ is a tree and hence has exactly one component.
In what follows, we discuss only most interesting cases of Introduce and Join nodes, leaving the cases of Leaf and Forget nodes to the reader.

**Introduce node.** Let \( j \) be the child of an introduce node \( i \) and \( X_i = X_j \cup \{v\} \). For every set \( X \subseteq X_i \), partition \( P = (P_1, P_2, \ldots, P_t) \) of \( X_i \setminus X \) and a forest \( H \subseteq E(G[X_i]) \) corresponding to the partition, we do the following. If \( v \) is a terminal vertex, it should not be in \( X \). If \( v \in X \), then \( c[X, H, P, i] = c[X \setminus \{v\}, H, P, j] \).

If \( v \not\in X \), then introducing \( v \) and adding some of its incident edges to our partial Steiner tree results in merging of some of the components and thus changing partition \( P \). We cannot make \( v \) adjacent to more than one vertex from the same connected component because this will create a cycle in the partial Steiner tree. By Lemma 2.3, all edges of the partial Steiner tree incident with \( v \) are in forest \( H \). Without loss of generality, we assume that \( v \in P_1 \). Let \( d = \deg_H(v) \) be the degree of \( v \) in \( H \). Then by introducing \( v \), set \( P_1 \) is obtained by merging \( v \) with \( d \) components. Hence

\[
c[X, H, P, i] = \min_{P'} c[X, H - v, P', j],
\]

where minimum is taken over all partitions \( P' = (P'_1, P'_2, \ldots, P'_t) \), where \( (P'_1, P'_2, \ldots, P'_t) \) is a partition of \( P_1 \setminus \{v\} \) such that each set \( P'_1 \) has exactly one neighbor of \( v \) in \( H \). Let us remark that for the special case \( d = 0 \), we take \( P' = (P_2, \ldots, P_t) \). There are at most \( t^t \) partitions of \( X_i \), thus for every fixed tuple \([X, H, P, i]\), the computation of tables with values of \( c \) takes time \( t^t \cdot t^{O(1)} \).

**Join node.** Let \( j \) and \( k \) be the children of the join node \( i \). Then \( X_i = X_j = X_k \). When merging two partial solutions, we have to be careful because merging can create cycles, see Fig. 2.3.
To avoid cycles while merging, we introduce auxiliary structure. Let $S_1, S_2, \ldots, S_p, p \leq t$, be subsets of $X_i$, not necessarily disjoint. We construct an auxiliary complete graph on vertices of $X_i$. For each set $S_q, q \in \{1, \ldots, p\}$, we built a tree $T_q$ spanning vertices of $S_q$. Let us emphasize that we do not assume that $T_q$ is a subgraph of $G$. We say that sets $S_1, S_2, \ldots, S_p$ can be \textit{merged into a tree} if the union $T_1 \cup T_2 \cup \cdots \cup T_p$ is a tree. In particular, if two sets share more than one vertex, they cannot be merged into a tree. Clearly, the time for checking if given sets can be merged into a tree is polynomial in $t$. We say that a partition $P = (P_1, P_2, \ldots, P_t)$ of $X_i \cap X$ is an \textit{acyclic merge} of partitions $P_j$ and $P_k$, if every set $P_q$ is the union of some sets of $P_j$ and $P_k$ can be merged into a tree.

Thus we have

$$c[X, H, P, i] = \min_{P_j, P_k, H_j, H_k} c[X, H_j, P_j, j] + c[X, H_k, P_k, k] - |E(H_j) \cap E(H_k)|,$$

where $H$ is the union of forests $H_j$ and $H_k$ and $P$ is an acyclic merge of $P_j$ and $P_k$.

The join operation is the most time consuming. To estimate the running time required to fulfill this operation, we note that there at most $2^t$ choices for set $X$ and at most $(t^2)^{t} = t^{O(t)}$ difference forests $H$. For each forest $H$, there are $t^{O(t)}$ choices for $H_j$ and for $H_k$. Finally, there are at most $t^{O(t)}$ partitions $P$; for each such partition, we have $t^{O(t)}$ partitions forming acyclic merges to $P$. Thus up to polynomial factor in $t$, which is anyhow dominated by the $O$ in the exponent, for every node the running time of computing the tables for function $c$ is $t^{O(t)}$. We conclude with the following theorem.

**Theorem 2.12.** Let $G$ be an $n$-vertex graph given together with its tree decomposition of with $t$. Then \textsc{Steiner Tree} is solvable in time $t^{O(t)} \cdot n$.

The examples of dynamic programming on graphs of bounded treewidth have a lot of similarities.
Here we mention the most common problems appearing in our book. Let us also note that the treewidth dependence for a number of such problems will be improved to single-exponential in Chapter 5. We leave the proof of the following theorem as an exercise, see Exercise 2.67.

**Theorem 2.13.** Let $G$ be an $n$-vertex graph given together with its tree decomposition of with $t$. Then

- **Steiner Tree**
- **Feedback Vertex Set**
- **Connected Dominating Set**
- **Odd Cycle Transversal**
- **Hamiltonian Path and $k$-Path**
- **Chromatic Number**
- **Cycle Packing**
- **Connected Vertex Cover**
- **Connected Feedback Vertex Set**
- **ADD YOUR FAVORITE PROBLEM HERE**

are solvable in time $t^{O(t)} \cdot n$.

### 2.4 Courcelle’s Theorem

As we have seen in the previous sections, many optimization problems are FPT being parameterized by the treewidth. A very general class of such problems can be described in a language of logic and this is the essence of Courcelle’s theorem. In this section we give a brief overview of Courcelle’s theorem and related results.

**MSO$_2$** is Monadic Second-Order logic on graphs with quantification both over subsets of vertices and of edges. The syntax of **MSO$_2$** consists of

- logical connectives $\lor$, $\land$, $\neg$, $\leftrightarrow$, $\Rightarrow$, with standard semantics;
- variables for vertices, edges, subsets of vertices, and subsets of edges;
- quantifiers $\forall$, $\exists$ that can be applied to these variables;
- and the following three binary relations:
  1. $u \in U$, where $u$ is a vertex (edge) variable and $U$ is a vertex (edge) set variable, with standard semantics;
  2. $\text{inc}(u, e)$, where $u$ is a vertex variable and $e$ is an edge variable, and the semantics is that $\text{inc}(u, e)$ is true if and only if edge $e$ is incident with vertex $u$;
  3. equality of variables.
The set variables are also called *monadic* (vertex/edge) variables.

Formally, formulas of $\text{MSO}_2$ are evaluated on graphs seen as logical structures. The domain is the disjoint union of the vertex set and the edge set of the graph, while the signature contains unary relations $\text{Vertex}(\cdot)$ and $\text{Edge}(\cdot)$ testing whether a given element is a vertex or an edge (usually used implicitly), and binary relation $\text{inc}(\cdot, \cdot)$ with interpretation that $\text{inc}(u, e)$ holds if and only if $\text{Vertex}(u)$, $\text{Edge}(e)$, and vertex $u$ is incident to edge $e$. We can also enrich the signature with constants representing single vertices or edges and unary relations representing prespecified sets of vertices or edges. Assume that $X$ is a vector of symbols enriching the signature, with interpretation $\overline{A}$ in a graph $G$. Then for a formula $\varphi(X)$ of $\text{MSO}_2$ with free variables from $X$ we say that $\varphi$ is satisfied in $G$ with prescribed interpretation $\overline{A}$ if and only if $h_{G, \overline{A}} = \varphi(\overline{A})$. Investigating logic on graphs is not the main topic of this book. Therefore, we usually make statements about formulas on graphs via this notion of satisfaction, rather than using the formal language of logical structures. The translation to the language of logical structures is in each case obvious.

For example, for vertices $v, u$, the relation of adjacency $\text{adj}(u, v)$ between $u$ and $v$ can be written as $\exists e \in E(G) (\text{inc}(u, e) \land \text{inc}(v, e))$. The inclusion of sets $C \subseteq V$ can be expressed as $\forall v \in V (\forall u, v \in V(G) \text{adj}(u, v))$. Then the formula

$$\exists C \subseteq V(G) \forall v \in C \exists u_1, u_2 \in C (u_1 \neq u_2 \land \text{adj}(u_1, v) \land \text{adj}(u_2, v))$$

is true if graph $G$ has a cycle.

Another example is 3-colorability. Indeed, we quantify existentially 3 monadic variables $X, Y, Z$ corresponding to vertices of the three colors, and check (i) that they form a partition of the vertex set (each vertex belongs to exactly one of the sets), and (ii) that each of them induces an independent set (no two vertices from the same set are adjacent).

$$\exists X, Y, Z \subseteq V (\forall v \in V (v \in X \lor v \in Y \lor v \in Z)) \land (\forall u, v \in V(G) \text{adj}(u, v) \Rightarrow (\neg(u \in X \land v \in X) \land \neg(u \in Y \land v \in Y) \land \neg(u \in Z \land v \in Z)))$$

And finally, one more example of a graph property expressible in $\text{MSO}_2$ is hamiltonicity. To express the property that a graph contains a Hamiltonian cycle passing through all vertices, we quantify existentially one monadic edge variable $C$ corresponding to the edge set of the Hamiltonian cycle, and check (i) that $C$ induces a connected graph (for every partition of $C$ into nonempty $X$ and $Y$ there is a vertex incident both to an edge of $C_1$ and to an edge of $C_2$), and (ii) that each vertex is incident with exactly two edges of $C$.

Two vertex sets $X, Y \subseteq C$ form a partition of $C$ if and only if

$$\text{partition}(X, Y) = \forall e \in C ((e \in X) \lor (e \in Y)) \land (e \in X \iff \neg e \in Y)$$

is true. The connectivity of $C$ can be expressed as
\text{connected}(C) = \forall X, Y \subset C \text{partition}(X, Y) \\
\Rightarrow \exists e_1 \in X \exists e_2 \in Y \forall v \in V(G)(\text{inc}(v, e_1) \land \text{inc}(v, e_2)).

For vertex \( v \in V(G) \) and \( C \subseteq E(G) \), we define

\[
\begin{align*}
\deg_0(C, v) &= \neg \exists e \in C \text{inc}(e, v), \\
\deg_1(C, v) &= \neg \deg_0(C, v) \\
&\land (\neg \exists e_1, e_2 \in C (e_1 \neq e_2 \land \text{inc}(e_1, v) \land \text{inc}(e_2, v))), \text{ and} \\
\deg_2(C, v) &= \neg \deg_0(C, v) \\
&\land (\neg \deg_1(C, v) \\
&\land (\neg \exists e_1, e_2, e_3 \in C (e_1 \neq e_2 \\
&\land e_2 \neq e_3 \land e_1 \neq e_3 \land \text{inc}(e_1, v) \land \text{inc}(e_2, v) \land \text{inc}(e_3, v)))).
\end{align*}
\]

Then hamiltonicity of \( G \) is expressed as

\[
\exists C \subseteq E(\text{connected}(C) \land (\forall v \in V(G) \deg_2(C, v))).
\]

Another way to express hamiltonicity in MSO$_2$ is to observe that if edge subset \( C \) spans vertices of \( G \), then for every pair \( u, v \in V(G) \), \( u \neq v \), there is a \((u, v)\)-path \( P \) in \( C \). In other words, for every pair of different \( u, v \), there is \( P \subset C \), such that for every \( x \in V(G) \), \( x \neq u, x \neq v \), either \( x \) is an internal vertex of \( P \) or \( x \) does not belong to \( P \). Then by defining

\[
\text{span}(C) = \forall u, v \in V \exists P \subseteq C \forall x \in V \\
\left( ((x = u \lor x = v) \land \deg_1(P, x)) \lor (x \neq u \land x \neq v \land (\deg_0(P, x) \lor \deg_2(P, x))) \right),
\]

we can express hamiltonicity as

\[
\exists C \subseteq E(\text{span}(H) \land (\forall v \in V \deg_2(H, v))).
\]

We ready to state the following fundamental theorem of Courcelle [51].

**Theorem 2.14 (Courcelle’s Theorem).** There exists an algorithm that, given an \( n \)-vertex graph \( G \) together with its tree decomposition of width \( t \), and a formula \( \varphi \) of MSO$_2$, checks if \( \varphi \) is satisfied in \( G \) in time \( f(||\varphi||, t) \cdot n \) for some computable function \( f \).

As we will see later, the requirement that \( G \) is given together with its tree decomposition is not necessary. Algorithms computing treewidth will be discussed in the next session and Notes. We remark that the formula \( \varphi \) in Theorem 2.14 can have some free variables (vertex or edge, and possibly monadic) that have prescribed interpretation in \( G \).

In the classical proof of Theorem 2.14, one encodes a tree decomposition of the input graph as a binary tree \( T \) over an alphabet \( \Sigma \), whose size bounded by a function of \( t \). Similarly, the MSO$_2$ formula \( \varphi \) on graphs can be translated...
to an equivalent \( \mathbf{MSO}_2 \) formula \( \varphi' \) on trees over \( \Sigma_t \). The finishing step is to transform \( \varphi' \) into a tree automaton \( \mathcal{A}' \) of size bounded by a function of \( ||\varphi'|| \), and run it on \( T \).

As usual with relations between \( \mathbf{MSO}_2 \) and automata, the size of the obtained automaton may depend non-elementarily on \( ||\varphi'|| \). Very roughly speaking, each quantifier alternation in \( \varphi \) adds one level of a tower to the dependence on \( t \). However, it must be admitted that precise tracking of the running time given by Courcelle’s theorem is generally very difficult and of course depends on the used version of the proof. For these reasons, algorithms given by Courcelle’s theorem are generally regarded as inefficient, and to provide precise bounds on the running time for a particular problem one usually needs to design an explicit dynamic program by hand. However, there are a few attempts of implementing Courcelle’s theorem efficiently [137, 138].

Courcelle’s theorem, as stated in Theorem 2.14, does not imply directly tractability of many important problems. Consider for example the INDEPENDENT SET problem. Clearly, we can express that a graph admits an independent set of size at least \( k \) by an \( \mathbf{MSO}_2 \) formula of length \( O(k^2) \), but this gives us only an algorithm with running time \( f(k, t) \cdot n \), instead of \( f(t) \cdot n \); recall that in this chapter we gave the dynamic programming routine for INDEPENDENT SET working in time \( O(2^t \cdot t^{O(1)} \cdot n) \), which is independent of the target cardinality \( k \). For this reason, the original framework of Courcelle has been extended by Arnborg et al. [7] to include also optimization problems; we remark that a very similar result was obtained independently by Borie et al. [28]. In the framework of Arnborg et al. we are given an \( \mathbf{MSO}_2 \) formula \( \varphi(X_1, X_2, \ldots, X_q) \) with \( X_1, X_2, \ldots, X_q \) being free monadic variables, and a linear combination \( \alpha(|X_1|, |X_2|, \ldots, |X_q|) \) of cardinalities of these sets. The result is that (i) one can in \( f(||\varphi||, t) \cdot n \) time optimize (minimize or maximize) the value of \( \alpha(|X_1|, |X_2|, \ldots, |X_q|) \) for sets \( X_1, X_2, \ldots, X_q \) for which \( \varphi \) is satisfied, (ii) in time \( f(||\varphi||, t) \cdot n^{O(1)} \) ask whether there exist sets \( X_1, X_2, \ldots, X_q \) satisfying \( \varphi \) with a precise value of \( \alpha(|X_1|, |X_2|, \ldots, |X_q|) \). Thus, we can express finding maximum size independent set by taking a constant-size formula \( \varphi(X) \) that checks whether \( X \) is an independent set, and applying the result of Arnborg et al. for maximization.

The natural question to what extent the complexity blow-up in Courcelle’s theorem is necessary has also been intensively studied. Frick and Grohe [108] have shown that, unless \( \mathbf{P}=\mathbf{NP} \), no algorithm with running time \( f(||\varphi||) \cdot n^{O(1)} \) with elementary function \( f \) even for model checking \( \mathbf{MSO}_2 \) on words. Note that we can interpret a word as a path equipped with, for every symbol \( \sigma \) of the alphabet, a unary relation \( U_\sigma \) that is true in vertices corresponding to positions on which \( \sigma \) occurs\(^1\). Thus, model checking \( \mathbf{MSO}_2 \) on paths enriched by unary relation on vertices is not likely to admit FPT model checking algorithms with elementary dependence on the length of the input formula. A different set of lower bounds have been given by Kreutzer and

\(^1\) Formally, a formula of \( \mathbf{MSO}_2 \) on words can also use the order of positions in the word. However, the order relation can be easily expressed in \( \mathbf{MSO}_2 \) using the successor relation.
Tazari [142, 143, 144] under Exponential-Time Hypothesis (see Chapter 8 for ETH). Intuitively, they show that if for a graph class \( C \) the treewidth of graphs from \( C \) cannot be bounded by \( \log^c n \) for some universal constant \( c \), then model checking MSO\(_2\) on this graph class is not likely even to be in XP; we refer to these works for precise statements of the results.

To conclude, all the aforementioned results show that treewidth as a graph parameter corresponds to tractability of the MSO\(_2\) logic on graphs in a well-defined and tight sense. It is notable that this uncovers an elegant link between complexity of graphs as topological structures, and complexity of model checking most natural variants of logic on them.

### 2.5 Computing treewidth

In all applications of the treewidth we discussed in this chapter, we were assuming that a tree decomposition is given as a part of the input. While deciding if an input graph has treewidth at most \( k \) is NP-complete, there are several FPT algorithms computing treewidth. A “simple” argument why \textsc{Treewidth} is FPT follows from deep theory of Robertson and Seymour overviewed in the first chapter. Indeed, it is easy to show that treewidth is “minor-monotone” parameter, i.e. for every minor \( H \) of a graph \( G \), \( \text{tw}(H) \leq \text{tw}(G) \), see Exercise 2.47. Thus the family of graphs of treewidth at most \( k \) is closed under the minor order. Hence, there is a set of \( f(k) \) forbidden minors for this family. Thus for every \( n \)-vertex graph \( G \), by making use of Theorem 1.86 and Proposition 1.87, we can decide in time \( O(f(k)n^3) \) if graph \( G \) is of treewidth at most \( k \). This algorithm is non-constructive because we do not know neither how large this family \( f(k) \) can be, nor how to compute it.

Should running time be \( O(f(k)n^3) \) or there are better recent algorithms?

To decide if the treewidth of \( G \) is at most \( k \) in time \( n^{O(k)} \) one can use graph searching (Theorem 2.7), see Exercise 2.56. In this section we explain a simple FPT-approximation algorithm for \textsc{Treewidth}. Let us remark that there are more efficient algorithms computing the treewidth in the literature. By the celebrated result of Bodlaender [19] the treewidth can be computed in time \( O(2^k n) \). There is a variety of other results in the literature establishing algorithms with various dependences between approximation factor, exponential dependency of \( k \) and polynomial of \( n \), see the Notes section for the references. Thus for many treewidth based applications in parameterized algorithms, we can safely assume that a tree decomposition is provided. The algorithm discussed in this section is based on the following ideas.
• It is well known and easy to prove that every \( n \)-vertex tree \( T \) has a vertex \( v \), such that every tree of \( T - v \) has at most \( \frac{n}{2} \) vertices. Similar fact can be proven for graphs of bounded treewidth. Graphs of treewidth \( k \) have balanced separators of size \( k + 1 \); it is possible to remove few vertices from the graph to leave connected components that are all “significantly” smaller than the original graph.

• We give an FPT algorithm that either finds a balanced separator of small size, or concludes that the treewidth of an input graph is more than \( k \).

• We proceed recursively and either succeed to decompose the graph by separators of small size and construct a tree decomposition of small width, or to prove that the treewidth is more than \( k \).

The main decomposition property exploited by the algorithm is that graphs of treewidth \( k \) have balanced separators of size \( k + 1 \). In particular, for any way to assign non-negative weights to the vertices, there exists a set \( B \) of size at most \( k + 1 \) such that the total weight of any connected component of \( G - B \) is at most half of the total weight of \( G \). While the following separation lemma holds for any non-negative weight function \( w \), for the treewidth algorithm we need only the statement for the weight function assigning weights 0 and 1 to the vertices of the graph.

For a weight function \( w : V(G) \rightarrow \{0, 1\} \), we define the weight of a vertex set to be the sum of the weights of the vertices in the set, namely \( w(S) = \sum_{v \in S} w(v) \).

To verify what is a connected component. Here it is treated as a set of vertices. If it is a subgraph, definition of the weight function \( w \) should be changed. It also propagates in several other places.

Let \( G \) be a graph, \( w : V(G) \rightarrow \{0, 1\} \) and \( \alpha > 0 \). We say that \( B \subseteq V(G) \) is \( \alpha \)-balanced separator, if every connected component of \( G - B \) is of weight at most \( \alpha \cdot w(V(G)) \). If we do not specify function \( w \), we assume that every vertex of \( G \) is of weight 1.

**Lemma 2.15 (Balanced Separators).** For any graph \( G \) of treewidth \( k \) and \( w : V(G) \rightarrow \{0, 1\} \), there is a \( \frac{1}{2} \)-balanced separator of size at most \( k + 1 \). Furthermore, in the case when we are given a tree decomposition \((T, \chi)\) of \( G \) of width \( k \), there is a node \( t \in V(T) \) such that the corresponding bag \( X_t \) is a \( \frac{1}{2} \)-balanced separator and thus such a separator can be found in polynomial time.

**Proof.** Let \((T, \chi)\) be a tree decomposition of \( G \) of width \( k \). We start from selecting an arbitrary node \( r \) of the tree \( T \) and making it a root. For a node \( t \) of \( T \), let \( V_t \) be the set of vertices of \( G \) contained in the bags of the subtree \( T_t \) of \( T \) rooted in \( t \). We select a node \( t \) of \( T \) such that
• $w(V_i) > w(V(G))/2$, and subject to that
• $t$ is at the maximum distance in $T$ from $r$.

We claim that $X_t$ is a $\frac{1}{2}$-balanced separator. Indeed, the weight of vertices from $V(G) \setminus V_t$ is at most $w(V(G))/2$. Let $t_1, \ldots, t_p$ be the children of $t$. Then for every $i \in \{1, \ldots, p\}$, $w(V_{t_i} \setminus X_t) < w(V(G))/2$. By Lemma 2.3, every connected component of $G - X_t$ is either contained in some $V_{t_i}$, or in $V(G) \setminus V_t$.

Finally, when a tree decomposition is given, such a node $t$ can be found in polynomial time.

In Lemma 2.15 to find a balanced separator, we need a tree decomposition of $G$ of width less than $k$. This obstacle can be avoided by making use of the following lemma.

Lemma 2.16. There is an algorithm that for a given $n$-vertex graph $G$, set $W \subseteq V(G)$ and integer $k$, in time $3^{|W|kO(1)} n$
• either outputs a set $X$ of size at most $k + 1$ such that every connected component of $G - X$ has at most $\frac{2}{3}|W|$ vertices which are in $W$, or
• concludes that $\text{tw}(G) > k$.

Proof. For $W \subseteq V(G)$, we define the weight function $w(V(G)) \to \{0, 1\}$ as $w(v) = 1$ if and only if $v \in W$. Then by Lemma 2.15, there exists a set $B$ of size at most $k + 1$ such that every component of $G - B$ has at most $\frac{1}{2}|W|$ vertices which are in $W$. This implies (see Exercise 2.65), that the connected components of $G - B$ can be partitioned into two sets, $L$ and $R$, such that at most $\frac{2}{3}|W|$ vertices of $W$ belong to components from $L$ and at most $\frac{2}{3}|W|$ of $W$ to components from $R$. Let $W_B = W \cap B$ and let $W_L$ and $W_R$ be the vertices of $W$ that are in the components from $L$ and $R$ respectively.

By trying all $3^{|W|}$ partitions of $W$ in three parts the algorithm correctly guesses $W_B$, $W_L$ and $W_R$. Because $B$ separates $W_L$ from $W_R$, we have that the size of a minimum vertex cut between $W_L$ and $W_R$ in $G - W_B$ is at most $|B \setminus W_B| \leq (k + 1) - |W_B|

A minimum vertex cut can be found by an easy reduction to the Maximum Flow problem. We create a network $N$ from $G - W_B$ by putting unit capacities to all edges of $G - W_B$ and by adding a source node $s$ adjacent to all vertices of $W_L$ and the sink node $t$ adjacent to all vertices of $W_R$. See Fig. 2.4. Clearly, every minimum vertex cut between $W_L$ and $W_R$ in $G - W_B$ is a minimum vertex cut in $N$ and vice versa. Thus we find a set $Z$ of size at most $(k + 1) - |W_B|$ that separates $W_L$ from $W_R$ in $G - W_B$.

By the algorithm of Ford and Fulkerson, Maximum Flow is solvable in time $O(|E(G)| \cdot \max |f|)$, where $|f|$ is the value of flow from $s$ to $t$, and since we are only interested in a set $Z$ of size at most $k - |W_B|$, one can run the max-flow algorithm in time $(n + m)kO(1)$. Because the number of edges in a graph of treewidth $k$ is $O(nk)$, see Exercise 2.62, the running time of this step is $nkO(1)$. 

$\Box$
Fig. 2.4: Construction of separator $X = W_R \cup Z$.

Having found $W_L$, $W_R$, $W_B$ and $Z$, we put $X = W_B \cup Z$. Let $C$ be a connected component of $G - X$. Because $C$ cannot contain vertices from both $W_L$ and $W_R$, we have that $|C \cap W| \leq \max\{|W_L|, |W_R|\} \leq \frac{2}{3}|W|$. The total running time of the algorithm is $O(3^{|W|}k^{O(1)}n)$.

If for none of the partitions of $W$ into $W_L, W_R, W_B$ resulted in a cutset $Z$ of size at most $(k + 1) - |W_B|$, by Lemma 2.15, this means that $\text{tw}(G) > k$, which the algorithm reports.

**Theorem 2.17.** There is an algorithm, that for a given $n$-vertex graph $G$ and integer $k$, in time $3^{3k}k^{O(1)}n^2$ either constructs a tree decomposition of $G$ of width at most $4k + 4$, or reports correctly that the treewidth of $G$ is more than $k$.

**Proof.** We give a recursive algorithm solving a more general problem. The algorithm takes as an input $G$, $k$ and $W \subseteq V(G)$ of size at most $3k + 4$, and either concludes that the treewidth of $G$ is larger than $k$ or finds a rooted tree decomposition of width at most $4k + 4$ such that the root bag of the decomposition contains $W$.

On input $(G, W, k)$ the algorithm starts by ensuring that $|W| = 3k + 4$. If $|W| < 3k + 4$, we enhance $W$ by adding arbitrary vertices to $W$ until equality is obtained. Then we apply Lemma 2.16 and in time $3^{|W|}k^{O(1)}n$

- either find a set $X$ of size at most $k + 1$ such that for each connected component $C$ of $G - X$, we have $|C \cap W| \leq \frac{2|W|}{3} = 2(k + 1) + \frac{2}{3}$,
- or conclude that the treewidth of $G$ is more than $k$.

Because $|C \cap W|$ is an integer, we have that $|C \cap W| \leq 2(k + 1)$. Let us note, that if we found such a set $X$, then for every connected component $C$ of $G - X$, we have that $|(W \cap C) \cup X| \leq 3(k + 1)$. For each component $C$ of $G - X$ the algorithm is called recursively on $(G[C \cup X], (W \cap C) \cup X, k)$. If $|C| \leq 3k + 4$, then the tree decomposition of $G[C \cup X]$ consisting of one bag of size $|C \cup X| \leq 4k + 5$ is constructed.
If either of the recursive calls returns that the treewidth is more than $k$, then the treewidth of $G$ is more than $k$ as well. Otherwise for every component $C$, we have constructed a rooted tree decomposition of $G[C \cup X]$ of width at most $4k + 4$ with root bag containing $(W \cap C) \cup X$. To make a tree decomposition of $G$, we make a new root node with bag $X \cup W$, and connect this bag to the roots of the tree decompositions of $G[C \cup X]$ for each component $C$. It is easy to verify that this is indeed a tree decomposition of $G$. The root bag of the constructed tree decomposition contains $W$, and the size of the root bag is $|W \cup X| \leq 4k + 5$. Hence the width of the decomposition is at most $4k + 4$, as claimed.

The running time of the algorithm is proportional to the number of recursive calls multiplied by the time $3^{2\mid W\mid}k^{O(1)}n$ required to accomplish each of the calls. At each recursive calls we basically construct a bag of the tree decomposition, and thus the total number of calls does not exceed the number of bags in the constructed decomposition. When we call on $(G[C \cup X], (W \cap C) \cup X, k)$, we have that $|X| \leq k + 1$ and no connected component of $G - X$ contains more than $2k + 2$ vertices of $W$. Thus the set $(W \cap C) \cup X$ has less than $3k + 4$ vertices and has to be enhanced by adding at least one new vertex from $C$. This means that every bag of the constructed decomposition contains at least one vertex which is not contained in its parent bag. Therefore, the number of bags, and thus the number of recursive calls is $O(n)$ resulting in the total running time $3^{2\mid W\mid}k^{O(1)}n^2 = 3^{3k}k^{O(1)}n^2$.

\[ \square \]

2.6 Applications for planar problems

Most NP-hard graph problems remain NP-hard even when the input graph is restricted to be planar. However, from the approximability or parameterized complexity perspective, many problems become significantly easier on planar graphs. In particular, many parameterized problems on planar graphs can be solved in parameterized subexponential time. Also many problems admit linear kernels and efficient polynomial-time approximation schemes on planar graphs. Often such algorithmic results are based on the bidimensional properties of the problems. In this section, we concentrate only on subexponential algorithms obtained by making use of bidimensionality.

2.6.1 Grid theorems

In the previous sections, we have seen that if the treewidth of a graph is small, then we can find it (or approximate) and then solve many optimization problems efficiently. In this section, we will find out that understanding of what makes the treewidth large, can be also very useful for designing pa-
rameterized algorithms. What is an obstruction for a small treewidth or what structure in a graph forces its treewidth to be large? We already know that if a graph contains a $t$-vertex graph as a minor, then its treewidth should be at least $t - 1$, see Exercise 2.46. However, inverse is not necessarily true, it is possible to show that graphs excluding $K_5$ as a minor can have arbitrarily large treewidth.

Another, extremely useful obstructions to small treewidth, are grid-minors. Let $t$ be a positive integer. The $t \times t$-grid $\mathbb{E}_t$ is a graph with vertex set $\{(x, y) \mid x, y \in \{1, 2, \ldots, t\}\}$. Thus $\mathbb{E}_t$ has exactly $t^2$ vertices. Two different vertices $(x, y)$ and $(x', y')$ are adjacent if and only if $|x - x'| + |y - y'| \leq 1$. See the left graph in Fig. 2.5. The border of $\mathbb{E}_t$ is the set of vertices with coordinates $(1, y), (t, y), (t, 1)$, and $(x, t)$, where $x, y \in \{1, 2, \ldots, t\}$.

It is easy to show that the treewidth of $\mathbb{E}_t$ is at most $t$. For example, $t + 1$ cops can always catch the robber by going from column to column of the grid. It is harder to prove that the treewidth of $\mathbb{E}_t$ is exactly $t$. Here Theorem 2.6 is very handy. The cross $C_{ij}$ of the grid is the union of the vertices of the $i$th column and the $j$th row. The set of crosses of $\mathbb{E}_t$ form a bramble of order $t$, thus the treewidth is at least $t - 1$. With a bit more of work, it is possible to construct a bramble of order $t + 1$ in $\mathbb{E}_t$, and thus to show that $\text{tw}(\mathbb{E}_t) = t$, see Exercise 2.79. Because grids contain no $K_5$ as a minor, we have that $K_5$-minor-free graphs can be of arbitrarily large treewidth.

As we already have seen, if a graph contains large grid as a minor, its treewidth is also large. What is much more surprising, is that the converse is also true, every graph of large treewidth contains a large grid as a minor. This was proved by Robertson and Seymour [181], who gave an exponential in $t$ bound on the treewidth of a graph excluding $\mathbb{E}_t$ as a minor. It was open for many years whether a polynomial relationship could be established between the treewidth of a graph $G$ and the size of its largest grid minor. In 2013 Chekuri and Chuzhoy [37] proved the following Theorem 2.18 establishing the first polynomial dependence of the treewidth and excluded grid.

**Theorem 2.18 (Excluded Grid Theorem [37]).** Let $t \geq 0$ be an integer. There exists a universal constant $c$, such that every graph of treewidth at least $c \cdot t^{99}$ contains $\mathbb{E}_t$ as a minor.
For a planar graph $G$ it is possible to get a much tighter relationship between the treewidth of $G$ and the size of the largest grid minor that $G$ contains. The following theorem is due to Robertson, Seymour and Thomas [183], we use here the refined version of which is due to Gu and Tamaki [112].

**Theorem 2.19 (Planar Excluded Grid Theorem [112, 183]).** Let $t \geq 0$ be an integer. Every planar graph $G$ of treewidth at least $\frac{9}{2} t$, contains $\boxplus_t$ as a minor. Furthermore, there exists a polynomial-time algorithm that for a given planar graph $G$ either outputs a tree decomposition of $G$ of width $\frac{9}{2} t$ or constructs a minor model of $\boxplus_t$ in $G$.

Planar Excluded Grid Theorem immediately implies the following corollary.

**Corollary 2.20.** The treewidth of an $n$-vertex planar graph does not exceed $\frac{9}{2} \sqrt{n} + 1$.

For our purposes, more important is that the Planar Excluded Grid Theorem allows us to identify many parameterized problems on planar graphs with parameter $k$, such that the treewidth of every yes-instance is $O(\sqrt{k})$.

It is also very handy to state one more Excluded Grid Theorem, this time not for minors but for edge contractions. For this we will define a new family of graphs which will play the role of grids. For an integer $t > 0$ the graph $\Gamma_t$ is obtained from the grid $\boxplus_t$ by adding for every $1 \leq x, y \leq t - 1$, the edge $(x, y), (x + 1, y + 1)$, and making the vertex $(t, t)$ adjacent to all vertices with $x \in \{1, t\}$ and $y \in \{1, t\}$. The graph $\Gamma_4$ is the graph in the right part in Fig. 2.5.

**Theorem 2.21.** For any connected planar graph $G$ and integer $t \geq 0$, if $\text{tw}(G) \geq 9(t + 1)$, then $G$ contains $\Gamma_t$ as a contraction. Furthermore there exists a polynomial-time algorithm that given $G$ either outputs a tree decomposition of $G$ of width $9(t + 1)$ or a set of edges whose contraction result in $\Gamma_t$.

**Proof (Sketch of the proof).** By Theorem 2.19, if the treewidth of $G$ is more than $9(t + 1)$, then $G$ contains $\boxplus_{2t+1}$ as a minor. This implies that, after a sequence of vertex/edge removals or contractions, $G$ can be transformed to $\boxplus_{2t+1}$. If we apply only the contractions in this sequence, we end up with some graph $H$, which is a partially triangulated grid, that is a planar graph obtained from grid $\boxplus_{2t+1}$ by adding some edges. We construct $\Gamma_t$ from $H$ by contracting edges as shown in Fig. 2.6.

To redraw Fig. 2.6 or to ask Dimitrios for permission to use the figure from the paper.
Theorem 2.23. Let $G$ be a planar graph, $v \in V(G)$ and $r \geq 1$. Then $\text{tw}(G^r_v) \leq 3(r+1)$. Moreover, a tree decomposition of width at most $3r+1$ can be constructed in polynomial time.

By Theorem 2.23, we have the following corollary.

2.6.2 Shifting technique

Before proceeding with subexponential algorithms, we give a short overview of shifting techniques. This technique is widely used in approximation algorithms for obtaining PTAS on planar graphs. But it can be also useful for parameterized algorithms as well.

For vertex $v$ of a graph $G$ and integer $r \geq 1$, we denote by $G^r_v$ the subgraph of $G$ induced by vertices within distance $r$ from $v$ in $G$. By Theorem 2.21, we have the following corollary.

Corollary 2.22. Let $G$ be a planar graph, $v \in V(G)$ and $r \geq 1$. Then $\text{tw}(G^r_v) \leq 18(r+1)$.

Proof. Let $v$ be a vertex of a planar graph $G$ and $r$ be a positive integer. If $\text{tw}(G^r_v) > 18(r+1)$ then by Theorem 2.21, $G^r_v$ contains $\Gamma_{2r+1}$ as a contraction. It is easy to see that for any vertex of $\Gamma_{2r+1}$, in particular for the vertex to whom $v$ was contracted, there is another vertex of $\Gamma_{2r+1}$ at distance at least $r+1$ from this vertex. Because contraction of edges does not increase the distances between vertices, this implies that there is a vertex of $G^r_v$ at distance at least $r+1$ from $m$, which is a contradiction.

It is possible to prove a better bound on the treewidth dependence from radius in planar graphs. The following result is due to Robertson and Seymour [179], see also the proof of Bodlaender for very similar notion of $k$-outerplanarity [20].

Theorem 2.23 ([179]). Let $G$ be a planar graph, $v \in V(G)$ and $r \geq 1$. Then $\text{tw}(G^r_v) \leq 3r + 1$. Moreover, a tree decomposition of width at most $3r+1$ can be constructed in polynomial time.

Fig. 2.6: The steps of the proof of Theorem 2.21. The two first steps are the boundary contraction of a partial triangulation of $\mathbb{S}_3$. The third step is the contraction to $\Gamma_4$. 

2.6.2 Shifting technique

Before proceeding with subexponential algorithms, we give a short overview of shifting techniques. This technique is widely used in approximation algorithms for obtaining PTAS on planar graphs. But it can be also useful for parameterized algorithms as well.

For vertex $v$ of a graph $G$ and integer $r \geq 1$, we denote by $G^r_v$ the subgraph of $G$ induced by vertices within distance $r$ from $v$ in $G$. By Theorem 2.21, we have the following corollary.

Corollary 2.22. Let $G$ be a planar graph, $v \in V(G)$ and $r \geq 1$. Then $\text{tw}(G^r_v) \leq 18(r+1)$.

Proof. Let $v$ be a vertex of a planar graph $G$ and $r$ be a positive integer. If $\text{tw}(G^r_v) > 18(r+1)$ then by Theorem 2.21, $G^r_v$ contains $\Gamma_{2r+1}$ as a contraction. It is easy to see that for any vertex of $\Gamma_{2r+1}$, in particular for the vertex to whom $v$ was contracted, there is another vertex of $\Gamma_{2r+1}$ at distance at least $r+1$ from this vertex. Because contraction of edges does not increase the distances between vertices, this implies that there is a vertex of $G^r_v$ at distance at least $r+1$ from $m$, which is a contradiction.

It is possible to prove a better bound on the treewidth dependence from radius in planar graphs. The following result is due to Robertson and Seymour [179], see also the proof of Bodlaender for very similar notion of $k$-outerplanarity [20].

Theorem 2.23 ([179]). Let $G$ be a planar graph, $v \in V(G)$ and $r \geq 1$. Then $\text{tw}(G^r_v) \leq 3r + 1$. Moreover, a tree decomposition of width at most $3r+1$ can be constructed in polynomial time.

By Theorem 2.23, we have the following corollary.
Corollary 2.24. Let \( v \) be a vertex of a planar graph \( G \) and let \( L_i \) be the vertices of \( G \) at distance \( i \), \( 0 \leq i \leq n \), from \( v \). Then for any \( i, j \geq 0 \), the treewidth of the subgraph \( G_{i,i+j} \) induced by vertices in \( L_i \cup L_{i+1} \cup \cdots \cup L_{i+j} \) does not exceed \( 3j + 1 \).

Proof. Let us note that graph \( G_{i,i+j} \) If we contract all edges of \( G \) with endpoints in vertices at distance less than \( i \) and delete all vertices from at distance more than \( i+j \) from \( v \), then the obtained graph \( H \) is planar, it contains \( G_{i,i+j} \) as a minor and in this graph all vertices are within distance \( j \) from \( v \). By Theorem 2.23, this yields that the treewidth of \( H \), and hence of \( G_{i,i+j} \), is at most \( 3j + 1 \).

The idea behind the shifting technique is as follows.

- Pick a vertex \( v \) of planar graph \( G \) and run breadth-first search (BFS) from \( v \).
- By Corollary 2.24, for any \( i, j \geq 0 \), the treewidth of the subgraph \( G_{i,i+j} \) induced by vertices in levels \( i, i+1, \ldots, i+j \) of BFS does not exceed \( 3j + 1 \).
- Now for an appropriate choice of parameters, we can find a “shift” of “windows”, i.e. a disjoint set of a small number of consecutive levels of BFS, “covering” the solution. Because every window is of small treewidth, we can employ the dynamic programing or the power of Courcelle’s theorem to solve the problem.

We give two examples of the shifting technique. Our first example is SUBGRAPH ISOMORPHISM. Let us remind, that here we are given two graphs, host graph \( G \) and pattern graph \( H \) and ask if \( H \) is isomorphic to a subgraph of \( G \). Since CLIQUE is a special case of SUBGRAPH ISOMORPHISM, the problem is \( W[1] \)-hard on general graphs being parameterized by the size of the pattern graph. It is possible to show by making use of Color Coding that the problem is solvable in time \( f(H)|V(G)|^{tw(H)} \) for some function \( f \), see Exercise 2.68. However, we show that on planar graphs this is FPT.

As a building block here we need the following lemma. The lemma can be proved either by designing dynamic programming algorithm or by making use of Courcelle’s theorem. We leave its proof as an exercise, see Exercise 2.69.

Lemma 2.25. SUBGRAPH ISOMORPHISM is FPT parameterized by \( tw(G) + |H| \).

In other words, there is an algorithm solving SUBGRAPH ISOMORPHISM in time \( f(|H|, tw(G))n^{O(1)} \) for some function \( f \). Let us remark that parameterization \( tw(G) + |H| \) is necessary, parameterization just by the treewidth will not work. It is known that SUBGRAPH ISOMORPHISM is NP-hard on forests, i.e. graphs of treewidth one, see Notes for the references.
Let $G$ be a planar host graph and $H$ be the pattern graph. We select a vertex $v$ of $G$ and run BFS from $v$. This part is clearly done in polynomial time. Let $k = |V(H)|$. For every fixed $j \in \{0, \ldots, \lfloor n/(k+1) \rfloor \}$, we construct graph $G^j$ by deleting vertices of BFS-levels $i(k+1)+j$ for all $i \in \{0, \ldots, k\}$. Thus for each $j$, after removing these levels only "windows" of $k$ consecutive levels are left. By the arguments in the grey box, every subgraph $G_{i,i+k}$ of $G$ induced by $k$ consecutive levels of BFS is of treewidth at most $3k + 1$. The treewidth of $G^j$ does not exceed the treewidth of its connected components, thus the treewidth of $G^j$ is at most $3k + 1$. By Lemma 2.25, on $G^j$, we can solve Subgraph Isomorphism in time $f(k) \cdot n^{O(1)}$. We do this for every $0 \leq j < k + 1$. For every $k$-vertex subset $X$ of $G$, for some choice of $j$, $X \cap V(G^j) = \emptyset$. Therefore, if $G$ contains $H$ as a subgraph, then for at least one value of $j$, $G^j$ also contains $H$. It means that we find a copy of $H$ in $G$ if there is one and thus prove the following theorem.

**Theorem 2.26.** Subgraph Isomorphism on planar graphs is FPT parameterized by $|V(H)|$.

The running time of the algorithm in Theorem 2.26 depends on how fast we can solve Subgraph Isomorphism on graphs of bounded treewidth. See the Notes for the current state of the art on the running time for this problem.

Our second example concerns the Bisection problem. For a given $n$-vertex graph $G$, weight function $w : V(G) \to \mathbb{N}$ and integer $k$, the task is to decide if there is a partition of $V(G)$ into sets $V_1$ and $V_2$ of weights $\lfloor w(V(G))/2 \rfloor$ and $\lfloor w(V(G))/2 \rfloor$ and such that the number of edges between $V_1$ and $V_2$ is at most $k$. In other words, we are looking for a balanced partition $(V_1, V_2)$ with a $(V_1, V_2)$-cut at most $k$.

As a building block for FPT algorithm on planar graphs, we need the following lemma, which proof is again left as an exercise, see Exercise 2.70.

**Lemma 2.27.** Bisection is solvable in time $2^t \cdot n^{O(1)}$ on an $n$-vertex given together with its tree decomposition of width $t$.

**Theorem 2.28.** Bisection on planar graphs is solvable in time $2^{O(k)} \cdot n^{O(1)}$.

**Proof.** As for Subgraph Isomorphism, we select a vertex $v$ of planar graph $G$ and run BFS from $v$. Because we are looking for a bisection with at most $k$ edges and thus with at most $2k$ vertices incident with these edges, we must take larger windows such that for some of the shift, all vertices incident with the edges of the cut of a solution, are covered by windows. More precisely, if there is a solution to Bisection, then for some $j \in \{0, \ldots, \lfloor n/(2k+1) \rfloor \}$ the vertices incident with the edges of the cut corresponding to the solution, intersect none of the levels $i(2k+1)+j$ for all $i \in \{0, \ldots, 2k\}$. We construct graph $G^j$ as follows. For each level $p = i(2k+1)+j$, we replace all vertices of this level by one vertex $v_p$ (and keeping the new vertex adjacent to all neighbors of the level) and put $w(v_p)$ to be equal to the sum of the weight of vertices from the $p$th level, see Fig. 2.7. By Exercise 2.48, the treewidth of $G^j$...
Fig. 2.7: Replacing vertices at level $p$ with one vertex $v_p$.

does not exceed the maximum treewidth of its 2-vertex connected component, which in turn, does not exceed $6k + 1$.

There is a solution to Bisection in $G$ if and only if there is a solution to Bisection in $G^j$ for some $j$. By applying Lemma 2.27 for every $j$, we solve Bisection in time $2^{O(k)} \cdot n^{O(1)}$.

Let us remark that the only property of planarity for shifting technique we used, was Theorem 2.23 and the property that planar graphs are closed under operation of taking minors. (This was used in the grey box when we contracted all edges of $G$ with endpoints in levels of BFS smaller than $i$.) We say that the class of graphs is of bounded local treewidth if there is function $f$, such that for every graph $G$, for every vertex $v$ of $G$, $\text{tw}(G_v) \leq f(r)$. For example, planar graphs are of bounded local treewidth. Graphs of vertex degree bounded by $d$ are also of bounded local treewidth because for every $v$, $G_v^r$ contains at most $d^r$ vertices, and thus $\text{tw}(G_v^r) \leq d^r$. However, this class Our arguments used for planar graphs are trivially extendable to minor-closed classes of graphs of bounded local treewidth. It was shown by Eppstein [91] that a minor-closed class of graphs is of locally bounded treewidth if and only if it exclude some apex graph as a minor. An apex is a graph obtained from a planar graph $G$ by adding one vertex and making it adjacent to some vertices of $G$. Then a class of graphs is apex-minor-free if every graph in this class does not contain some fixed apex graph as a minor. Demaine and Hajiaghayi [73] refined the result of Eppstein by showing that function $f$ is linear.

2.6.3 Bidimensionality

Planar Excluded Grid Theorem provides a powerful tool for designing algorithms in planar graphs. In all these algorithms we use the following win/win approach. We compute the treewidth of a given planar graph and if the treewidth is small, we use dynamic programing to find a solution. Otherwise, we know that our graph should contain a large grid as a minor. Then we are able to conclude with the right answer and thus we win in both cases.
Let us give first a few examples of this strategy. In all these examples by making use of Planar Excluded Grid Theorem, we obtain algorithms with running time \textit{subexponential} in the parameter.

Let $G$ be a planar graph and we want to solve \textsc{Vertex Cover}, i.e. to decide for a given parameter $k$, if the vertex cover of $G$ is at most $k$. One way to obtain a subexponential $2^{O(\sqrt{k})} \cdot n^{O(1)}$ algorithm is to find a planar kernel with $O(k)$ vertices. Because the kernel is planar, its treewidth is $O(\sqrt{k})$. Then we can use dynamic programming on graphs of bounded treewidth to obtain the desired solution. However, here we want to explain a different algorithm for \textsc{Vertex Cover}, which will be easily extended for many other problems.

To give an algorithm for \textsc{Vertex Cover}, we need to answers the following three simple questions.

(i) How large can be vertex cover of $t$? It is easy to check that $t$ contains a matching of size $t^2/2$, and thus every vertex cover of $t$ is of cardinality at least $t^2/2$.

(ii) Given a tree decomposition of width $t$ of $G$, how fast we can solve \textsc{Vertex Cover}? By Corollary 2.10, this can be done in time $2^t \cdot t^{O(1)} \cdot n$.

(iii) Is \textsc{Vertex Cover} minor-closed? In other words, is it true that for every minor $H$ of graph $G$, the vertex cover of $H$ does not exceed the vertex cover of $G$?

It is easy to see, see Exercise 2.72 that if a graph $G$ has a vertex cover of size at most $k$, then the same holds for every minor of $G$. Thus if $G$ contains $t$ for $t \geq \sqrt{2k + 1}$, as a minor, then by (i), $G$ has no vertex cover of size $k$. By Planar Excluded Grid Theorem, this means that the treewidth of a planar graph with vertex cover of cardinality $k$ does not exceed $9/2 \sqrt{2k + 1}$. We summarize the above discussions with the following algorithm. For $t = \sqrt{2k + 1}$, by making use if Theorem 2.19, we either compute in polynomial time a tree decomposition of width at most $9/2$ or conclude that $G$ has $t$ as a minor. In the second case, $G$ has no vertex cover of size $k$. If $\text{tw}(G) \leq 9/2 \sqrt{2k + 1}$, then by (ii), we can solve \textsc{Vertex Cover} in time $2^{9/2 \sqrt{2k + 1}} \cdot k^{O(1)} \cdot n = 2^{O(\sqrt{k})} \cdot n + n^{O(1)}$. The total running time of the algorithm is $2^{O(\sqrt{k})} \cdot n + n^{O(1)}$.

It is instructive to extract the properties of \textsc{Vertex Cover} which were essential for the subexponential algorithm.

\begin{itemize}
  \item[(P1)] The size of any solution in $t$ is of order $\Omega(t^2)$.
  \item[(P2)] On graphs of treewidth $t$, the problem is solvable in time $2^{O(t)} \cdot n^{O(1)}$.
  \item[(P3)] The problem is minor-closed, i.e. if $G$ has a solution of size $k$, then every minor of $G$ also has a solution of size $k$.
\end{itemize}

Can the same arguments work for \textsc{Independent Set}? Of course yes, the only difference with \textsc{Vertex Cover} is that if the treewidth of $G$ is larger
than a certain threshold, then we can immediately conclude that \((G, k)\) is a yes-instance of **Independent Set**. Thus these three properties of parameterized problems are sufficient conditions for solvability in subexponential time in planar graphs. Similarly, one can provide subexponential algorithms for **Feedback Vertex Set** or \(k\)-**Path**. But before we formalize this observation, let us consider a few more examples.

Our next problem is **Dominating Set**. It is easy to check that the problem satisfies \((P1)\) and we also proved that it satisfies \((P2)\). However, \((P3)\) does not hold. For example, to dominate vertices of a \(3\ell\)-vertex cycle \(C\) we need \(\ell\) vertices. But if we add to \(C\) a vertex adjacent to all vertices of \(C\), then the new graph contains \(C\) as a minor but is dominated by one vertex. Thus the approach that worked for **Vertex Cover**, does not work for **Dominating Set**. On the other hand, while not being closed under taking of minors, domination is closed under edge-contraction. That is, if \(G\) can be dominated by \(k\) vertices, then any graph obtained from \(G\) by contracting some of the edges can be also dominated by \(k\) vertices. What we need here is the structure similar to grid to which every planar graph of large treewidth can be contracted. But this is exactly what Theorem 2.21 provides us!

So what is the domination number of \(\Gamma_t\)? We do not know the exact formula (and this can be a difficult question to answer) but a lower bound is easy. Every vertex except the the right-lower corner vertex \((t, 1)\) can dominate at most 9 vertices. The right-lower corner vertex dominates only 4 vertices. Thus every dominating set of \(\Gamma_t\) is of size at least \(t^2 - 4t + 4 = \Omega(t^2)\). Therefore every connected planar graph \(G\) with dominating set of size at most \(k\) has treewidth \(O(\sqrt{k})\). Combining with dynamic programing, we succeed to solve the problem in time \(2^{O(\sqrt{k})} \cdot n^{O(1)}\). And again, the essential properties of the problem we used were (i) the solution is of order \(\Omega(k^2)\) in \(\Gamma_k\), (ii) the problem is closed under edge contraction; and (iii) the problem is solvable in single-exponential time parameterized by the treewidth of the input graph.

Let us try to formalize the discussions we had so far. We restrict our attention to vertex-subset problems. Edge-subset problems can be defined similarly and same arguments will work for them, we do not discuss them here. Let \(\phi\) be a computable function which takes as an input graph \(G\), a set \(S \subseteq V(G)\) and outputs \texttt{true} or \texttt{false}. The interpretation of \(\phi\) is that it defines the space of feasible solutions \(S\) for a graph \(G\) by returning whether \(S\) is feasible for \(G\). For an example, for the **Dominating Set** problem we have that \(\phi(G, S) = \texttt{true}\) if and only if \(N[S] = V(G)\) and for **Independent Set**, we have that \(\phi(G, S) = \texttt{true}\) if and only if no two vertices \(u, v\) from \(S\) are adjacent.

For function \(\phi\), we define **vertex-subset problem II** as a parameterized problem, where input is a graph \(G\) and an integer \(k\), the parameter is \(k\).
2.6 Applications for planar problems

For function $\phi$, every vertex-subset problem is either maximization or minimization. For maximization problem, the task is to decide whether there is a set $S \subseteq V(G)$ such that $|S| \geq k$ and $\phi(G, S) = \text{true}$. Similarly, for minimization problem, we are looking for a set $S \subseteq V(G)$ such that $|S| \leq k$ $\phi(G, S) = \text{true}$.

Thus, for an instance $(G, k)$ of a minimization (or maximization) problem $\psi$, we have $(G, k) \in \psi$, or in other words, $(G, k)$ is a yes-instance of $\psi$ if and only if there is $S \subseteq V(G)$ such that $\phi(G, S) = \text{true}$ and $|S| \leq k$ (or $|S| \geq k$ for maximization).

We note that the definition of vertex-subset problems captures many problems which, at a first glance, don’t look as if they could be captured by this definition. An example is the Cycle Packing problem. Here input is a graph $G$ and integer $k$, and the task is to determine whether there exist $k$ cycles $C_1, C_2, \ldots, C_k$ in $G$ that are pairwise vertex disjoint. This is a vertex-subset problem because $G$ has $k$ disjoint cycles if and only if there exists a set $S \subseteq V(G)$ of size at least $k$ and $\phi(G, S)$ is true, where $\phi(G, S) = \text{true}$ if and only if there is a subgraph $G'$ of $G$ such that each connected component of $G'$ is a cycle and each connected component of $G'$ contains exactly one vertex of $S$. It is possible to show that in this case checking whether $\phi(G, S)$ is true for a given graph $G$ and set $S$ is NP-complete. But since we require from $\phi$ only to be computable, this definition nevertheless shows that Cycle Packing is a vertex-subset problem. Another example is $k$-Path, where we are asked for a given graph $G$ and integer $k$, whether $G$ contains a path of length $k$. Of course, $k$-Path is much more natural to view as an edge subset problem, but it is also a vertex-subset problem, where $\phi(G, S)$ is defined to be true if and only if $G$ contains a path spanning all vertices of $S$.

For any vertex subset minimization problem $\psi$, we have that $(G, k) \in \psi$ implies that $(G, k') \in \psi$ for all $k' \geq k$. Similarly, for maximization problem, we have that $(G, k) \in \psi$ implies that $(G, k') \in \psi$ for all $k' \geq k$. Thus the notion of “optimality” is well defined for vertex subset problems.

**Definition 2.29.** For a vertex-subset minimization problem $\psi$,

$$OPT_\psi(G) = \min\{k \mid (G, k) \in \psi\}.$$

If there is no $k$ such that $(G, k) \in \psi$, we put $OPT_\psi(G) = +\infty$.

For a vertex-subset maximization problem $\psi$,

$$OPT_\psi(G) = \max\{k \mid (G, k) \in \psi\}.$$

If no $k$ such that $(G, k) \in \psi$ exists, then $OPT_\psi(G) = -\infty$.

We say that a vertex-subset problem $\psi$ is **contraction-closed** if for every $G$ and $uv \in E(G)$, $OPT_\psi(G/uv) \leq OPT_\psi(G)$.

We are now ready to define bidimensional problems.
Definition 2.30 (Bidimensional problem). A vertex subset problem $\Pi$ is bidimensional if it is contraction-closed, and there exists a constant $c > 0$ such that $OPT_\Pi(I_k) \geq ck^2$.

It is usually easy to determine whether a problem is bidimensional. Take for an example Vertex Cover. Contracting or deleting an edge does not increase the size of the minimum vertex cover, so the problem is contraction-closed. And as we already observed, $OPT_\Pi(k) \leq k^2/2$. Thus $OPT_\Pi(I_k) \geq k^2/2$ and Vertex Cover is bidimensional. Similarly, Feedback Vertex Set, Induced Matching, Cycle Packing, Scattered Set for fixed value of $d$, $k$-Path, Dominating Set, and $r$-Center are bidimensional, see Exercise 2.74.

The crucial properties used to obtain subexponential algorithms in our examples was the sublinear bound of the parameter by the treewidth. This is exactly what makes bidimensionality useful for algorithmic applications.

Lemma 2.31 (Parameter-Treewidth Bound). Let $\Pi$ be a bidimensional problem. Then there exists a constant $\alpha_\Pi$ such that for any connected planar graph $G$, $tw(G) \leq \alpha_\Pi \cdot \sqrt{OPT_\Pi(G)}$. Furthermore, there exists a polynomial time algorithm that for a given $G$ constructs a tree decomposition of $G$ of width at most $\alpha_\Pi \cdot \sqrt{OPT_\Pi(G)}$.

Proof. Consider a bidimensional problem $\Pi$, and a connected planar $G$. Let $t$ be the maximum integer such that $G$ contains $I_t$ as a contraction. Since $\Pi$ is contraction-closed and bidimensional, it follows that $OPT_\Pi(G) \geq OPT_\Pi(I_t) \geq ct^2$. By Theorem 2.21, the treewidth of $G$ is at most $9t$, and a tree decomposition of this width can be found in polynomial time. Thus $OPT_\Pi(G) \geq c (\frac{tw(G)^2}{9^2})$ and the lemma follows for $\alpha_\Pi = 9/\sqrt{c}$.

The next theorem follows almost directly from Lemma 2.31.

Theorem 2.32. Let $\Pi$ be a bidimensional problem such that there exists an algorithm for $\Pi$ with running time $2^{O(1)}n^{O(1)}$ when a tree decomposition of the input graph $G$ of width $t$ is given. Then $\Pi$ is solvable in time $2^{O(\sqrt{t})}n^{O(1)}$ on connected planar graphs.

Proof. The proof of the theorem is identical to the algorithms we described for Vertex Cover and Dominating Set. By Lemma 2.31, for every bidimensional problem $\Pi$, there exists a constant $\alpha_\Pi$ such that for any connected planar graph $G$, $tw(G) \leq \alpha_\Pi \cdot \sqrt{OPT_\Pi(G)}$ and a tree decomposition if width $\alpha_\Pi \cdot \sqrt{OPT_\Pi(G)}$ can be computed in polynomial time. Thus for an instance $(G, k)$ of $\Pi$, where $G$ is a connected planar graph, if $tw(G) \leq \alpha_\Pi \cdot \sqrt{k}$, we use time $2^{O(\sqrt{t})}n^{O(1)}$ algorithm to solve the problem. Otherwise $(G, k)$ is a no-instance if $\Pi$ is a minimization vertex-subset problem or a yes-instance if $\Pi$ is a maximization problem.
Let us remark that the requirement of connectivity of $G$ in Theorem 2.32 is necessary. In Exercise 2.77, we give an example of bidimensional problem $\Pi$ such that Lemma 2.31 does not hold for $\Pi$. On the other hand, for most of the natural problems, in particular problems listed in Corollaries 2.33 and 2.34, we do not need the input graph $G$ to be connected. This is due to the fact that each of these problems is monotone subject to removal of connected components. In other words, for $\Pi$ and for every connected component $C$ of $G$, we have that $OPT_\Pi(G - C) \leq OPT_\Pi(G)$.

By combining the bidimensionality of many problems solvable in single-exponential time parameterized by the treewidth, see Exercise 2.73, we obtain the following corollary of Theorem 2.32.

**Corollary 2.33.** On planar graphs

- Vertex Cover,
- Independent Set,
- Dominating Set,
- $r$-Center for fixed $r$,
- Induced Matching, and
- Scattered Set for fixed $d$

are solvable in time $2^{O(\sqrt{k})}n^{O(1)}$.

Similarly, by combining Lemma 2.31 with Theorem 2.13, we obtain the following corollary.

**Corollary 2.34.** On planar graphs

- Steiner Tree parameterized by the size of the tree,
- Feedback Vertex Set,
- Connected Dominating Set,
- $k$-Path,
- Cycle Packing,
- Connected Vertex Cover,
- Connected Feedback Vertex Set, and
- ADD YOUR FAVORITE PROBLEM HERE

are solvable in time $k^{O(\sqrt{k})}n^{O(1)}$.

By making use of “clever” dynamic programming techniques from Chapter 5, for almost each of these problems, except Cycle Packing, the running time can be improved to $2^{O(\sqrt{k})}n^{O(1)}$.

Let us briefly mention possible extensions of bidimensionality for more general classes of graphs. Planar Excluded Grid Theorem can be generalized to graphs excluding some fixed graph $H$ as a minor, i.e. $H$-minor-free graphs. Demaine and Hajiaghayi proved that $[73]$ for every integer $t > 0$, every $H$-minor-free graph $G$ of treewidth more than $\alpha_H t$, where $\alpha_H$ is a constant depending on $H$, contains $I_t$ as a minor. Using this, it is possible
to show that the treewidth-parameter bound $tw(G) \leq \alpha_H \cdot \sqrt{OPT_H(G)}$ holds for much more general classes of apex-minor-free graphs. An apex is a graph obtained from a planar graph $G$ by adding one vertex and making it adjacent to some vertices of $G$. Then a class of graphs is apex-minor-free if every graph in this class does not contain some fixed apex graph as a minor. Thus for example, the bidimensional arguments imply a subexponential algorithm for **Dominating Set** on apex-minor-free graphs, but do not imply imply such an algorithm for minor-free graphs. While **Dominating Set** is solvable in subexponential time on minor-free graphs, the solution requires additional ideas. If we relax the notion of bidimensionality to minor-bidimensionality for problems closed under operations of taking minors, then for minor-bidimensional problems, the treewidth-parameter bound holds for $H$-minor-free graphs. For example **Vertex Cover** or **Feedback Vertex Set** admit subexponential algorithms on minor-free graphs because they are minor-bidimensional.

### 2.7 Irrelevant vertex technique

In the **Planar Deletion** problem we are given graph $G$ and integer $k$. The question is whether there is a deletion vertex set $D$ of size at most $k$ such that $G - D$ is planar.

If a graph $G$ can be turned into a planar graph by deleting at most $k$ vertices, then all its minors have the same property. Thus by the theorem of Robertson and Seymour (Theorem 1.86) for every fixed $k$, the problem can be characterized by a finite set of forbidden minors and thus is FPT. Since we do not know how many are in the set of forbidden minors, these arguments are not constructive. In this section we give a constructive FPT algorithm for **Planar Deletion**, but most interesting than the algorithm is the technique to obtain it. The algorithm is based on the *irrelevant vertex technique*.

This technique originates from the work of Robertson and Seymour for their famous FPT algorithm for the **Vertex Disjoint Paths** problem. For a graph $G$ given with a set of $k$ pairs of terminal vertices, the task is to find $k$ vertex disjoint paths connecting all pairs of terminals. The algorithm of Robertson and Seymour is very complicated and is using very deep structural theorems from Graph Minors. On a very general level, the algorithm can be roughly summarized as follows. As long as the treewidth of graph $G$ is large, then it is possible to find a vertex $v$ that is solution-irrelevant: every collection of $k$ of any solution can be rerouted to an equivalent one avoiding $v$. Thus $v$ can be safely removed. By repeating these arguments, we eventually reduce the treewidth of the graph and will be able to solve the problem by dynamic programming. Since the work of Robertson and Seymour, the irrelevant vertex technique was used in many parameterized algorithms. While the
general win/win (either the treewidth is small, or we can reduce the input) approach is easy, the devil is in details, and most of the algorithms based on irrelevant edge/vertex techniques are very non-trivial and technical. However, for Planar Deletion the proofs are much easier, and it is possible to give almost complete proof in one section. We prove the following theorem.

**Theorem 2.35.** Planar Deletion is solvable in time $2^O(k^2 \log k)n^{O(1)}$.

The remaining part of this section is devoted to the proof of the theorem. We will implement the following plan.

- We use iterative compression. For a given graph $G$ and set $D$ of size $k+1$ such that $G - D$ is planar, we ask if there is a set $D'$ of size $k$ such that $G - D'$ is planar. The only “black box” which we do not explain is the fact that on graphs of treewidth $t$, one can solve the iterative compression variant of Planar Deletion in time $2^{O(t \log t)}n$.
- If the treewidth of $G - D'$ is smaller than some threshold, we use the black box to solve the problem. Otherwise, $G - D'$ contains a large grid $H$ as a minor.
- If the grid is sufficiently large, we show that every yes-instance contains $k + 2$ “concentric” cycles “centered” in a vertex $v$, such that the outer cycle $X$ separates all the cycles from the remaining graph and the connected component $G - X$ containing all these cycles is planar. Then (this is the most difficult part of the proof), $v$ is irrelevant and thus can be safely deleted.

**Planarity definition, Kuratowski theorem in Graph Appendix?**

We start from the following criterion of planarity. Let $X$ be a cycle in graph $G$. An $X$-bridge in $G$ is a subgraph of $G$ which is either a chord of $X$, or a connected component $B$ of $G - V(X)$ together with all edges between $B$ and $X$, and their endpoints. If $B$ is a $X$-bridge, then the vertices $V(B) \cap V(X)$ are the attachments of $B$. Two $X$-bridges $B_1, B_2$ overlap if at least one of the following conditions is satisfied:

- $B_1$ and $B_2$ have at least three attachments in common, or
- the cycle $X$ contains distinct vertices $a, b, c, d$ (in this cyclic order) such that $a$ and $c$ are attachments of $B_1$, while $b$ and $d$ are attachments of $B_2$.

For a graph $G$ with a cycle $X$, the corresponding overlap graph $O(G, X)$ has the $X$-bridges in $G$ as its vertices, with an edge between two bridges if they overlap. The following characterization of planar graphs will be essential to our arguments.

Any planar drawing of $X$ in the plane partitions the rest of the plane into two disjoint open sets, the faces of $X$. Let us note that if two $X$-bridges $B_1$
and \( B_2 \) are planar, then they overlap if and only if they cannot be drawn without intersection in one face of \( X \).

The following criterion of planarity is well known. We leave the proof of it as Exercise 2.78.

**Lemma 2.36.** Graph \( G \) is planar if and only if it has a cycle \( X \) such that for every \( X \)-bridge \( B \), graph \( X \cup B \) is planar and the overlap graph \( O(G,X) \) is bipartite.

Let \( G \) be a connected graph, \( u,v \in V(G), u \neq v \). Let \( X \) be a \((u,v)\)-separator, we assume that \( u,v \notin X \). The connected component \( C \) of \( G - X \) containing \( v \) is referred to as the \( v \)-component. We say that \( X \) is a **connected separator** if \( G[X] \) is connected and **cyclic** if \( G[X] \) contains a Hamiltonian cycle.

**Lemma 2.37.** Let \( G \) be a connected graph, \( u,v \in V(G), u \neq v \). Let \( X_1 \) and \( X_2 \) be disjoint connected \((u,v)\)-separators and let \( C_i \) be the \( v \)-component of \( G - X_i, i = 1,2 \). Then either \( C_1 \cup X_1 \subseteq C_2 \) or \( C_2 \cup X_2 \subseteq C_1 \).

**Proof.** Let \( P \) be a path in \( G \) from \( v \) to \( u \). This path should meet a vertex from \( X_1 \) and a vertex from \( X_2 \). Let \( x \) be the first vertex of \( X_1 \cup X_2 \) in \( P \). If \( x \in X_1 \) then the last vertex of \( X_1 \cup X_2 \) in \( P \) should be from \( X_2 \). If it was not the case, because \( X_1 \) is connected and disjoint from \( X_2 \), then there would be an \((v,u)\)-path avoiding \( X_2 \). Similarly, if \( x \in X_2 \), then the last vertex of \( X_1 \cup X_2 \) in \( P \) is from \( X_1 \). Without loss of generality, let us assume that \( x \in X_1 \). Thus the last vertex of \( P \) is from \( X_2 \). Then in every path \( P' \) from \( v \) to \( u \) the first vertex of \( X_1 \cup X_2 \) hit by \( P' \) is also from \( X_1 \). Indeed, if \( P' \) hits a vertex from \( X_2 \) first, because \( X_2 \) is connected and the last vertex of \( P \) from \( X_1 \cup X_2 \) is a vertex of \( X_2 \), we would be able to reach \( u \) from \( v \) avoiding \( X_1 \). Thus \( C_1 \cup X_1 \subseteq C_2 \). \( \Box \)

**Definition 2.38 (Irrelevant Vertex).** Let \((G,k)\) be an instance of PLANAR DELETION. Vertex \( v \) of graph \( G \) is irrelevant **irrelevant vertex** if for every set \( D \) of size at most \( k \), \( G - D \) is planar if and only if \( G - (D \cup \{v\}) \) is planar.

Thus \( v \) is irrelevant vertex, then \((G,k)\) is a yes-instance if and only if \((G-\{v\},k)\).

The following lemma establishes a criterion of for being an irrelevant vertex.

**Definition 2.39 (Irrelevant Vertex).** Vertex \( v \) is **irrelevant** if there is a vertex \( u \neq v \) and vertex disjoint cyclic \((v,u)\)-separators \( X_1, X_2, \ldots, X_{k+2} \) such that

- for every \( i \in \{1,\ldots,k+1\} \), \( X_i \) is a subset of the \( v \)-component of \( G - X_{i+1} \), and
- the \( v \)-component of \( G - X_{k+2} \) is planar.

**Lemma 2.40.** Let \( v \) be a vertex of graph \( G \) such that there is a vertex \( u \neq v \) and vertex disjoint cyclic \((v,u)\)-separators \( X_1, X_2, \ldots, X_{k+2} \) such that
2.7 Irrelevant vertex technique

Fig. 2.8: Vertex \( v \) separated by \( X_i \) and \( X_j \).

• for every \( i \in \{1, \ldots, k+1\} \), \( X_i \) is a subset of the \( v \)-component of \( G - X_{i+1} \), and
• the \( v \)-component of \( G - X_{k+2} \) is planar.

Then \( v \) is irrelevant vertex of graph \( G \).

Proof. Let \( v \) be a vertex of graph \( G \) satisfying the conditions of the lemma. We show that for every set \( D \) of size at most \( k \), \( G - D \) is planar if and only if \( G - (D \cup \{v\}) \) is planar.

If \( G - D \) is planar then clearly \( G - (D \cup \{v\}) \) is planar. Suppose that \( G - (D \cup \{v\}) \) is planar. By the definition of an irrelevant vertex, there is \( u \neq v \) and vertex disjoint cyclic \((u,v)\)-separators \( X_1, X_2, \ldots, X_{k+2} \) such that for every \( i \in \{1, \ldots, k+1\} \), \( X_i \) is a subset of the \( v \)-component of \( G - X_{i+1} \), and the \( v \)-component of \( G - X_{k+2} \) is planar. Let us note that by Lemma 2.37, this implies that for every \( i \in \{1, \ldots, k+2\} \) the \( v \)-component of \( G - X_i \) is planar.

Because \( |D| \leq k \), at least two separators \( X_i \) and \( X_j \) in \( G \) contain no vertex from \( D \). Let \( i < j \). By Lemma 2.37, \( X_i \) is contained in the \( v \)-component of \( G - X_j \) in \( G \).

Let \( X \) be a Hamiltonian cycle in \( G[X_j] \). We want to use the planarity criterion (Lemma 2.36) applied for cycle \( X \) and graph \( G - D \). First of all, every \( X \)-bridge in \( G - D \) not containing \( v \) is also a bridge in \( G - (D \cup \{v\}) \) and thus is planar. The \( X \)-bridge in \( G - D \) containing \( v \), is contained in the \( v \)-component of \( G - X_j \) and thus is also planar.

Now for the overlap graph \( O(G - D, X) \). Let \( B \) be the \( X \)-bridge of \( G - D \) containing \( v \). Then \( B = B_1 \cup \cdots \cup B_t \cup \{v\} \), where \( B_i \) are some \( X \)-bridges of \( G - (D \cup \{v\}) \). But because \( X_i \) is connected separator containing no vertex of \( D \), it is also a connected separator in \( G - (D \cup \{v\}) \). Then either \( B \) has
no attachments in \( X \) or it has exactly the same attachment as the \( X \)-bridge in \( G - (D \cup \{v\}) \) containing \( X_i \), see Fig. 2.8. Thus graphs \( O(G - D, X) \) and \( O(G - (D \cup \{v\}), X) \) are the same.

Now we apply the technique of iterative compression which was already discussed in Section 1.3. The essence of the iterative compression is the compression step. In our case, we want to solve the following Planar Deletion Compression problem: For a given graph \( G \) and vertex set \( D \) of size \( k + 1 \) such that \( G - D \) is planar, the task is to find a set \( D' \) of size at most \( k \) such that \( G - D' \) is planar or to conclude that no such set exists. If we succeed to solve Planar Deletion Compression by an FPT algorithm, then we would be able to show by the standard technique from Section 1.3 that planar deletion is FPT. Let us remind that we fix an arbitrarily ordering \( v_1, v_2, \ldots, v_n \) of \( V(G) \), and then for each \( i \in \{1, \ldots, n\} \), solve Planar Deletion Compression on the subgraph induced by the first \( i \) vertices, obtain a planar deletion set \( D \) of size \( k \), and obtain the new instance of Planar Deletion Compression on the graph induced by the first \( i + 1 \) vertices with planar deletion set \( D \cup \{v\} \) of size at most \( k + 1 \).

To solve Planar Deletion Compression, for each of the \( 2^{k+1} \) possible partitions of \( D \) into sets \( D_1 \) and \( D_2 \), we want to check if there is a solution \( D' \) such that \( D \cap D' = D_1 \). In other words, we guess the vertices \( D_1 \) of \( D \) from the new solution \( D' \) and the vertices \( D_2 \) which are not in \( D' \). For each such guess, we try to solve the annotated version of the problem, where we ask for a planar deletion set \( D' \) in graph \( G' = G - D_1 \) such that \( D' \cap D_2 = \emptyset \) and \( |D'| \leq k - |D_1| \).

Now the idea of the algorithm is as follows. Graph \( G' - D_2 = G - (D_1 \cup D_2) \) is planar. If the treewidth of \( G' - D_2 \) is at most \( \frac{9}{2} f(k) \) (we will define the value \( f(k) \) a bit later), then the treewidth of \( G' \) is at most \( \frac{9}{2} f(k) + |D_2| \leq \frac{9}{2} f(k) + k \), and we apply the dynamic programming approach to solve the problem in FPT time. If the treewidth of \( G' - D_2 \) is more than \( f(k) \), then by Theorem 2.19, it contains a large grid as a minor. We will show that then \( G' \) contains an irrelevant vertex and we would be able to identify this vertex in polynomial time. In this case, we can identify and delete irrelevant vertices until the treewidth of \( G' - D_2 \) becomes \( \frac{9}{2} f(k) \).

It is not difficult to show that Planar Deletion Compression is FPT parameterized by the treewidth of the input graph, see Exercise 2.82. However, the running time claimed in the following lemma from [129] requires additional ideas. We do not prove this lemma here.

**Lemma 2.41 ([129])**. Planar Deletion Compression is solvable in time \( 2^{O((t \log t) \cdot H)} \) on graphs of treewidth at most \( t \).

For \( p = 2k + 5 \) and \( f(k) = \lfloor (p + 4) \sqrt{k(k + 1) + 4} \rfloor \), let \( H \) be an \( f(k) \times f(k) \)-grid. We claim that if \( G' - D_2 \) contains \( H \) as a minor, then \( G' \) contains an irrelevant vertex. Grid \( H \) can be obtained by contracting and deleting edges of \( G' \). Let \( \tilde{H} \) be the partially triangulated grid obtained only by edge contractions and let \( G_{c} \) be the graph obtained from \( G' \) by contracting
2.7 Irrelevant vertex technique

Fig. 2.9: Set $\mathcal{P}$ of $p \times p$-grids in $[(p + 4)\sqrt{k(k + 1)} + 1 + 4] \times [(p + 4)\sqrt{k(k + 1)} + 1 + 4]$-grid $H$.

these edges. Grid $H$ contains $k(k+1)+1$ vertex disjoint $p \times p$-grids such that each of these grids is at distance at least two from the border of $H$ and also the distance in $H$ between any two of these grids is at least five, see Fig. 2.9. Let $\mathcal{P}$ be the set of these grids.

We prove the following claim.

Claim. In graph $G_c$, no vertex of $D_2$ can have neighbors in $\tilde{H}$ in more than $k+2$ of $p \times p$-grids from $\mathcal{P}$.

Proof. To prove the claim we show that if this is not the case, then there is no solution $D'$ such that $D' \cap D_2 = \emptyset$. Targeting towards a contradiction, suppose that a vertex $x \in D_2$ has neighbors in $k+2$ of $p \times p$-grids from $\mathcal{P}$ and that there is an optimal solution $D'$ such that $D' \cap D_2 = \emptyset$.

Because $D'$ is of size at most $k$, there are at least two $\tilde{p} \times \tilde{p}$-grids, say $Y$ and $Z$ containing no vertices of $D'$ and moreover, such that no vertex within distance 2 in $H$ from these grids contains a vertex of $D'$ as well. Let $y$ be a neighbor of $x$ from $Y$ and $z$ be a neighbor of $x$ in $Z$. It is easy to show that for any two disjoint $p \times p$-subgrids of $H$, there are at least $p>k$ vertex disjoint paths connecting these grids. Thus there is at least one path from $y$ to $z$ containing no vertex of $D'$.

Because both grids are at distance at least two from the border of $H$ and at distance at least five from each other, the union of these grids with a path from $y$ to $z$ and with vertex $x$ contains graph $K_5$ as a minor. To see this, the vertices of grid $Y$ and the vertices adjacent to $Y$ contain a model of $K_4$. We select a model of $K_4$ containing $y$ and such that each vertex of $K_4$ except $y$ contains a vertex at distance 1 from $Y$, see Fig. 2.10. An $yz$-path avoiding $D'$ is drawn in blue in Fig. 2.10. As far as this path (passing from $z$) hits a vertex at distance 2 from $Y$, we can reroute it along vertices at distance 2 from $Y$ in such a way that it touches every element of the model of
Fig. 2.10: Constructing $K_5$ in a neighbourhood of $x \in D_2$.

$K_4$ except $y$. Because no vertex within distance 2 from $Y$ contains a vertex from $D'$, by this construction we obtain a model of $K_5 - e$, i.e. complete graph $K_5$ minus an edge, in $G - D'$. Finally, extending the model of vertex $y$ by adding $x$, we construct the model of $K_5$. Because the distance to the border of $H$ from $Y$ is at least 2, this construction works for any position of $y$ in $Y$. By Kuratowski theorem, planar graph cannot contain $K_5$ as a minor contradicting our assumption that $D'$ is a solution of Planar Deletion Compression.

Therefore, we can assume that for every vertex of $D_2$, its neighbors hit at most $k + 1$ of the $p \times p$-grids from $\mathcal{P}$, and thus there is at least one $p \times p$-grid, say $F$, having no neighbor in $D_2$. Thus the vertices of $F$ induce a planar graph in $G_c$. Also because $p = 2k + 5$, $F$ contains a vertex $v$ and $k + 2$ vertex disjoint cyclic separators separating $v$ from vertices outside $F$. The models of these separators in $G$ contain cyclic separators, and thus $v$ (or all vertices from its model, to be precise) is irrelevant.

To summarize, we solve Planar Deletion Compression, with the following algorithm. If the treewidth of graph $G - D$ is at most $\frac{3}{2} f(k)$, we perform dynamic programming and use Lemma 2.41 to solve the problem in time $2^{O(f(k) \log f(k))} n$. Otherwise, for each partition of $D$ into $D_1$ and $D_2$, we are able to identify an irrelevant vertex $v$ in polynomial time. Graph $G$ has a solution $D'$ of size $k$ such that $D' \cap D = D_1$ if and only if graph $G - v$ has solution with the same properties. The total running time of the algorithm is then $2^k \cdot 2^{O(f(k) \log f(k))} \cdot n^{O(1)} = 2^{O(k^2 \log k)} \cdot n^{O(1)}$.

Finally, let us note that with more ideas the running time of the algorithm can be improved, see Notes to this section.
2.8 Beyond treewidth

Treewidth and tree decomposition are of great importance in graph theory as well as in the design and analysis of graph algorithms. While treewidth is a very fundamental and useful measure of a graph, a natural question to ask if there a better measure? For example, many problems are trivial on complete graphs, but because complete graphs have large treewidth, this type of tractability is not captured by the treewidth. In this section we briefly discuss an alternative width-measure, namely rank-width, its advantages and drawbacks.

\textit{f-width of Sets.} Let $\mathcal{U}$ be a set of elements and let $f : 2^\mathcal{U} \rightarrow \mathbb{Z}^+$ be a function assigning to each subset of $\mathcal{U}$ an integer. A \textit{rooted binary tree} $T$ is a directed tree with a specified vertex $r$ called the root such that the root $r$ has two incoming edges and no outgoing edges and every vertex other than the root has exactly one outgoing edge and either two or zero incoming edges. A leaf of a rooted binary tree is a vertex with no incoming edges. A descendent of an edge $e$ of a rooted binary tree $T$ is the set of vertices from which there exists a directed path to $e$.

A \textit{decomposition} of a set $\mathcal{U}$ is a pair $(T, \mu)$ of a rooted binary tree $T$ and a bijection $\mu$ from $\mathcal{U}$ to the set of all leaves of $T$. For a decomposition $(T, \mu)$ of $\mathcal{U}$ and an edge $e$ of the binary rooted tree $T$, let $X_e \subseteq \mathcal{U}$ be the set of all elements of $\mathcal{U}$ being assigned to a leaf of $T$ which is also a descendent of $e$. Now we define the \textit{f-width of a decomposition} $(T, \mu)$ of $\mathcal{U}$ as the minimum of $f(\mu^{-1}(X_e))$ over all edges $e$ of $T$. Finally, the \textit{f-width} of a finite set $\mathcal{U}$, denoted by $w_f(\mathcal{U})$, is the minimum f-width over all possible decompositions of $\mathcal{U}$. If $|\mathcal{U}| \leq 1$ then $\mathcal{U}$ has no decomposition but we let $w_f(\mathcal{U}) = f(\mathcal{U})$.

Let $G$ be a graph. There are two important width parameters of $G$ that can be defined via $f$-width.

\textit{Branch-width.} For branch-width, we put $\mathcal{U} = E(G)$, the edge set of $G$. For every $X \subseteq E(G)$ its \textit{border} $\delta(X)$ is the set of vertices from $V(G)$ such that every $v \in \delta(X)$ is adjacent to an edge from $X$ and to an edge from $E(G) \setminus X$. The \textit{branch-width} of $G$ is the $f$-width of $\mathcal{U} = E(G)$ with $f(X) = |\delta(X)|$. This definition can be extended to hypergraphs and matroids.

Branch-width is closely related to the tree-width. It is possible to show, that for every graph of branch-width $b$, $b \leq tw(G) + 1 \leq \lfloor \frac{3}{2}b \rfloor$. The most notable exception from the complexity perspective, while both problems are NP-complete, on planar graphs branch-width is computable in polynomial time and the computational complexity of treewidth on planar graph is a long standing open question.

However the second $f$-width parameter, behaves quite differently than the treewidth.

\textit{Rank-width.} The \textit{rank-width} of a graph $G$ is the $f$-width of $\mathcal{U} = V(G)$, where $f = \rho_G$, the cut-rank function of $G$. Here the cut-rank function is defined as
follows. For a vertex subset $X \subseteq V(G)$, let
\[ B_G(X) = (b_{i,j})_{i \in X, j \in V(G) \setminus X}, \]
be the $|X| \times |V(G) \setminus X|$ matrix over the binary field $\mathbb{GF}(2)$ such that $b_{i,j} = 1$ if and only if $\{i, j\} \in E(G)$. In other words, $B_G(X)$ is the adjacency matrix of the bipartite graph formed from $G$ by removing all edges but the edges between $X$ and $V(G) \setminus X$. Finally, $\rho_G(X)$ is the rank of $B_G(X)$.

For example, the rank-width of a complete graph is at most one. Rank-width enjoys several nice properties of treewidth. For every $k$, Hlinený and Oum in [123] gave $f(k) \cdot n^{O(1)}$ time algorithm computing a rank decomposition of width at most $k$, or correctly reporting that the rank-width of a given graph is at least $k$. Also many problems are FPT when parameterized by the rank-width. Roughly speaking, for problems expressible in $\text{MSO}_2$ with logical formulas that do not use edge set quantifications (so-called $\text{MSO}_1$ logic), it is possible to extend the meta theorem of Courcelle to graphs of bounded rank-width. As was shown by Courcelle, Makowsky, and Rotics [54], all problems expressible in $\text{MSO}_1$ logic are fixed-parameter tractable when parameterized by the rank-width. On the other hand, it is possible to show that some of the problems expressible in $\text{MSO}_2$, like HAMILTONIAN CYCLE or EDGE DOMINATING SET are $\text{W}[1]$-hard parameterized by the rank-width.

### 2.9 Exercises

2.42. Show that the pathwidth of an $n$-vertex tree is $O(\log n)$. Construct a class of trees of pathwidth $k$ and $O(3^k)$ vertices.

2.43. Give an algorithm that for a given path decomposition of graph $G$ of width $p$ constructs in time $O(p^2 n)$ a nice path decomposition of the same width.

2.44. What is the treewidth of (a) a complete graph; (b) of a complete bipartite graph; (c) of a forest?

2.45. Prove that every clique of a graph is contained in some bag of its tree decomposition.

2.46. Let $\omega(G)$ denotes the maximum clique size of graph $G$. Show that for any graph $G$, $\text{tw}(G) \geq \omega(G) - 1$.

2.47. Show that treewidth is “minor-monotone” parameter, i.e. for every minor $H$ of a graph $G$, $\text{tw}(H) \leq \text{tw}(G)$.

2.48. Show that the treewidth of a graph $G$ is equal the maximum treewidth of its 2-vertex connected components.

2.49. What is the treewidth of a graph containing no $K_4$ as a minor?

2.50. A graph is outerplanar if it can be embedded in the plane such that all its vertices are on one face. What is the treewidth of an outerplanar graph?
2.9 Exercises

2.51. Show that the treewidth of a simple graph dose not increase after subdividing its edges. (A caveat: when subdividing parallel edges, treewidth can increase from 1 to 2).

2.52. For a graph $G$ given together with its tree decomposition of width $t$, construct in time $O^{(1)}(n)$ a data structure such that for any two vertices $x,y \in V(G)$, it is possible to check in time $O(t)$ if $x$ and $y$ are adjacent.

2.53. Let $T = (T, \chi)$ be a tree decomposition of graph $G$, $t$ be a node of $T$, and $X_t$ be the corresponding bag. Show that for every connected component $C$ of $G - X_t$, the vertices of $C$ are contained in bags of exactly one of the subtrees of $T - t$.

2.54. Prove Lemma 2.3: Let $(T, \chi)$ be a tree decomposition and $st$ be an edge of $T$ and let $T_s$ and $T_t$ be the subtrees of $T - st$ obtained from $T$ by deleting $st$. Let $V_s$ and $V_t$ be the vertices of $G$ contained in bags of $T_s$ and $T_t$, correspondingly. Prove that $\partial(V_s) \subseteq X_s \cap X_t$ and that $X_s \cap X_t$ separates $V_s$ and $V_t$.

2.55. The vertex set of a $d$-dimensional hypercube consists of all possible $2^d$ binary $d$-dimensional vectors. Vertices $v$ and $u$ are adjacent if and only if the corresponding vectors differ in exactly one coordinate. Show that the treewidth of a $d$-dimensional hypercube is $\Theta(\frac{2^d}{\sqrt{d}})$.

2.56. Give an algorithm deciding in time $n^{O(k)}$ if the treewidth of graph $G$ is at most $k$.

2.57. Give an algorithm that for a given tree decomposition of a graph $G$ of width $t$ constructs a nice tree decomposition of $G$ width $t$ in time $O(t^2 \cdot n)$.

2.58. Show that every interval graph has no induced cycles of length more than three.

2.59. This exercise consists of the crucial steps used to prove Theorem 2.7.

1. For a pair of vertices $u, v$ from the same connected component of graph $G$, a vertex set $S$ is a $(u,v)$-separator if $u$ and $v$ are in different connected components of $G - S$. An $(u,v)$-separator is minimal, if it does not contain any other $(u,v)$-separator. Finally, set $S$ is a minimal separator if $S$ is a minimal $(u,v)$-separator for some $u, v \in V(G)$. Let us remark that a minimal separator $S$ can properly contain another minimal separator $S'$. This can happen if $S'$ separates other pair of vertices than $S$. A connected component $C$ of $G - S$ is full if $S = N(C)$.

Show that every minimal separator $S$ has at least two full components.

Show that every minimal separator of a chordal graph is a clique.

2. (Dirac’s Lemma) A vertex $v$ is simplicial if its closed neighbourhood $N[v]$ is a clique. Show that every chordal graph $G$ on at least two vertices has at least two nonadjacent simplicial vertices. Moreover, if $G$ is not complete, then it has at least two nonadjacent simplicial vertices.

3. Prove that every chordal graph $G$ admits a tree decomposition such that every bag of the decomposition is a maximal clique of $G$.

4. Prove Theorem 2.7.

2.60. We define $k$-tree inductively. A clique on $k + 1$ vertices is a $k$-tree. A $k$-tree $G$ can be obtained from a $k$-tree by adding a new vertex and making it adjacent to $k$ vertices from a clique of $G$. Show that every $k$-tree is a chordal graph of treewidth $k$. Prove that for every graph $G$ and integer $k$, $G$ is a subgraph of a $k$-tree if and only if $tw(G) \leq k$.

2.61. Let $T$ be a tree and $T_1, T_2, \ldots, T_n$ be subtrees of $T$. Graph $G$ is an intersection graph of subtrees of a tree, if there is a tree $T$ such that for every $v \in V(G)$, we can associate a subtree $T_v$ of $T$ such that $uv \in E(G)$ if and only if $T_u \cap T_v \neq \emptyset$. Show that graph is chordal if and only if it is an intersection graph of subtrees of a tree.
2.62. Let $G$ be an $n$-vertex graph of treewidth at most $k$. Show that the number of edges in $G$ is at most $kn$.

2.63. Improve the algorithm for DOMINATING SET parameterized by treewidth $t$ to obtain the running time $4^t \cdot t^{O(1)} \cdot n$.

2.64. Show that the following property: graph $G$ does not contain graph $H$ as a minor (here we interpret graph $H$ as of a constant size) is expressible in MSO$_2$.

2.65. For any graph $G$ of treewidth $k$ and weight function $w : V(G) \rightarrow \{0, 1\}$, there is a partition of $V(G)$ into $L$, $B$ and $R$ such that
\[
\max\{w(L), w(R)\} \leq \frac{2w(V(G))}{3}
\]
and
\[
|B| \leq k + 1.
\]

2.66. Prove the following version of the classical result of Lipton and Tarjan on separators in planar graphs. For any planar $n$-vertex graph $G$ and $W \subseteq V(G)$, there is a set $S \subseteq V(G)$ of size at most $\frac{2}{7} \sqrt{n} + 1$ such that every connected component of $G - B$ contains at most $\frac{|W|}{2}$ vertices of $W$.

2.67. Let $G$ be an $n$-vertex graph given together with its tree decomposition of with $t$. Show that
- FEEDBACK VERTEX SET
- CONNECTED DOMINATING SET
- ODD CYCLE TRANSVERSAL
- HAMILTONIAN PATH and $k$-PATH
- CHROMATIC NUMBER
- CYCLE PACKING
- CONNECTED VERTEX COVER
- CONNECTED FEEDBACK VERTEX SET
are solvable in time $t^{O(1)} \cdot n$.

2.68. Show that SUBGRAPH ISOMORPHISM is solvable in time $f(H)|V(G)|^{\text{tw}(H)}$ for some function $f$.

2.69. Show that SUBGRAPH ISOMORPHISM is FPT parameterized by $\text{tw}(G) + |H|$.

2.70. Show that BISECTION is solvable in time $2^{\text{tw}(G)} \cdot n^{O(1)}$.

2.71. Show that CHROMATIC NUMBER is FPT parameterized by treewidth.

2.72. Show that vertex cover is a minor-closed parameter. In other words, for any minor $H$ of graph $G$, vertex cover of $H$ does not exceed the vertex cover of $G$.

2.73. Let $t$ be the treewidth of the input graph $G$.
- In the $r$-CENTER problem, we are asked to find $k$ vertices such that every other vertex of $G$ is at distance at most $r$ from some of these vertices. Show that $r$-CENTER is solvable in time $r^{O(t)} n^{O(1)}$.
- In the INDUCED MATCHING problem we are asked if there is a subset of $2k$ vertices in $G$ inducing matching. Show that INDUCED MATCHING is solvable in time $2^{O(t)} n^{O(1)}$. 
2.74. Show that **Feedback Vertex Set**, **Induced Matching**, **Cycle Packing**, **Scattered Set** for fixed value of $d$, $k$-Path, **Dominating Set**, and $r$-Center for fixed $r$, are bidimensional.

2.75. **List Coloring** is a generalization of **Vertex Coloring**: given a graph $G$, a set of colors $C$, and a list function $L: V(G) \to 2^C$ (that is, a subset of colors $L(v)$ for each vertex $v$), the task is to assign a color $c(v) \in L(v)$ to each vertex $v \in V(G)$ such that adjacent vertices receive different colors. Show that on an $n$-vertex graph $G$, **Vertex Coloring** can be solved in time $n^{O(tw(G))}$.

2.76. Show that **Scattered Set** is FPT on planar graphs parameterized by $k + d$. For fixed $d$, give an algorithm of running time $2^{O(\sqrt{n})}n^{O(1)}$.

2.77. We define vertex-subset maximization problem $\Pi$ such that for a graph $G$ with the maximum size of independent set $\alpha(G)$ and the number of isolated vertices $i(G)$, $OPT(\Pi(G)) \max\{0, \alpha(G) - 2 \cdot i(G)\}$. Prove that $\Pi$ is bidimensional and construct a family of planar graphs $G_k$ that show that for every $k \geq 1$, $OPT(\Pi(G_k)) = 0$ and $tw(G_k) \geq k$. Let us note that due to Lemma 2.31 graphs $G_k$ cannot be connected.

2.78. Prove Lemma 2.36: Graph $G$ is planar if and only if it has a cycle $X$ such that for every $X$-bridge $B$, graph $X \cup B$ is planar and the overlap graph $O(G, X)$ is bipartite.

2.79. Prove that for every $t > 1$, $\Xi_t$ contains a bramble of order $t + 1$ and thus $tw(\Xi_t) = t$.

2.80. The input of the **Partial Vertex Cover** problem is an graph $G$ with two integers $k$ and $s$, and $(G, k, s)$ is a yes-instance if $G$ has a set of $k$ vertices that covers at least $s$ edges. Show that on planar graphs **Partial Vertex Cover** is FPT parameterized by $k$. Give a subexponential $2^{O(\sqrt{n})}n^{O(1)}$ algorithm for **Partial Vertex Cover** on planar graphs.

2.81. In the **Tree Spanner** problem, for a given connected graph $G$ and integer $k$, the task is to decide if $G$ contains a spanning tree $T$ such that for every pair of vertices $u, v$ of $G$, the distance between $u$ and $v$ in $G$ is at most $k$ times the distance between $u$ and $v$ in $T$. Show that on planar graphs **Tree Spanner** is FPT parameterized by $k$.

2.82. Show that **Planar Deletion** and **Planar Deletion Compression** are FPT parameterized by the treewidth of the input graph.

**Hints**

2.42. Take a minimal $n$-vertex tree $T$ of pathwidth $k$, i.e., a tree such that each of its subtrees has smaller pathwidth. There is a vertex $v$ of $T$ such that every branch of $T$ has at most $n/2$ vertices but each of the branches has smaller pathwidth due to minimality of $T$. Proceed by induction. For the second part of the exercise. Using graph searching, show that if a tree $T$ has a vertex $v$ such that at least three components of $T - \{v\}$ have pathwidth $k$, then the pathwidth of $T$ is at least $k + 1$.

2.52. Construct an orientation $H$ of $G$ with outdegrees at most $t + 1$. Such orientation can be constructed from a tree decomposition: begin with empty $H$, pick the lowest forget
vertex node for vertex $v$, orient all edges incident with $v$ outside $v$ in $H$, remove $v$ from the tree decomposition, etc.

2.55 A $k$-vertex subset $X$ of $C$ is optimal, if for every $k$-vertex set $Y$ of $C$, the size of the border $|\partial(Y)|$ is at least $|\partial(X)|$. Consider an ordering $L = (v_1, v_2, \ldots, v_{2d})$ of vertices of $C$ such that vertex $v_1$ precedes $v_2$ if and only if $||v_1|| < ||v_2||$ or $||v_1|| = ||v_2||$ and $v_1$ precedes $v_2$ lexicographically. Show that for every $i$, the set of the first $i$ vertices is optimal. This statement is called isoperimetric inequality due to the following. The classical isoperimetric problem is to identify among all closed curves in the plane of fixed perimeter the curve maximizing the area of its enclosed region. For the plane this is the circle. Similar property holds for vertices of the hypercube $C$—the first $\sum_{i=1}^{d/2} \binom{d}{i}$ vertices in the ordering $L$ form the $d/2$-dimensional ball in $C$. The border of the ball is $\binom{d/2}{d/2}$ and we claim that this ball is the optimal set. (This is also a classical result of Harper [120].) To show the lower bound, consider the cops and robber game when the first time cops cleared the optimal set. (This is also a classical result of Harper [120].) To show the lower bound, consider the cops and robber game when the first time cops cleared the optimal set. (This is also a classical result of Harper [120].) To show the lower bound, consider the cops and robber game when the first time cops cleared the optimal set. (This is also a classical result of Harper [120].) To show the lower bound, consider the cops and robber game when the first time cops cleared the optimal set. (This is also a classical result of Harper [120].)

2.56 We want to find out whether $k$ cops have a winning strategy on a graph $G = (V, E)$, which implies that the treewidth of $G$ is at most $k - 1$. To find it out, there is a standard technique from games on graphs. The idea is to represent all configurations of the game and their dependencies by a directed graph called the arena. The vertex set of the arena is partitioned into two sets of nodes $V_1$ and $V_2$. It also also has a specified node $v \in V_1$ corresponding to the initial position of the game and a node $u \in V_2$ corresponding to the final positions. There are two players, Player 1 and Player 2, who alternatively move a token along arcs of the arena. The game starts by placing the token on $v$, and then the game alternates in rounds, starting with Player 1. At each round, if the token is on a vertex from $V_i$, $i = 1, 2$, then the $i$th player slides the token from its current position along an arc to a new position. Player 1 wins if he manage to slide the token to the final position node $u$. Otherwise, Player 2 wins. Construct an auxiliary directed graph $H$, the arena, with nodes of the arena corresponds to a subset $X \subseteq V(G)$ of size at most $k$, corresponding to the position occupied by cops, and a connected component $R$ of $G - X$ selected by the robber. Thus there are $\sum_{i=1}^{k} \binom{n}{i} \cdot n = n^{O(k)}$ nodes in $H$. How to construct arcs in such a graph such that $H$ will contains the complete information on the game? How to search $H$ to decide if $k$ cops can succeed?

2.59 1. Use the fact that every minimal separator $S$ has at least two full components.
2. Use induction on the number of vertices in $G$. If $G$ has two non-adjacent vertices $v_1$ and $v_2$, then a minimal $(v_1, v_2)$-separator $S$ is a clique. Use induction assumption to find a simplicial vertex in each of the full components. 3. Let $v$ be a simplicial vertex of $G$. Use induction on the number of vertices $n$ and construct tree decomposition of $G$ from tree decomposition of $G - v$. 4. (i) $\Rightarrow$ (ii). Let $T = (T, \chi)$ be a tree decomposition of width $k$. Let $H$ be the supergraph of $G$ obtained from $G$ by transforming every bag of $T$ into a clique. Then $T = (T, \chi)$ is also a tree decomposition of $H$ of width $k$, and thus the maximum clique size of $H$ does not exceed $k + 1$. Show that $H$ is chordal. (ii) $\Rightarrow$ (iii). Here you need to construct the search strategy of cops. This strategy imitates the strategy of two cops on a tree. (iii) $\Rightarrow$ (i). If $k + 1$ cops can catch a visible robber in $G$, then by the arguments above, $G$ has no bramble of order $k + 1$. By Theorem 2.6, the treewidth of $G$ is at most $k$.

2.63 Redefine the meaning of grey vertices such that the number of possible triples in formula 2.3 is 4 instead of 5.

2.64 To encode that $G$ contains a model of $H$ we can do the following. Let $v_1, \ldots, v_h$ be the vertices of $H$. For every vertex $v_i$, there is vertex subset $V_i$ of $V(G)$. Each of these sets is connected (we already saw how to express connectivity of a set in MSO$_2$). These sets...
are pairwise disjoint (every vertex of $G$ is in at most one of the sets). And for every edge of $H$ there should be an edge between the corresponding sets in $G$.

2.66 Use the Balanced Separators lemma (Lemma 2.15) and Planar Excluded Grid Theorem.

2.80 The problem is not bidimensional however, it can be reduced to a a bidimensional problem. We order the vertices of an input graph $G$ according to their degrees. Show that there is a solution $C$ such that for some $i$, $C$ is a dominating set in the graph induced by the first $i$ vertices. From here we can use bidimensionality.

2.81 A spanner of a sufficiently large grid should contain a path crossing grid from top to bottom and from left to right. Thus if grid is large, the spanner should contain a subgrid as a minor and thus a cycle. A caveat: the problem is not contraction closed, it is possible to contract edges and increase the parameter. While grid arguments will work at the end, we have to be careful here.

Notes

The treewidth of a graph is a natural and fundamental graph parameter, it was rediscovered several times in different settings. Here we follow the definition of Robertson and Seymour [179, 180, 181]. Equivalent notions were introduced earlier by Halin in [118] and to Bertelé and Brioschi [12]. See also Arnborg and Proskurowski [8]. The proof that computing the treewidth of a graph is NP-hard is due to Arnborg, Corneil and Proskurowski [6].

The notion of vertex separation number is due to Lengauer [147]. This notion was studied due to graph searching, gate-matrix layout, and other graph parameters [90, 134, 135, 168]. The proof that vertex separation number is equal to pathwidth is attributed to Kinnersley [133].

The classical book of Golumbic discusses different algorithmic and structural properties of perfect, and in particular chordal graphs [110]. Dirac’s Lemma is due to Dirac [77]. Theorem 2.6 is proved by Seymour and Thomas in [189]. The proof of this theorem is also contained in Diestel’s book [76].

Algorithm for Dominating Set of running time $4^t \cdot t^O(1)n$ is due to Alber et al. [2]. An algorithm for weighted Steiner Tree is discussed by Chimani et al. [45]. Techniques for reducing the polynomial dependence of the treewidth in dynamic programming for different problems are discussed by Bodlaender et al. [21] and Chimani et al [45]. The paper of Bodlaender, Bonsma and Lokshtanov [21] discusses the ways of optimizing the polynomial dependence on the treewidth in the dynamic programming algorithms.

Courcelle’s Theorem is proven in [52, 51]. The extensions for optimization problems are due to Arnborg et al. [7] and Borie et al. [28].

The first treewidth FPT-approximation algorithm is due to Robertson and Seymour [180]. Our description of the algorithm mostly follows Reed [178]. There is a large variety of parameterized and approximation algorithms for treewidth in the literature which
differ by approximation ratio, the exponential dependence of \( k \) and polynomial dependence of the input. The best known polynomial time approximation is due to Feige et al. [93], this is a factor \( O(k \cdot \sqrt{\log k}) \) approximation algorithm running in time \( n^{O(1)} \). The best known exact FPT algorithm is due to Bodlaender [19], this algorithm for every fixed \( k \), decides if the treewidth of a given graph at most \( k \) can be done in linear, more precisely, \( k^O(k^3) \) time. There are several approximation algorithms, including the one given in this chapter, with better dependence of \( k \), but worse dependence of \( n \), like \( (3 + 2/3) \)-approximation in time \( O(2^{69.82k^3}k^3n^2) \) of Amir [5]. A single-exponential (as a function of treewidth) and linear (as the input length), i.e. time \( O(2^{O(k)})n \) \( 5 \)-approximation is given in [24].

Planar Excluded Grid Theorem is due to Robertson, Seymour and Thomas [183]. The refined version of this theorem used in this chapter is due to Gu and Tamaki [112]. Daimaine and Hajiaghayi extended this theorem for \( H \)-minor-free graphs in [73]. The first Excluded Grid Minor Theorem is due to Robertson and Seymour [181]. There were several improvements in the exponential dependency of treewidth in the size of the excluded grid by Robertson, Seymour and Thomas [183], Kawarabayashi and Kobayashi [132], and Leaf and Seymour [146]. It was open for many years whether a polynomial relationship could be established between the treewidth of a graph \( G \) and the size of its largest grid minor. This question was resolved positively by Chekuri and Chuzhoy in [37] (Theorem 2.18).

Shifting technique is one of the common techniques for designing PTAS on planar graphs. It dates back to the work of Baker [9]. Grohe showed in [111] how shifting technique combined with other tools from Robertson-Seymour Graph Minors, can be extended to \( H \)-minor-free graphs.

For complexity of Subgraph Isomorphism for different parameterizations, we refer to the paper of Marx and Pilipczuk [164]. Eppstein [91] gave the first linear time algorithm for Subgraph Isomorphism on planar graphs in time \( k^{O(k)}n \). Dorn [79] improved the running time to \( 2^{O(k)}n \). To the best of our knowledge, the existence of polynomial algorithm for Bisection on planar graphs remains open. For general graphs the problem is in FPT.

Bidimensionality was defined in [70], see also surveys [72, 80]. Contraction exclusion theorem is from [99]. Kernelization algorithms for bidimensional problems are given in [103]. EPTAS for bidimensional problems are discussed in [71, 101]. Subexponential algorithm for Partition Vertex Cover is from [102]. The parameterized algorithms for Tree Spanner on planar, and more generally on apex-minor-free graphs are given in [88].

The proof of planarity criterion (Lemma 2.36) can be found in [193, Thm 3.8]. The irrelevant vertex technique was used in the work of Robertson and Seymour on Vertex Disjoint Paths [182]. There are several constructive algorithms for Planar Deletion [166, 131], resulting in algorithm of running time \( k^{O(k)}n \) [129].

Rank-width was defined by Oum and Seymour [171]. Hlinený and Oum in [123] showed that deciding if the rank-width of a graph is at most \( k \) is FPT. From the work of Courcelle, Makowsky, and Rotics [54] on a related parameter, named clique-width, it follows that all problems expressible in MSO\(_1\) logic are fixed-parameter tractable when parameterized by the rank-width of a graph. Lower bounds on dynamic programming algorithms for clique-width and thus for rank-width, are discussed in [98].

Further reading. The book of Diestel [76] contains a nice overview of the role the treewidth is playing in the Graph Minors project. We also recommend the following surveys of Reed [178, 177]. The proof of Courcelle’s theorem and its applications are discussed in the book of Flum and Grohe [97]. The book of Courcelle and Engelfriet [53] is a a thorough description of monadic second-order logic on graphs. An extensive overview of different algorithmic and combinatorial aspects of treewidth as well as its relation to other graph parameters is given in the survey of Bodlaender [20]. Overview of different extensions of treewidth and rank-width to directed graphs, matroids, hypergraphs, etc., is provided in the survey of Hlinený et al. [124].
Chapter 3
Finding cuts and separators

Problems related to cutting a graph into parts satisfying certain properties or related to separating different parts of the graph from each other form a classical area of graph theory and combinatorial optimization, with strong motivation coming from applications. Many different versions of these problems were studied in the literature: one may remove sets of edges or vertices in a directed or undirected graph, the goal can be separating two or more terminals from each other, cutting the graph into a certain number of parts, subject to constraints on the sizes of the parts, etc. Besides some notable exceptions (e.g., minimum cut $s-t$ cut, minimum multiway cut in planar graphs with fixed number of terminals), most of these problems are NP-hard. In this chapter, we investigate the fixed-parameter tractability of some of these problems parameterized by the size of the solution, that is, the size of the cut that we remove from the graph (one could parameterize these problems also by, e.g., the number of terminals, while leaving the size of the solution unbounded, but such parameterizations are not the focus of this chapter). It turns out that small cuts has certain very interesting extremal combinatorial aspects that can be exploited in FPT algorithms. The notion of important cuts formalizes this extremal property and give a very convenient tool for the treatment of these problems.

There is another class of problems that are tightly related to cut problems: transversal problems, where we have to select edges/vertices to hit certain objects in the graph. For example, the Odd Cycle Transversal problem asks for a set of $k$ vertices hitting every odd cycle in the graph. As we have seen in reference, iterative compression can be used to transform Odd Cycle Transversal into a series of minimum cut problems. The Directed Feedback Vertex Set problem asks for a set of $k$ vertices hitting every directed cycle in a graph. The fixed-parameter tractability of Directed Feedback Vertex Set was a long-standing open problem; its solution was eventually reached by a combination of iterative compression and solving a directed cut problem (essentially) using important cuts. There are other ex-
amples where the study of a transversal problem reveals that it can be turned into an appropriate cut problem.

There is a particular difficulty that we face in the presentation of the basic results for the cut problems. There are many variants: we can delete vertices or edges, the graph can be directed or undirected, we may add weights (e.g., to forbid certain edges/vertices to be in the solution). While the basic theory is very similar for these variants, they require different notation and slightly different proofs. In this chapter, rather than presenting the most general form of these results, we mostly focus on the undirected edge versions, as they are the most intuitive and notationally cleanest. Then we go to the directed edge versions to treat certain problems specific to directed graphs (e.g., DIRECTED FEEDBACK VERTEX SET) and finish the chapter by briefly commenting on the vertex variants of these results.

We assume basic familiarity with the concept of cuts, flows, and algorithms for finding minimum cuts. Nevertheless, Section 3.1 reviews some of the finer points of minimum cuts in the form we need that later. Section 3.2 introduces the notion of important cuts and presents the bound on their number and an algorithm to enumerate them. Then Section 3.3 uses important cuts to solve problems such as EDGE MULTIWAY CUT.

Very recently, a new way of using important cuts was introduced in one of the first FPT algorithm multicut. The “random sampling of important separators” technique allows us to restrict our attention to solutions that have a certain structural property, which can make the search for the solution much easier or can reduce it to some other problem. After the initial application for multicut problems, several other results used this technique as an ingredient of the solution. Unfortunately, most of these results are either on directed graphs (where the application of the technique is more delicate) or use several other nontrivial steps, making their presentation beyond the scope of this textbook. Section 3.4 presents the technique on a clustering problem where the task is to partition the graphs into classes of size at most \( p \) such that there are at most \( q \) edges leaving each class. This example is somewhat atypical (as it is not directly a transversal problem), but we can use it to demonstrate the random sampling technique in a self-contained way.

Section 3.5 generalizes the notion of important cuts to directed graphs and states the corresponding results without proofs. We point out that there are significant differences in the directed setting: the way important cuts were used to solve EDGE MULTIWAY CUT in undirected graphs does not generalize to directed graphs. Nevertheless, we can solve a certain directed cut problem called SKEW EDGE MULTICUT using important cuts. In Section 3.6, a combination of the algorithm for SKEW EDGE MULTICUT and iterative compression is used to show the fixed-parameter tractability of DIRECTED FEEDBACK VERTEX SET and DIRECTED FEEDBACK ARC SET.

Section 3.7 discusses the version of the results for vertex-deletion problems. We define the notions required for handling vertex-deletion problems and
3.1 Minimum cuts

An \((X, Y)\)-cut is a set \(S\) of edges that separate \(X\) and \(Y\) for each other, that is, \(G \setminus S\) has no \(X - Y\) path. We need to distinguish between different notions of minimality. An \((X, Y)\)-cut \(S\) is a minimum \((X, Y)\)-cut if there is no \((X, Y)\)-cut \(S'\) with \(|S'| < |S|\). An \((X, Y)\)-cut is (inclusionwise) minimal if there is no \((X, Y)\)-cut \(S'\) with \(S' \subset S\) (see Figure 3.1). Observe that every minimum cut is minimal, but not necessarily the other way around. We allow parallel edges in this section: this is essentially the same as having arbitrary integer weights on the edges.

It will be convenient to look at minimal \((X, Y)\)-cuts from a different perspective, viewing them as edges on the boundary of a certain set of vertices. If \(G\) is an undirected graph and \(R \subseteq V(G)\) is a set of vertices, then we denote by \(\Delta_G(R)\) the set of edges with exactly one endpoint in \(R\) (we omit the subscript \(G\) if it is clear from the context). Let \(S\) be a minimal \((X, Y)\)-cut in \(G\) and let \(R\) be the set of vertices reachable from \(X\) in \(G \setminus S\); clearly, we have \(X \subseteq R \subseteq V(G)\setminus Y\). Then it is easy to see that \(S\) is precisely \(\Delta(R)\). Indeed, every such edge has to be in \(S\) (otherwise a vertex of \(V(G)\setminus R\) would be reachable from \(X\)) and removing any edge of \(S\) with both endpoints in \(R\) or both endpoints in \(V(G)\setminus R\) would not change the fact that the set is an \((X, Y)\)-cut, contradicting minimality.

**Proposition 3.1.** If \(S\) is a minimal \((X, Y)\)-cut, then \(S = \Delta(R)\), where \(R\) is the set of vertices reachable from \(X\) in \(G \setminus S\).

---

**Fig. 3.1:** The set \(\Delta\{x_1, x_2, a, b, c, d, e, f\} = \{e y_1, f y_2\}\) is a minimum \((X, Y)\)-cut (a hence minimal); the set \(\Delta\{x_1, x_2, a, b\}\) = \{ac, bc, bd\} is a minimal \((X, Y)\)-cut, but not minimal. The set \(\Delta\{x_1, x_2, a, c, d\}\) = \{x_2b, ab, bc, bc, ce, df\} is an \((X, Y)\)-cut, but not minimal.
Therefore, we may always characterize a minimal \((X, Y)\)-cut \(S\) as \(\Delta(R)\) for some set \(X \subseteq R \subseteq V(G) \setminus Y\). Let us also note that \(\Delta(R)\) is an \((X, Y)\)-cut for every such \(R\), but not necessarily a minimal \((X, Y)\)-cut (see Figure 3.1).

The well-known Max-Flow Min-Cut duality implies that the size of the minimum \((X, Y)\)-cut is the same as the maximum number of pairwise edge-disjoint \(X - Y\) paths. Classical maximum flow algorithms can be used to find a minimum cut and a corresponding collection of edge-disjoint \(X - Y\) paths of the same size. We do not review the history of these algorithms and their running times here, as in setting, we usually want to find a cut of size at most \(k\), where \(k\) is assumed to be a small constant. Therefore, the following fact is sufficient for our purposes: each round of the algorithm of Ford and Fulkerson takes linear time, and \(k\) rounds are sufficient to decide if there is an \((X, Y)\)-cut of size at most \(k\). We state this fact in the following form.

**Theorem 3.2.** Given a graph \(G\), disjoint sets \(X, Y \subseteq V(G)\), and an integer \(k\), there is an \(O(k(|V(G)| + |E(G)|))\) time algorithm that either

- correctly concludes that there is no \((X, Y)\)-cut of size at most \(k\), or
- returns a minimum \((X, Y)\)-cut \(\Delta(R)\) and a collection of \(|\Delta(R)|\) pairwise edge-disjoint \(X - Y\) paths.

Submodular set functions play an essential role in many areas of combinatorial optimization, and they are especially important for problems involving cuts and connectivity. Let \(f : 2^{V(G)} \to \mathbb{R}\) be a set function assigning a real number to each subset of vertices of a graph \(G\). We say that \(f\) is submodular if it satisfies the following inequality for every \(X, Y \subseteq V(G)\):

\[
f(A) + f(B) \geq f(A \cap B) + f(A \cup B) . \tag{3.1}
\]

We will use the well-known fact that the function \(d_G\) is submodular.

**Theorem 3.3.** The function \(d_G\) is submodular for every undirected graph \(G\).

**Proof.** Let us classify each edge \(e\) according to the location of its endpoints (see Figure 3.2) and calculate its contribution to the two sides of (3.1):

1. If both endpoints of \(e\) are in \(A \cap B\), in \(A \setminus B\), in \(B \setminus A\), or in \(V(G) \setminus (A \cup B)\), then \(e\) contributes 0 to both sides.
2. If one endpoint of \(e\) is in \(A \cap B\), and the other is either in \(A \setminus B\) or in \(B \setminus A\), then \(e\) contributes 1 to both sides.
3. If one endpoint of \(e\) is in \(V(G) \setminus (A \cup B)\), and the other is either in \(A \setminus B\) or in \(B \setminus A\), then \(e\) contributes 1 to both sides.
4. If \(e\) is between \(A \cap B\) and \(V(G) \setminus (A \cup B)\), then \(e\) contributes 2 to both sides.
5. If \(e\) is between \(A \setminus B\) and \(B \setminus A\), then \(e\) contributes 2 to the left-hand side and 0 to the right-hand side.

As the contribution of each edge \(e\) to the left-hand side is at least as much as its contribution to the right-hand side, inequality (3.1) follows. \(\square\)
A reason why submodularity of $d_G$ is particularly relevant to cut problems is that if $\Delta(A)$ and $\Delta(B)$ are both $(X,Y)$-cuts, then $\Delta(A \cap B)$ and $\Delta(A \cup B)$ are both $(X,Y)$-cuts: indeed, $A \cap B$ and $A \cup B$ both contain $X$ and disjoint from $Y$. Therefore, we can interpret Theorem 3.3 as saying that if we have to $(X,Y)$-cuts $(A)$ and $(B)$ of a certain size, then two new $(X,Y)$-cuts $(A \setminus B)$, $(A \setminus B)$ can be created and there is a bound on their total size.

The minimum $(X,Y)$-cut is not necessarily unique, in fact, a graph can have large number of minimum $(X,Y)$-cuts. Suppose, for example, that $X = \{x\}$, $Y = \{y\}$, and $k$ paths connect $x$ and $y$, each of length $n$. Then selecting one edge from each path gives a minimum $(X,Y)$-cut, hence there are $n^k$ different minimum $(X,Y)$-cuts. However, as we show below, there is a unique minimum $(X,Y)$-cut $\Delta(R_{\min})$ that is closest to $X$ and a unique minimum $(X,Y)$-cut $\Delta(R_{\max})$ closest to $Y$, in the sense that the sets $R_{\min}$ and $R_{\max}$ are minimum and maximum possible, respectively (see Figure 3.3(a)). The proof of this statement follows from an easy application of the submodularity of $d_G$.

**Theorem 3.4.** Let $G$ be a graph and $X, Y \subseteq V(G)$ two disjoint sets of vertices. There are two minimum $(X,Y)$-cuts $\Delta(R_{\min})$ and $\Delta(R_{\max})$ such that if $\Delta(R)$ is a minimum $(X,Y)$-cut, then $R_{\min} \subseteq R \subseteq R_{\max}$.

**Proof.** Consider the collection $\mathcal{R}$ of every set $R \subseteq V(G)$ for which $\Delta(R)$ is a minimum $(X,Y)$-cut. We show that there is a unique inclusionwise minimal set $R_{\min}$ and a unique inclusionwise maximal set $R_{\max}$ in $\mathcal{R}$. Suppose for contradiction that $\Delta(R_1)$ and $\Delta(R_2)$ are minimum cuts for two inclusionwise minimal sets $R_1 \neq R_2$ of $\mathcal{R}$. By (3.1), we have

$$d_G(R_1) + d_G(R_2) \geq d_G(R_1 \cap R_2) + d_G(R_1 \cup R_2).$$

If $\lambda$ is the minimum $(X,Y)$-cut size, then the left-hand side is exactly $2\lambda$, hence the right-hand side is at most $2\lambda$. Observe that $\Delta(R_1 \cap R_2)$ and $\Delta(R_1 \cup R_2)$ are both $(X,Y)$-cuts. Taking into account that $\lambda$ is the minimum $(X,Y)$-cut size, the right-hand side is also exactly $2\lambda$, with both terms being exactly

![Fig. 3.2: The different types of edges in the proof of Theorem 3.3](image-url)
Fig. 3.3: (a) A graph $G$ with 3 edge-disjoint $(X, Y)$-paths and the $(X, Y)$-cuts $R_{\text{min}}$ and $R_{\text{max}}$ of Theorem 3.4. (b) The corresponding residual directed graph $D$ defined in the proof of Theorem 3.5.

That is, $\Delta(R_1 \cap R_2)$ is a minimum $(X, Y)$-cut. Now $R_1 \neq R_2$ implies that $R_1 \cap R_2 \subset R_1, R_2$, contradicting the assumption that both $R_1$ and $R_2$ are inclusionwise minimal in $\mathcal{R}$.

The same argument gives a contradiction if $R_1 \neq R_2$ are inclusionwise maximal sets of the collection: then we observe that $\Delta(R_1 \cup R_2)$ is also a minimum $(X, Y)$-cut. □

While the proof of Theorem 3.4 is not algorithmic, one can find, e.g., $R_{\text{min}}$ by repeatedly adding vertices to $Y$ as long as this does not increase the minimum cut size (Exercise 3.3). However, there is a linear-time algorithm for finding these sets.

**Theorem 3.5.** Let $G$ be a graph and $X, Y \subseteq V(G)$ two disjoint sets of vertices. Let $k$ be the size of the minimum $(X, Y)$-cut. The sets $R_{\text{min}}$ and $R_{\text{max}}$ of Theorem 3.4 can be found in time $O(k(|V(G)| + |E(G)|))$.

**Proof.** Let us invoke the algorithm of Theorem 3.2 and let $P_1, \ldots, P_k$ be the pairwise edge-disjoint $X - Y$ paths returned by the algorithm. We build the residual directed graph $D$ as follows. If edge $xy$ of $G$ is not used by any of the paths $P_i$, then we introduce both $(x, y)$ and $(y, x)$ into $D$. If edge $xy$ of
3.2 Important cuts

$G$ is used by some $P_i$ in such a way that $x$ is closer to $X$ on path $P_i$, then we introduce the directed edge$^1$ $(y, x)$ into $D$ (see Figure 3.3(b)).

We show that $R_{\min}$ is the set of vertices reachable from $X$ in the residual graph $D$ and $R_{\max}$ is the set of vertices from which $Y$ is not reachable in $D$.

Let $\Delta(R)$ be a minimum $(X, Y)$-cut of $G$. Each $P_i$ uses an edge of $\Delta(R)$. In fact, as there are $k$ such edges, each $P_i$ uses exactly one edge of $\Delta(R)$. This means that after $P_i$ leaves $R$, it never returns to $R$. Therefore, if $P_i$ uses an edge $ab \in \Delta(R)$ with $a \in R$ and $b \notin R$, then $a$ is closer to $X$ on $P_i$. This implies that $(a, b)$ is not an edge of $D$. As this is true for every edge of the cut $\Delta(R)$, we get that $V(G) \setminus R$ is not reachable from $X$ in $D$; in particular, $Y$ is not reachable.

Let $R_{\min}$ be the set of vertices reachable from $X$ in $D$. We have shown in the previous paragraph that $Y$ is not reachable from $X$ in $D$, hence $\Delta(R_{\min})$ is an $(X, Y)$-cut of $G$. If we can show that this cut is a minimum $(X, Y)$-cut, then we are done: we have shown that if $\Delta(R)$ is a minimum $(X, Y)$-cut, then $V(G) \setminus R$ is not reachable from $X$, implying that $V(G) \setminus R \subseteq V(G) \setminus R_{\min}$ and hence $R_{\min} \subseteq R$.

Every path $P_i$ uses at least one edge of the $(X, Y)$-cut $\Delta(R_{\min})$. Moreover, $P_i$ cannot use more than one edge of the cut: if $P_i$ leaves $R_{\min}$ and later returns to $R_{\min}$ on an edge $ab$ with $a \notin R_{\min}$, $b \in R_{\min}$ and $a$ is closer to $X$ on $P_i$, then $(b, a)$ is an edge of $D$ and it follows that $a$ is also reachable from $X$ in $D$, contradicting $a \notin R_{\min}$. Therefore, $\Delta(R_{\min})$ can have at most $k$ edges, implying that it is a minimum $(X, Y)$-cut. Therefore, $R_{\min}$ satisfies the requirements.

A symmetrical argument shows that the set $R_{\max}$ containing all vertices from which $Y$ is not reachable in $D$ satisfies the requirements. \qed

3.2 Important cuts

Most of the results of this chapter are based on the following definition.

**Definition 3.6.** Let $G$ an undirected graph and let $X, Y \subseteq V(G)$ be two disjoint sets of vertices. Let $S \subseteq E(G)$ be an $(X, Y)$-cut and let $R$ be the set of vertices reachable from $X$ in $G \setminus S$. We say that $S$ is an important $(X, Y)$-cut if it is inclusionwise minimal and there is no $(X, Y)$-cut $S'$ with $R_{\min}$.

$^1$ The reader might find introducing the edge $(x, y)$ instead of $(y, x)$ more natural and indeed the rest of the proof would work just as well after appropriate changes. However, here we follow the standard definition of residual graphs used in network flow algorithms.
An intuitive interpretation of Definition 3.6 is that we want to minimize the size of the \((X,Y)\)-cut and at the same time we want to maximize the set of vertices that remain reachable from \(X\) after removing the cut. The important \((X,Y)\)-cuts are the \((X,Y)\)-cuts that are “Pareto efficient” with respect to these two objectives: increasing the set of vertices reachable from \(X\) requires strictly increasing the size of the cut.

Let us point out that we do not want the number of vertices reachable from \(X\) to be maximal, we just want that this set of vertices is inclusionwise maximal (i.e., we have \(R \subseteq R'\) and \(|R| < |R'|\) in the definition).

The following proposition formalizes is an immediate consequence of the definition. This is the property of important \((X,Y)\)-cuts that we use in the algorithms.

**Proposition 3.7.** Let \(G\) be an undirected graph and \(X, Y \subseteq V(G)\) two disjoint sets of vertices. Let \(S\) be an \((X,Y)\)-cut and let \(R\) be the set of vertices reachable from \(X\) in \(G \cap S\). Then there is an important \((X,Y)\)-cut \(S' = \Delta(R')\) (possibly, \(S' = S\)) such that \(|S'| \leq |S|\) and \(R \subseteq R'\).

**Proof.** Let \(S^* \subseteq S\) be a minimal \((X,Y)\)-cut and let \(R^* \supseteq R\) be the set of vertices reachable from \(X\) in \(G \setminus S^*\). If \(S^*\) is an important \((X,Y)\)-cut, then we are done. Otherwise, there is an \((X,Y)\)-cut \(S' = \Delta(R')\) for some \(R' \supseteq R^* \supseteq R\) and \(|S'| \leq |S^*| \leq |S|\). If \(S'\) is an important \((X,Y)\)-cut, then we are done. Otherwise, we can repeat the argument: each time we strictly increase the set of vertices reachable from \(X\) and the size of the cut does not increase. Eventually, the process has to stop and we obtain an important \((X,Y)\)-cut. 

To check whether a given \((X,Y)\)-cut \(\Delta(R)\) is important, one needs to check whether there is another \((X,Y)\)-cut of the same size “after” \(R\), that is, whether there is an \((R \cup v, Y)\)-cut of size not larger for some \(v \in V(G) \setminus R\).

This can be done efficiently using Theorem 3.5.

**Proposition 3.8.** Given a graph \(G\), two disjoint sets \(X, Y \subseteq V(G)\), and an \((X,Y)\)-cut \(\Delta(R)\) of size \(k\), it can be tested in time \(O(k(|V(G)| + |E(G)|))\) whether \(\Delta(R)\) is an important cut.

**Proof.** Observe that \(\Delta(R)\) is an important \(X \setminus Y\) cut if and only if it is the unique minimum \((R,Y)\) cut. Therefore, if we compute the minimum \((R,Y)\)-cut \(\Delta(R_{\text{max}})\) using the algorithm of Theorem 3.5, then \(\Delta(R)\) is an important \((X,Y)\)-cut if and only if \(R = R_{\text{max}}\). \(\square\)
3.2 Important cuts

Theorem 3.4 shows that $\Delta(R_{\text{max}})$ is the unique minimum $(X,Y)$-cut that is an important $(X,Y)$-cut: we have $R_{\text{max}} \supset R$ for every other minimum $(X,Y)$-cut $\Delta(R)$. However, for sizes larger than the minimum cut size, there can be a large number of incomparable $(X,Y)$-cuts of the same size. Consider the example in Figure 3.4. Any $(X,Y)$-cut $(R)$ is an important $(X,Y)$-cut: the only way to extend the set $R$ is to move some vertex $v_i \in V(G) \setminus (R \cup Y)$ into the set $R$, but then the cut size increases by one. Therefore, the graph has exactly $\binom{n}{x}$ important $(X,Y)$-cuts of size $n + x$.

The main result of the section is a bound on the number of important cuts (of size at most $k$) and an algorithm for enumerating them. We need first the following simple observations, whose proofs are given as Exercise 3.5.

**Proposition 3.9.** Let $G$ be a graph, $X, Y \subseteq V(G)$ be two disjoint sets of vertices, and $S = \Delta(R)$ be an important $(X,Y)$-cut.

1. For every $e \in S$, the set $S \setminus \{e\}$ is an important $(X,Y)$-cut in $G \setminus e$.
2. If $S$ is an $(X',Y')$-cut for some $X' \supset X$, then $S$ is an important $(X',Y')$-cut.

The first key observation in bounding the number of important $(X,Y)$-cuts is that every important $(X,Y)$-cut is after the set $R_{\text{max}}$ of Theorem 3.4.

**Lemma 3.10.** If $\Delta(R)$ is an important $(X,Y)$-cut, then $R_{\text{max}} \subseteq R$.

**Proof.** Let us apply the submodular inequality (3.1) on $R_{\text{max}}$ and $R$:

$$d(R_{\text{max}}) + d(R) \geq d(R_{\text{max}} \cap R) + d(R_{\text{max}} \cup R).$$

Let $\lambda$ be the minimum $(X,Y)$-cut size. The first term $d_G(R_{\text{max}})$ on the left-hand side is exactly $\lambda$ (as $\Delta(R_{\text{max}})$ is a minimum $(X,Y)$-cut). Furthermore, $\Delta(R_{\text{max}} \cap R)$ is an $(X,Y)$-cut, hence we have that the first term on the right-hand side is at least $\lambda$. It follows then that the second term of the left-hand side is at least the second term on the right-hand side, that is, $d_G(R_{\text{max}}) \geq d_G(R \cup R_{\text{max}})$. If $R_{\text{max}} \not\subseteq R$, then $R_{\text{max}} \cup R$ is a proper superset of $R$. However, then the $(X,Y)$-cut $\Delta(R_{\text{max}} \cup R)$ contradicts the assumption...
that $\Delta(R)$ is an important $(X, Y)$-cut. Thus we have proved that $R_{\text{max}} \subseteq R$.

We are now ready to present the bound on the number of important cuts and the algorithm to enumerate them.

**Theorem 3.11.** Let $X, Y \subseteq V(G)$ be two disjoint sets of vertices in graph $G$, let $k \geq 0$ be an integer, and let $S_k$ be the set of all $(X, Y)$-important cuts of size at most $k$. Then $|S_k| \leq 4^k$ and $S_k$ can be constructed in time $|S_k| \cdot k \cdot (|V(G)| + |E(G)|)$.

**Proof.** We prove that there are at most $2^{2k-\lambda}$ important $(X, Y)$-cuts of size at most $k$, where $\lambda$ is the size of the smallest $(X, Y)$-cut. Clearly, this implies the upper bound $4^k$ claimed in the theorem. The statement is proved by induction on $2k - \lambda$. If $\lambda > k$, then there is no $(X, Y)$-cut of size $k$, and therefore the statement holds if $2k - \lambda < 0$. Also, if $\lambda = 0$ and $k \geq 0$, then there is a unique important $(X, Y)$-cut of size at most $k$: the empty set.

The proof is by branching on an edge $xy$ leaving $R_{\text{max}}$: an important $(X, Y)$-cut either contains $xy$ or not. In both cases, we can recurse on an instance where the measure $2k - \lambda$ is strictly smaller.

Let $\Delta(R)$ be an important $(X, Y)$-cut and let $\Delta(R_{\text{max}})$ be the minimum $(X, Y)$-cut defined by Theorem 3.4. By Lemma 3.10, we have $R_{\text{max}} \subseteq R$. As we have assumed $\lambda > 0$, there is at least one edge $xy$ with $x \in R_{\text{max}} \subseteq R$ and $y \notin R_{\text{max}}$. Then vertex $y$ is either in $R$ or not. If $y \notin R$, then $xy$ is an edge of the cut $\Delta(R)$ and then $S \setminus \{xy\}$ is an important $(X, Y)$-cut in $G' = G \setminus xy$ of size at most $k' := k - 1$ (Prop. 3.9(1)). Removing an edge can decrease the size of the minimum $(X, Y)$-cut size by at most one, hence the size $\lambda'$ of the minimum $(X, Y)$-cut in $G'$ is at least $\lambda - 1$. Therefore, $2k' - \lambda' < 2k - \lambda$ and the induction hypothesis implies that there are at most $2^{2k' - \lambda'} \leq 2^{2k - \lambda - 1}$ important $(X, Y)$-cuts of size $k'$ in $G'$, and hence at most that many important $(X, Y)$-cuts of size $k$ in $G$ that contain the edge $xy$.

Let us count now the important $(X, Y)$-cuts not containing the edge $xy$. As $R_{\text{max}} \subseteq R$, the fact that $xy$ is not in the cut implies that even $R_{\text{max}} \cup \{y\} \subseteq R$ is true. Let $X' = R_{\text{max}} \cup \{y\}$; it follows that $\Delta(R)$ is an $(X', Y)$-cut and in fact an important $(X', Y)$-cut by Prop. 3.9(2). There is no $(X', Y)$-cut $\Delta(R')$ of size $\lambda$: such a cut would be an $(X, Y)$-cut with $R_{\text{max}} \subseteq R_{\text{max}} \cup \{y\} \subseteq R'$, contradicting the definition of $R_{\text{max}}$. Thus the minimum size $\lambda'$ of an $(X', Y)$-cut is greater than $\lambda$. It follows by the induction assumption that the number of important $(X', Y)$-cuts of size at most $k$ is at most $2^{2k - \lambda'} \leq 2^{2k - \lambda - 1}$, which is a bound on the number of important $(X, Y)$-cuts of size $k$ in $G$ that do not contain $v$.

Adding the bounds in the two cases, we get the required bound $2^{2k - \lambda}$. An algorithm for enumerating all the at most $4^k$ important cuts follows from the
above proof. First, we can compute \( R_{\text{max}} \) using the algorithm of Theorem 3.5. Pick an arbitrary edge \( xy \in \Delta(R_{\text{max}}) \). Then we branch on whether edge \( xy \) is in the important cut or not, and recursively find all possible important cuts for both cases. Note that this algorithm enumerates a superset of all important cuts: by our analysis above, every important cut is found, but there is no guarantee that all the constructed cuts are important (see Exercise 3.6). Therefore, the algorithm has to be followed by a filtering phase where we use Proposition 3.8 to check for each returned cut whether it is important.

The search tree of the algorithm has at most \( 4^k \) leaves and the work to be done in each node is \( \mathcal{O}(k(|V(G)| + |E(G)|)) \). Therefore, the total running time of the branching algorithms is \( \mathcal{O}(4^k \cdot k(|V(G)| + |E(G)|)) \) and returns at most \( 4^k \) cuts. This is followed by the filtering phase, which takes time \( \mathcal{O}(4^k \cdot k(|V(G)| + |E(G)|)) \).

The following combinatorial bound is very helpful when analyzing algorithms based on branching by selecting an important cut (see Theorem 3.16). Its proof is essentially the same as the proof of Theorem 3.11 with an appropriate induction statement (see Exercise 3.7).

**Lemma 3.12.** Let \( G \) be an undirected graph and let \( X, Y \subseteq V(G) \) be two disjoint sets of vertices. If \( S \) is the set of all important \((X,Y)\)-cuts, then \( \sum_{S \in S} 4^{-|S|} \leq 1 \).

The reader might wonder how tight the bound \( 4^k \) in Theorem 3.11 is. The example in Figure 3.4 already showed that the number of important cuts of size at most \( k \) can be exponential in \( k \). We can show that the bound \( 4^k \) is tight up to polynomial factors; in particular, the base of the exponent has to be 4. Consider a rooted complete binary tree \( T \) with \( n > k \) levels; let \( X \) contain only the root and \( Y \) contain all the leaves (see Figure 3.5). Then every rooted full binary subtree \( T' \) with \( k \) leaves gives rise to an important \((X,Y)\)-cut of size \( k \). Indeed, the cut \( \Delta(R) \) obtained by removing the edges incident to the leaves of \( T' \) is an important \((X,Y)\)-cut, as moving more vertices to \( R \) would clearly increase the cut size. It is well known that the number of full subtrees with exactly \( k \) leaves of a rooted complete binary tree is precisely the Catalan number \( C_k = \frac{(2k)!}{k!(k+1)!} \geq 4^k/k^{O(1)} \).

As an application of the bound important cuts, we can prove the following surprisingly simple, but still nontrivial combinatorial result: only a bounded number of edges incident to \( Y \) are relevant to \((X,Y)\)-cuts of size at most \( k \).

**Lemma 3.13.** Let \( G \) be an undirected graph and let \( X, Y \subseteq V(G) \) be two disjoint sets of vertices. The union of all minimal \((X,Y)\)-cuts of size at most \( k \) contains at most \( k \cdot 4^k \) edges incident to \( Y \).

**Proof.** Let \( F \) contain an edge \( e \) if it is incident to a vertex of \( Y \) and it appears in an \((X,Y)\)-important cut of size at most \( k \). By Theorem 3.11, we have \( |F| \leq 4^k \cdot k \). We claim that if an edge \( e \) incident to \( Y \) is not in \( F \), then it cannot appear in any minimal \((X,Y)\)-cut of size at most \( k \). Suppose for
contradiction that $e$ appears in an $(X, Y)$-cut $\Delta(R)$ of size at most $k$. If this is an important cut, then $e$ is in $F$. Otherwise, Proposition 3.47 implies that there is an important $(X, Y)$-cut $\Delta(R')$ of size at most $k$ with $R \subseteq R'$. Edge $e$ has an endpoint $x \in R$ and an endpoint $y \notin R$. As $R \cap Y = \emptyset$, endpoint $y$ of $e$ has to be in $Y$. Now we have $x \in R \subseteq R'$ and $y \in Y \subseteq V(G) \setminus R'$, hence $e$ is an edge of $\Delta(R')$ as well, implying $e \in F$. □

3.3 Multiway Cut

Let $G$ be a graph and $T \subseteq V(G)$ be a set of terminals. An edge multiway cut is a set $S$ of edges such that every component of $G \setminus S$ contains at most one vertex of $T$. Given a graph $G$, terminals $T$, and an integer $k$, the Edge Multiway Cut problem asks if a multiway cut of size at most $k$ exists. If $|T| = 2$, that is, $T = \{t_1, t_2\}$, then Edge Multiway Cut is the problem of finding a $(t_1, t_2)$-cut of size at most $k$, hence it is polynomial-time solvable. However, the problem becomes NP-hard already for $|T| = 3$ terminals [67].

In this section, we show that Edge Multiway Cut is FPT parameterized by $k$. The following observation connects Edge Multiway Cut and the concept of important cuts:

Lemma 3.14 (Pushing Lemma for Edge Multiway Cut). Let $t \in T$ be a terminal that is not separated from $T \setminus t$ in an undirected graph $G$. If
G has a multiway cut $S$, then it also has a multiway cut $S^*$ with $|S^*| \leq |S|$ such that $S^*$ contains an important $(t, T \cap t)$-cut.

Proof. Let $R$ be the component of $G \setminus S$ containing $t$. As $S$ is a multiway cut, $R$ is disjoint from $T \setminus t$ and hence $S_R = \Delta(R)$ is a $(t, T \setminus t)$-cut contained in $S$. By Proposition 3.47, there is an important $(t, T \setminus t)$-cut $S' = \Delta(R')$ with $R \subseteq R'$ and $|S'| \leq |S_R|$ (see Figure 3.6). We claim that $S^* = (S \setminus S_R) \cup S'$, which has size at most $|S|$, is also a multiway cut, proving the statement of the lemma.

Replacing the $(t, T \setminus t)$-cut $S_R$ in the solution with an important $(t, T \setminus t)$-cut $S'$ cannot break the solution: as it is closer to $T \setminus t$, it can be even more helpful in separating the terminals in $T \setminus t$ from each other.

Clearly, there is no path between $t$ and any other terminal of $T \setminus t$ in $G \setminus S^*$, as $S' \subseteq S^*$ is a $(t, T \setminus t)$-cut. Suppose therefore that there is a path $P$ between two distinct terminals $t_1, t_2 \in T \setminus t$ in $G \setminus S^*$. If $P$ goes through a vertex of $R \subseteq R'$, then it goes through at least one edge of the $(t, T \setminus t)$ cut $S' = \Delta(R')$, which is a subset of $S^*$, a contradiction. Therefore, we may assume that $P$ is disjoint from $R$. Then $P$ does not go through any edge of $S_R = \Delta(R)$ and therefore the fact that $P$ is disjoint from $S^*$ implies that it is disjoint from $S$ as well, contradicting the assumption that $S$ is a multiway cut.

Using this observation, we can solve the problem by branching on the choice of an important cut and including it into the solution:
Theorem 3.15. **Edge Multiway Cut** can be solved in time $O(4^k \cdot k \cdot (|V(G)| + |E(G)|))$.

**Proof.** We solve the problem by a recursive branching algorithm. If all the terminals are separated from each other, then we are done. Otherwise, let $t \in T$ be a terminal not separated from the rest of the terminals. Let us use the algorithm of Theorem 3.11 to construct the set $S_k$ consisting of every important $(t, T \setminus t)$-cut of size at most $k$. By Lemma 3.14, there is a solution that contains one of these cuts. Therefore, we branch on the choice of one of these cuts: for every important cut $S_0 \subseteq S_k$, we recursively solve the **Edge Multiway Cut** instance $(G \setminus S_0, T, k - |S_0|)$. If one of these branches returns a solution $S$, then clearly $S \cup S_0$ is a multiway cut of size at most $k$ in $G$.

The correctness of the algorithm is clear from Lemma 3.14. We claim that the search tree explored by the algorithm has at most $4^k$ leaves. We prove this by induction on $k$, thus let us assume that the statement is true for every value less than $k$. This means that we know that the recursive call $(G \setminus S', T, k - |S'|)$ explores a search tree with at most $4^k \cdot |S'|$ leaves. Using Lemma 3.12, we can bound the number of leaves of the search tree by

$$\sum_{S' \in S_k} 4^{k-|S'|} \leq 4^k \cdot \sum_{S' \in S_k} 4^{-|S'|} \leq 4^k.$$

**TODO: total work**

A slight generalization of **Edge Multiway Cut** is **Edge Multiway Cut for Sets**, where the input is a graph $G$, pairwise disjoint sets $T_1, \ldots, T_p$, and an integer $k$; the task is to find a set $S$ of edges that pairwise separates these sets from each other, that is, there is no $T_i - T_j$ path in $G \setminus S$ for any $i \neq j$. **Edge Multiway Cut for Sets** can be reduced to **Edge Multiway Cut** simply by consolidating each set into a single vertex.

**Theorem 3.16.** **Edge Multiway Cut for Sets** can be solved in time $O(4^k \cdot k \cdot (|V(G)| + |E(G)|))$.

**Proof.** We construct a graph $G'$ as follows. The vertex set of $G'$ is $(V(G) \setminus \bigcup_{i=1}^p T_i) \cup T$, where $T = \{v_1, \ldots, v_p\}$. For every edge $xy \in E(G)$, there is a corresponding edge in $G'$: if endpoint $x$ or $y$ is in $T$, then it is replaced by $v_i$. It is easy to see that there is a one-to-one correspondence between the solutions of the **Edge Multiway Cut for Sets** instance $(G, (T_1, \ldots, T_p), k)$ and the **Edge Multiway Cut** instance $(G', T, k)$.

Another well-studied generalization of **Edge Multiway Cut** can be obtained if, instead of requiring that all the terminals are separated from each other, we require that a specified set of pairs of terminals are separated from each other. The input in the **Edge Multicut** problem is a graph $G$, pairs $(s_1, t_1), \ldots, (s_\ell, t_\ell)$, and an integer $k$; the task is to find a set $S$ of at most $k$ edges such that $G \setminus S$ has no $s_i - t_i$ path for any $1 \leq i \leq \ell$. If we have
3.4 \((p, q)\)-Clustering

A bounded number of pairs of terminals, \textsc{Edge Multicut} can be reduced to \textsc{Edge Multiway Cut for Sets}: we can guess how the components of \(G \setminus S\) for the solution \(S\) partition the \(2\ell\) vertices \(s_i, t_i\) \((1 \leq i \leq \ell)\) and solve the resulting \textsc{Edge Multiway Cut for Sets} instance. The \(2\ell\) terminals can be partitioned in at most \((2\ell)^{2\ell}\) different ways, thus we can reduce \textsc{Edge Multicut} into at most \((k + 1)^{2\ell}\) instances of \textsc{Edge Multiway Cut for Sets}. This shows that \textsc{Edge Multicut} is FPT with combined parameters \(k\) and \(\ell\). We may also observe that the removal of at most \(k\) edges can partition a component of \(G\) into at most \((k+1)\) components, hence we need to consider at most \((k+1)^{2\ell}\) partitions.

**Theorem 3.17.** \textsc{Edge Multicut} can be solved in time \((2\ell)^{2\ell} \cdot 4^k \cdot k(|V(G)| + |E(G)|)\) or in time \((k + 1)^{2\ell} \cdot 4^k \cdot k(|V(G)| + |E(G)|)\).

It is a more challenging question whether the problem is FPT parameterized by \(k\) (the size of the solution) only. The proof of this requires several non-trivial technical steps and is beyond the scope of this chapter. We remark that one of the papers proving the fixed-parameter tractability of \textsc{Edge Multicut} was the paper introducing the random sampling of important separators technique [165].

**Theorem 3.18 ([29, 165]).** \textsc{Edge Multicut} can be solved in time \(2^{O(k^3)} \cdot n^{O(1)}\).

### 3.4 \((p, q)\)-Clustering

The typical goal in clustering problems is to group a set of objects such that, roughly speaking, similar objects appear together in the same group. There are many different ways of defining the input and the objective of clustering problems: depending on how similarity of objects is interpreted, what constraints we have on the number and size of the clusters, and how the quality of the clustering is measured, we can define very different problems of this form. Often a clustering problem is defined in terms of a (weighted) graph: the objects are the vertices and the presence of a (weighted) edge indicates similarity of the two objects and the absence of an edge indicates that the two objects are not considered to be similar.

In this section, we present an algorithm for a particular clustering problem on graphs that is motivated by partitioning circuits into several field programmable gate arrays (FPGAs). We say that a set \(C \subseteq V(G)\) is a \((p, q)\)-cluster if \(|C| \leq p\) and \(d(C) \leq q\). A \((p, q)\)-partition of \(G\) is a partition of \(V(G)\) into \((p, q)\)-clusters. Given a graph \(G\) and integers \(p\) and \(q\), the \((p, q)\)-\textsc{Partition} problem asks if \(G\) has a \((p, q)\)-partition. The main result of this section is showing that \((p, q)\)-\textsc{Partition} is FPT parameterized by \(q\). The proof is based on the technique of random sampling of important separators and serves as a self-contained demonstration of this technique.
An uncrossing argument shows that the trivial necessary condition that every vertex is in a \((p, q)\)-cluster is also a sufficient condition.

Checking whether \(v\) is in a \((p, q)\)-cluster is trivial in \(n^{O(q)}\) time, but an FPT algorithm parameterized by \(q\) is more challenging.

The crucial observation is that a minimal \((p, q)\)-cluster containing \(v\) is surrounded by important cuts.

A randomized reduction allows us to reduce finding a \((p, q)\)-cluster containing \(v\) to a knapsack-like problem.

The derandomization of the reduction uses the standard technique of splitters.

A necessary condition for the existence of \((p, q)\)-partition is that for every vertex \(v \in V(G)\) there exists a \((p, q)\)-cluster that contains \(v\). Very surprisingly, it turns out that this trivial necessary condition is actually sufficient for the existence of a \((p, q)\)-partition. The proof (Lemma 3.20) needs a variant of submodularity: we say that a set function \(f : 2^{V(G)} \to \mathbb{R}\) is \textit{posimodular} if it satisfies the following inequality for every \(A, B \subseteq V(G)\):

\[
f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A). \quad (3.2)
\]

This inequality is very similar to the submodular inequality (3.1) and the proof that \(d_G\) has this property is a case analysis similar to the proof of Theorem 3.3. The only difference is that edges of type 4 contribute 2 to the left-hand side and 0 to the right-hand side, while edges of type 5 contribute 2 to both sides.

\textbf{Theorem 3.19.} \textit{The function} \(d_G\) \textit{is posimodular for every undirected graph} \(G\).

We are now ready to prove that every vertex being in a \((p, q)\)-cluster is a sufficient condition for the existence of a \((p, q)\)-partition.

\textbf{Lemma 3.20.} \textit{Let} \(G\) \textit{be an undirected graph and let} \(p, q \geq 0\) \textit{be two integers. If every} \(v \in V(G)\) \textit{is contained in some} \((p, q)\)-\textit{cluster, then} \(G\) \textit{has a} \((p, q)\)-\textit{partition. Furthermore, given a set of} \((p, q)\)-\textit{clusters} \(C_1, \ldots, C_n\) \textit{whose union is} \(V(G)\), \textit{a} \((p, q)\)-\textit{partition can be found in polynomial time.}

\textit{Proof.} Let us consider a collection \(C_1, \ldots, C_n\) of \((p, q)\)-clusters whose union is \(V(G)\). If the sets are pairwise disjoint, then they form a partition of \(V(G)\) and we are done. If \(C_i \subseteq C_j\), then the union remains \(V(G)\) even after throwing away \(C_i\). Thus we can assume that no set is contained in another.

The posimodularity of the function \(d_G\) allows us to uncross two clusters \(C_i\) and \(C_j\) if they intersect.
Suppose that $C_i$ and $C_j$ intersect. Now either $d(C_i) \geq d(C_i \setminus C_j)$ or $d(C_j) \geq d(C_j \setminus C_i)$ must be true; it is not possible that both $d(C_i) < d(C_i \setminus C_j)$ and $d(C_j) < d(C_j \setminus C_i)$ hold, as this would violate the posimodularity of $d$. Suppose that $d(C_j) \geq d(C_j \setminus C_i)$. Now the set $C_j \setminus C_i$ is also a $(p, q)$-cluster: we have $d(C_j \setminus C_i) \leq d(C_j) \leq q$ by assumption and $|C_j \setminus C_i| < |C_i| \leq p$. Thus we can replace $C_j$ by $C_j \setminus C_i$ in the collection: it will remain true that the union of the clusters is $V(G)$. Similarly, if $d(C_i) \geq d(C_i \setminus C_j)$, then we can replace $C_i$ by $C_i \setminus C_j$.

Repeating these steps (throwing away subsets and resolving intersections), we eventually arrive at a pairwise-disjoint collection of $(p, q)$-clusters. Each step decreases the number of cluster pairs $(C_i, C_j)$ that have non-empty intersection. Therefore, this process terminates after a polynomial number of steps. \hfill $\square$

The proof of Lemma 3.20 might suggest that we can obtain a partition by simply taking, for every vertex $v$, a $(p, q)$-cluster $C_v$ that is inclusionwise minimal with respect to containing $v$. However, such clusters can still cross. For example, consider a graph on vertices $a, b, c, d$ where every pair of vertices expect $a$ and $d$ are adjacent. Suppose that $p = 3$, $q = 2$. Then $\{a, b, c\}$ is a minimal cluster containing $b$ (as more than two edges are going out of each of $\{b\}, \{b, c\}$, and $\{a, b\}$) and $\{b, c, d\}$ is a minimal cluster containing $c$. Thus unless we choose the minimal clusters more carefully in a coordinated way, they are not guaranteed to form a partition. In other words, there are two symmetric solutions $\{\{a, b, c\}, \{d\}\}$ and $\{\{a\}, \{b, c, d\}\}$ for the problem, and the clustering algorithm has to break this symmetry somehow.

In light of Lemma 3.20, it is sufficient to find a $(p, q)$-cluster $C_v$ for each vertex $v \in V(G)$. If there is a vertex $v$ for which there is no such cluster $C_v$, then obviously there is no $(p, q)$-partition; if we have such a $C_v$ for every vertex $v$, then Lemma 3.20 gives us a $(p, q)$-partition in polynomial time. Therefore, in the rest of the section, we are studying the $(p, q)$-CLUSTER problem, where, given a graph $G$, a vertex $v \in V(G)$, and integers $p$ and $q$, it has to be decided if there is a $(p, q)$-cluster containing $v$.

For fixed $q$, the $(p, q)$-CLUSTER problem can be solved by brute force: enumerate every set $F$ of at most $q$ edges and check if the component of $G \setminus F$ containing $v$ is a $(p, q)$-cluster. If $C_v$ is a $(p, q)$-cluster containing $v$, then we find it when $F = \Delta(C_v)$ is considered by the enumeration procedure.

**Theorem 3.21.** For every fixed $q$, there is an $n^{O(q)}$ time algorithm for $(p, q)$-CLUSTER.

The main result of the section is showing that it is possible to solve $(p, q)$-CLUSTER (and hence $(p, q)$-PARTITION) more efficiently: in fact, it is fixed-parameter tractable parameterized by $q$. By Lemma 3.20, all we need to show is that $(p, q)$-CLUSTER is fixed-parameter tractable parameterized by $q$. We introduce a somewhat technical variant of $(p, q)$-CLUSTER, the SATELLITE
Problem, which is polynomial-time solvable. Then we show how to solve $(p, q)$-Cluster using an algorithm for the Satellite Problem.

The input of the Satellite Problem is a graph $G$, integers $p$, $q$, a vertex $v \in V(G)$, and a partition $(V_0, V_1, \ldots, V_r)$ of $V(G)$ such that $v \in V_0$ and there is no edge between $V_i$ and $V_j$ for any $1 \leq i < j \leq r$. The task is to find a $(p, q)$-cluster $C$ satisfying $V_0 \subseteq C$ such that for every $1 \leq i \leq r$, either $C \cap V_i = \emptyset$ or $V_i \subseteq C$ (see Figure 3.7).

Since the sets $\{V_i\}$ form a partition of $V(G)$, we have $r \leq n$. For every $V_i$ ($1 \leq i \leq r$), we have to decide whether to include or exclude it from the solution cluster $C$. If we exclude $V_i$ from $C$, then $d(C)$ increases by $d(V_i)$, the number of edges between $V_0$ and $V_i$. If we include $V_i$ into $C$, then $|C|$ increases by $|V_i|$.

To solve Satellite Problem, we need to solve the knapsack-like problem of including sufficiently many $V_i$’s such that $d(C) \leq q$, but not including too many to ensure $|C| \leq p$.

**Lemma 3.22.** The Satellite Problem can be solved in polynomial time.

**Proof.** For a subset $S$ of $\{1, \ldots, r\}$, we define $C(S) = V_0 \cup \bigcup_{i \in S} V_i$. Notice that $d(C(S)) = d(V_0) - \sum_{i \in S} d(V_i)$. Hence, we can reformulate the Satellite Problem as finding a subset $S$ of $\{1, \ldots, r\}$ such that $\sum_{i \in S} d(V_i) \geq d(V_0) - q$ and $\sum_{i \in S} |V_i| \leq p - |V_0|$. Thus, we can associate with every $i$ an item with value $d(V_i)$ and weight $|V_i|$. The objective is to find a set of items with total value at least $v_{\text{target}} := (V_0) - q$ and total weight at most $w_{\text{max}} := p - |V_0|$. This problem is known as Knapsack and can be solved in polynomial time by a classical dynamic programming algorithm in time polynomial in the number $r$ of items and the maximum weight $w_{\text{max}}$ (assuming that the weights are positive integers). In our case, both $r$ and $w_{\text{max}}$ are polynomial in the size of the graph $G$. 

**Fig. 3.7:** Instance of Satellite Problem with a solution $C$. Excluding $V_2$ and $V_4$ from $C$ decreased the size of $C$ by the gray area, but increased $d(C)$ by the red edges.
of the input, hence a polynomial-time algorithm for \textsc{Satellite Problem} follows.

We briefly sketch how the dynamic programming algorithm works. For $0 \leq i \leq r$ and $0 \leq j \leq w_{\text{max}}$, we define $T[i, j]$ to be the maximum value of a set $S \subseteq \{0, 1, \ldots, i\}$ that has total weight at most $j$. By definition, $T[0, j] = 0$ for every $j$. Assuming that we have computed $T[i - 1, j]$ for every $j$, we can then compute $T[i, j]$ for every $j$ using the following recurrence relation:

$$T[i, j] = \max \{T[i - 1, j], T[i - 1, j - |V_i|] + d(V_i)\}.$$ 

That is, the optimal set $S$ either does not include item $i$ (in which case it is also an optimal set for $T[i - 1, j]$), or includes item $i$ (in which case removing item $i$ decreases the value by $d(V_i)$, decreases the weight bound by $|V_i|$, and what remains should be optimal for $T[i - 1, j - |V_i|]$). After computing every value $T[i, j]$, we can check whether $v$ is in a suitable cluster by checking whether $T[r, w_{\text{max}}] \geq v_{\text{target}}$ holds.

What remains is to be shown is how to reduce $(p, q)$-Cluster to the \textsc{Satellite Problem}. We first present a randomized version of this reduction as it is cleaner and conceptually simpler. Then we discuss how the standard technique of splitters can be used to derandomize the reduction.

The following definition connects the notion of important cuts with our problem.

**Definition 3.23.** We say that a set $X \subseteq V(G)$, $v \notin X$ is \textit{important} if

1. $d(X) \leq q$,
2. $G[X]$ is connected,
3. there is no $Y \supseteq X$, $v \notin Y$ such that $d(Y) \leq d(X)$ and $G[Y]$ is connected.

It is easy to see that $X$ is an important set if and only if $\Delta(X)$ is an important $(u, v)$-cut of size at most $q$ for every $u \in X$. Thus we can use Theorem 3.11 to enumerate every important set, and Lemma 3.12 to give an upper bound on the number of important sets. The following lemma establishes the connection between important sets and finding $(p, q)$-clusters: we can assume that the components of $G \setminus C$ for the solution $C$ are important sets.

**Lemma 3.24.** Let $C$ be an inclusionwise minimal $(p, q)$-cluster containing $v$. Then every component of $G \setminus C$ is an important set.

**Proof.** Let $X$ be a component of $G \setminus C$. It is clear that $X$ satisfies the first two properties of Definition 3.23 (note that $\Delta(X) \subseteq \Delta(C)$ implies $d(X) \leq d(C) \leq q$). Thus let us suppose that there is a $Y \supseteq X$, $v \notin Y$ such that $d(Y) \leq d(X)$ and $G[Y]$ is connected. Let $C' := C \setminus Y$. Note that $C'$ is a proper subset of $C$: every neighbor of $X$ is in $C$, thus a connected superset of $X$ has to contain at least one vertex of $C$. It is easy to see that $C'$ is a $(p, q)$-cluster: we have $\Delta(C') \subseteq (\Delta(C) \setminus \Delta(X)) \cup \Delta(Y)$ and therefore $d(C') \leq d(C) - d(X) + d(Y) \leq d(C) \leq q$ and $|C'| \leq |C| \leq p$. This contradicts the minimality of $C$. \hfill $\Box$
We are now ready to present the randomized version of the reduction.

**Lemma 3.25.** Given a graph $G$, vertex $v \in V(G)$, and integers $p$ and $q$, we can construct in time $2^{O(q)} \cdot n^{O(1)}$ an instance $I$ of the Satellite Problem such that

- If some $(p,q)$-cluster contains $v$, then $I$ is a yes-instance with probability $2^{-O(q)}$.
- If there is no $(p,q)$-cluster containing $v$, then $I$ is a no-instance.

**Proof.** For every $u \in V(G)$, $u \neq v$, let us use the algorithm of Theorem 3.11 to enumerate every important $(u,v)$-cut of size at most $q$. For every such cut $S$, let us put the component $K$ of $G \setminus S$ containing $u$ into the collection $\mathcal{X}$. Note that the same component $K$ can be obtained for more than one vertex $u$, but we put only one copy into $\mathcal{X}$.

Let $\mathcal{X}'$ be a subset of $\mathcal{X}$, where each member $K$ of $\mathcal{X}$ is chosen with probability $4^{-d(K)}$ independently at random. Let $Z$ be the union of the sets in $\mathcal{X}'$, let $V_1, \ldots, V_r$ be the connected components of $G[Z]$, and let $V_0 = V(G) \setminus Z$. It is clear that $V_0, V_1, \ldots, V_r$ describe an instance $I$ of the Satellite Problem, and a solution for $I$ gives a $(p,q)$-cluster containing $v$. Thus we only need to show that if there is a $(p,q)$-cluster $C$ containing $v$, then $I$ is a yes-instance with probability $2^{-O(q)}$.

We show that the reduction works if the union $Z$ of the selected important sets satisfies two constraints: it has to cover every component of $G \setminus C$ and it has to be disjoint from the vertices on the boundary of $C$. The probability of the event that these constraints are satisfied is $2^{-O(q)}$.

Let $C$ be an inclusionwise minimal $(p,q)$-cluster containing $v$. Let $S$ be the set of vertices on the boundary of $C$, i.e., the vertices of $C$ incident to $\Delta(C)$. Let $K_1, \ldots, K_i$ be the components of $G \setminus C$. Note that every edge of $\Delta(C)$ enters some $K_i$, thus $\sum_{i=1}^{t} d(K_i) = d(C) \leq q$. By Lemma 3.24, every $K_i$ is an important set, and hence it is in $\mathcal{X}$. Consider the following two events:

- **(E1)** Every component $K_i$ of $G \setminus C$ is in $\mathcal{X}'$ (and hence $K_i \subseteq Z$).
- **(E2)** $Z \cap S = \emptyset$.

The probability that (E1) holds is $\prod_{i=1}^{t} 4^{-d(K_i)} = 4^{-\sum_{i=1}^{t} d(K_i)} \geq 4^{-q}$. Event (E2) holds if for every $w \in S$, no set $K \in \mathcal{X}'$ with $w \in K$ is selected into $\mathcal{X}'$. It follows directly from the definition of important cuts that for every $K \in \mathcal{X}'$ with $w \in K$, the set $\Delta(K)$ is an important $(w,v)$-cut. Thus by Lemma 3.12, $\sum_{K \in \mathcal{X}, w \in K} 4^{-d(K)} \leq 1$. The probability that $Z \cap S = \emptyset$ can be bounded by
In the first inequality, we use that every term is less than 1 and every term on the right hand side appears at least once on the left hand side; in the second inequality, we use that $1 + x \geq \exp(x/(1 + x))$ for every $x > -1$. Events (E1) and (E2) are independent: (E1) is a statement about the selection of a subcollection $A \subseteq \mathcal{X}$ of at most $q$ sets that are disjoint from $S$, while (E2) is a statement about not selecting any member of a subcollection $B \subseteq \mathcal{X}$ of at most $|S| \cdot 4^d$ sets intersecting $S$. Thus by probability $2^{-O(q)}$, both (E1) and (E2) hold.

Suppose that both (E1) and (E2) hold, we show that instance $I$ of the Satellite Problem is a yes-instance. In this case, every component $K_i$ of $G \setminus C$ is a component $V_j$ of $G[Z]$: $K_i \subseteq Z$ by (E1) and every neighbor of $K_i$ is outside $Z$. Thus $C$ is a solution of $I$, as it can be obtained as the union of $V_0$ and some components of $G[Z]$. \(\square\)

Lemma 3.25 gives a randomized reduction from $(p, q)$-Cluster to Satellite Problem, which is polynomial-time solvable (Lemma 3.22). Therefore, there is a randomized algorithm for $(p, q)$-Cluster with running time $2^{O(q)} \cdot p^{O(1)}$ and success probability $p_{\text{correct}} = 2^{-O(q)}$. By repeating the algorithm $[1/p_{\text{correct}}] = 2^{O(q)}$ times, the probability of a false answer decreases to $(1 - p_{\text{correct}})^{[1/p_{\text{correct}}]} \leq 1/e$ (using $1 - x \leq e^{-x}$).

**Corollary 3.26.** There is a $2^{O(q)} \cdot p^{O(1)}$ time randomized algorithm for $(p, q)$-Cluster with constant error probability.

To derandomize the proof of Lemma 3.25 and to obtain a deterministic version of Corollary 3.26, we use the standard technique of splitters. We present here only a simpler version of the derandomization, where the dependence on $q$ in the running time is of the form $2^{O(q^2)}$. It is possible to improve this dependence to $2^{O(q)}$, matching the bound in Corollary 3.26. However, the details are somewhat technical and we omit here the description of this improvement.

An $(n, k, k^2)$-splitter is a family of functions from $[n]$ to $[k^2]$ such that for any subset $X \subseteq [n]$ with $|X| = k$, one of the functions $f$ in the family is injective on $X$, that is, $f(x_1) \neq f(x_2)$ for any $x_1 \neq x_2$ in $X$. The family containing all $(k^2)^n$ functions from $[n]$ to $[k^2]$ is clearly an $(n, k, k^2)$-splitter. However, it is possible to construct splitters of significantly smaller size: polynomial in $k$ and logarithmic in $n$.
Theorem 3.27 ([?]). For every \( n, k \geq 1 \), it is possible to construct in time polynomial in \( n \) and \( k \) an \((n, k, k^2)\)-splitter of size \( O(k^6 \log k \log n) \).

The main idea of the derandomization is to replace the random selection of the subfamily \( X' \) by a deterministic selection based on the functions in a splitter family.

Theorem 3.28. \((p, q)\)-Cluster can be solved in time \( 2^{O(q^2)} \cdot n^{O(1)} \).

**Proof.** In the algorithm of Lemma 3.25, a random subset of a universe \( X \) of size \( s = |X| \leq 4^q \cdot n \) is selected. If the \((p, q)\)-Cluster problem has a solution \( C \), then there is a collection \( A \subseteq X \) of at most \( a := q \cdot 4^q \) sets and a collection \( B \subseteq X \) of at most \( b := q \cdot 4^q \) sets such that if every set in \( A \) is selected and no set in \( B \) is selected, then \((E1)\) and \((E2)\) hold. Instead of selecting a random subset, we try every function \( f \) in an \((s, a + b, (a + b)^2)\)-splitter family and every subset \( F \subseteq [(a + b)^2] \) of size \( a \) (there are \( \binom{(a+b)^2}{a} = 2^{O(q^2)} \) such sets \( F \)). For a particular choice of \( f \) and \( F \), we select those sets \( S \in X \) into \( X' \) for which \( f(S) \in F \). The size of the splitter family is \( 2^{O(q)} \cdot \log n \) and the number of possibilities for \( F \) is \( 2^{O(q^2)} \). Therefore, we construct \( 2^{O(q^2)} \cdot \log n \) instances of the Satellite Problem.

By the definition of the splitter, there will be a function \( f \) that is injective on \( A \cup B \), and there is a subset \( F \) such that \( f(S) \in F \) for every set \( S \) in \( A \) and \( f(S) \notin F \) for every set \( S \) in \( B \). For such an \( f \) and \( F \), the selection will ensure that \((E1)\) and \((E2)\) hold. This means that the constructed instance of the Satellite Problem corresponding to \( f \) and \( F \) has a solution as well. Thus solving every constructed instance of the Satellite Problem in polynomial time gives a \( 2^{O(q^2)} \cdot n^{O(1)} \) algorithm for \((p, q)\)-Cluster. \( \square \)

One can obtain a more efficient derandomization by a slight change of the construction in the proof of Theorem 3.28 and a more careful analysis. This gives a deterministic algorithm with \( 2^{O(q)} \) dependence on \( q \).

Theorem 3.29 ([149]). \((p, q)\)-Cluster can be solved in time \( 2^{O(q)} \cdot n^{O(1)} \).

Finally, we have shown in Lemma 3.20 that \((p, q)\)-Partition reduces to \((p, q)\)-Cluster, hence fixed-parameter tractability follows for \((p, q)\)-Partition as well.

Theorem 3.30. \((p, q)\)-Partition can be solved in time \( 2^{O(q)} \cdot n^{O(1)} \).

### 3.5 Directed Graphs

Problems on directed graphs are notoriously more difficult than problems on undirected graphs. This phenomenon has been observed equally often in
the area of polynomial-time algorithms, approximability, and fixed-parameter tractability. Let us see if the techniques based on important cuts survive the generalization to directed graphs. We denote by $\Delta^+_G(R)$ be the set of edges starting in $R$ and ending in $V(G) \setminus R$. As for undirected graphs, every minimal $(X,Y)$-cut $S$ can be expressed as $\Delta^+_G(R)$ for some $X \subseteq R \subseteq V(G) \setminus R$. Important cuts can be defined analogously for directed graphs.

**Definition 3.31.** Let $G$ be a directed graph and let $X,Y \subseteq V(G)$ be two disjoint sets of vertices. Let $S \subseteq E(G)$ be an $(X,Y)$-cut, and let $R$ be the set of vertices reachable from $X$ in $G \setminus S$. We say that $S$ is an important $(X,Y)$-cut if it is minimal and there is no $(X,Y)$-cut $S'$ with $|S'| \leq |S|$ such that $R \subseteq R'$, where $R$ is the set of vertices reachable from $X$ in $G \setminus S'$.

Proposition 3.47 generalizes to directed graphs in a straightforward way.

**Proposition 3.32.** Let $G$ be a directed graph and $X,Y \subseteq V(G)$ two disjoint sets of vertices. Let $S$ be an $(X,Y)$-cut and let $R$ be the set of vertices reachable from $X$ in $G \cap S$. Then there is an important $(X,Y)$-cut $S' = \Delta^+_G(R)$ (possibly, $S' = S$) such that $|S'| \leq |S|$ and $R \subseteq R'$.

We state without proof that the bound of $4^k$ of Theorem 3.11 holds also for directed graphs.

**Theorem 3.33.** Let $X,Y \subseteq V(G)$ be two sets of vertices in a directed graph $G$, let $k \geq 0$ be an integer, and let $S_k$ be the set of all $(X,Y)$-important cuts of size at most $k$. Then $|S_k| \leq 4^k$ and $S_k$ can be constructed in time $|S_k| \cdot k(|V(G)| + |E(G)|)$.

Also, an analog of the bound Lemma 3.12 holds for directed important cuts.

**Lemma 3.34.** Let $G$ be a directed graph and let $X,Y \subseteq V(G)$ be two disjoint sets of vertices. If $S$ is the set of all important $(X,Y)$-cuts, then $\sum_{S \in S} 4^{-|S|} \leq 1$.

Given a directed graph $G$, a set $T \subseteq V(G)$ of terminals, and an integer $k$, the Directed Edge Multiway Cut problem asks for a set $S$ of at most $k$ edges such that $G \setminus S$ has no directed $t_1 \to t_2$ path for any two distinct $t_1,t_2 \in T$. Theorem 3.33 gives us some hope that we would be able to use the techniques of Section 3.3 for undirected Edge Multiway Cut also for directed graphs. However, the Pushing Lemma (Lemma 3.14) is not true on directed graphs, even for $|T| = 2$; see Figure 3.8 for a simple counterexample where the unique minimum multiway cut does not use any important cut between the two terminals. The particular point where the proof of Lemma 3.14 breaks down in the directed setting is that a $T \setminus t \to t$ path may appear after replacing $S$ with $S^*$. This means that a straightforward generalization of Theorem 3.16 to directed graphs is not possible. Nevertheless, using different arguments (in particular, using the random sampling of important separators technique), it is possible to show that Directed Edge Multiway Cut is FPT parameterized by the size $k$ of the solution.
Fig. 3.8: The unique important \((t_1, t_2)\)-cut is the edge \((b, t_2)\), the unique important \((t_2, t_1)\)-cut is the edge \((b, t_1)\), but the unique minimum edge multiway cut is the edge \(\{a, b\}\).

**Theorem 3.35 ([46, 47]).** Directed Edge Multiway Cut can be solved in time \(2^{O(k^2)} \cdot n^{O(1)}\).

The input of the Directed Edge Multicut problem is a directed graph \(G\), a set of pairs \((s_1, t_1), \ldots, (s_\ell, t_\ell)\), and an integer \(k\), the task is to find a set \(S\) of at most \(k\) edges such that \(G \setminus S\) has no directed \(s_i \rightarrow t_i\) path for any \(1 \leq i \leq k\). This problem is more general than Directed Edge Multiway Cut: the requirement that no terminal can be reached from any other terminal in \(G \setminus S\) can be expressed by a set of \(|T|(|T| - 1)\) pairs \((s_i, t_j)\).

In contrast to the undirected version (Theorem 3.18), Directed Edge Multicut is \(W[1]\)-hard parameterized by \(k\), even on directed acyclic graphs. But the problem is interesting even for small values of \(\ell\). The argument of Theorem 3.17 (essentially, guessing how the components of \(G \setminus S\) partition the terminals) no longer works, as the relation of the terminals in \(G \setminus S\) can be more complicated in a directed graph. The case \(\ell = 2\) can be reduced to Directed Edge Multiway Cut in a simple way (see Exercise 3.11), thus Theorem 3.35 implies that Directed Edge Multicut for \(\ell = 2\) is FPT parameterized by \(k\). The case of a fixed \(\ell \geq 3\) and the case of jointly parameterizing by \(\ell\) and \(k\) are open for general directed graphs. However, there is a positive answer for directed acyclic graphs.

**Theorem 3.36 ([141]).** Directed Edge Multiway Cut on directed acyclic graphs is FPT parameterized by \(\ell\) and \(k\).

There is a variant of directed multicut where the pushing argument does work. Skew Edge Multicut has the same input as Directed Edge Multicut, but now the task is to find a set \(S\) of at most \(k\) edges such that \(G \setminus S\) has no directed \(s_i \rightarrow t_j\) path for any \(i \geq j\). While this problem is somewhat unnatural, it will be an important ingredient in the algorithm for Directed Feedback Vertex Set in Section 3.6.

**Lemma 3.37 (Pushing Lemma for Skew Edge Multicut).** Let \((G, ((s_1, t_1), \ldots, (s_\ell, t_\ell)), k)\) be an instance of Skew Edge Multicut. If the instance has a solution \(S\), then it has a solution \(S'\) with \(|S'| \leq |S|\) that contains an important \((s_\ell, \{t_1, \ldots, t_\ell\})\)-cut.
Proof. Let \( T = \{ t_1, \ldots, t_k \} \) and let \( R \) be the set of vertices reachable from \( s_\ell \) in \( G \setminus S \). As \( S \) is a solution, \( R \) is disjoint from \( T \) and hence \( \Delta^+(R) \) is an \((s_\ell, T)\)-cut. If \( S_R = \Delta^+(R) \) is an important \((s_\ell, T)\)-cut, then we are done: \( S \) contains every edge of \( \Delta^+(R) \). Otherwise, Property 3.32 implies that there is an important \((X, Y)\)-cut \( S' = \Delta^+(R') \) such that \( R \subseteq R' \) and \(|S'| \leq |S_R|\). We claim that \( S^* = (S \setminus S_R) \cup S' \), which has size at most \(|S|\) is also a solution, proving the statement of the lemma.

The reason why replacing \( S_R \) with the important \((s_\ell, T)\)-cut \( S' \) does not break the solution is because every path that we need to cut in \text{Skew Edge Multicut} ends in \( T = \{ t_1, \ldots, t_k \} \).

Clearly, there is no path between \( s_\ell \) and any terminal in \( T \) in \( G \setminus S^* \), as \( S' \subseteq S^* \) is an \((s_\ell, T)\)-cut. Suppose therefore that there is an \( s_i \to t_j \) path \( P \) for some \( i \geq j \) in \( G \setminus S^* \). If \( P \) goes through a vertex of \( R \subseteq R' \), then it goes through at least one edge of the \((s_\ell, T)\)-cut \( S' = \Delta^+(R') \), which is a subset of \( S^* \), a contradiction. Therefore, we may assume that \( P \) is disjoint from \( R \). Then \( P \) does not go through any edge of \( S_R = \Delta(R) \) and therefore the fact that it is disjoint from \( S^* \) implies that it is disjoint from \( S \), contradicting the assumption that \( S \) is a solution. \( \square \)

Equipped with Lemma 3.34 and Lemma 3.37, a branching algorithm very similar to the proof of Theorem 3.16 shows the fixed-parameter tractability of \text{Skew Edge Multicut}.

**Theorem 3.38.** \text{Skew Edge Multicut} can be solved in time \( O(4^k \cdot k \cdot (|V(G)| + |E(G)|)) \).

### 3.6 Directed Feedback Vertex Set

For undirected graphs, the \text{Feedback Vertex Set} problem asks for a set of at most \( k \) vertices whose deletion makes the graph acyclic, that is, a forest. **References to Feedback Vertex Set in other chapters.** The edge version of \text{Feedback Vertex Set} is polynomial-time solvable.

For directed graphs, both the edge and the vertex versions are interesting. A **feedback vertex set** of a directed graph \( G \) is set \( S \subseteq V(G) \) of vertices such that \( G \setminus S \) has no directed cycle, while a **feedback arc set**\(^2\) is set \( S \subseteq E(G) \) of edges such that \( G \setminus S \) has no directed cycle. Given a graph \( G \) and an integer \( k \), \text{Directed Feedback Vertex Set} asks for a feedback vertex set of size at

\(^2\) Some authors prefer to use the term “arc” for the edges of directed graphs. While we are using the term “directed edge” in this book, the term “arc” is very commonly used in the context of the \text{Directed Feedback Arc Set} problem, hence we also use it in this section.
most $k$. The Directed Feedback Arc Set is defined analogously. There is a simple reduction from the vertex version to the edge version (Exercise 3.12 asks for a reduction in the reverse direction).

**Proposition 3.39.** Directed Feedback Vertex Set can be reduced to Directed Feedback Arc Set in linear time without increasing the parameter.

**Proof.** Let $(G, k)$ be an instance of Directed Feedback Vertex Set. We construct a graph $G'$, where two vertices $v_{in}$ and $v_{out}$ correspond to each vertex $v \in V(G)$. For every edge $(a, b) \in E(G)$, we introduce the edge $(a_{out}, b_{in})$ into $G'$. Additionally, for every $v \in V(G)$, we introduce the edge $(v_{in}, v_{out})$ into $G'$. Observe that there is a one-to-one correspondence between the directed cycles of length exactly $t$ in $G$ and the directed cycles of length exactly $2t$ in $G'$.

We claim that $G$ has a feedback vertex set $S$ of size at most $k$ if and only if $G'$ has a feedback arc set $S'$ of size at most $k$. For the forward direction, given a feedback vertex set $S \subseteq V(G)$ of $G$, let $S'$ contain the corresponding edges of $G'$, that is, $S' = \{(v_{in}, v_{out}) \mid v \in S\}$. If there is a directed cycle $C'$ in $G' \setminus S'$, then the corresponding cycle $C$ in $G$ contains at least one vertex $v \in S$. But then the corresponding edge $(v_{in}, v_{out})$ of $C'$ is in $S'$, a contradiction.

For the reverse direction, let $S' \subseteq E(G)$ be a feedback arc set of $G'$. We may assume that every edge of $S'$ is of the form $(v_{in}, v_{out})$: if $(a_{out}, b_{in}) \in S'$, then we may replace it with $(b_{in}, a_{out})$, as every directed cycle going through the former edge goes through the latter edge as well. Let $S = \{v \mid (v_{in}, v_{out}) \in S'\}$. Now if $G \setminus S$ has a directed cycle $C$, then the corresponding directed cycle $C'$ of $G$ goes through an edge $(v_{in}, v_{out}) \in S'$, implying that vertex $v$ of $C$ is in $S$, a contradiction. \qed

In the rest of the section, we show the fixed-parameter tractability of Directed Feedback Arc Set using the technique of iterative compression.

**References to other examples of ic.**

We apply a trick that is often useful when solving edge-deletion problems using iterative compression: the initial solution that we need to compress is a vertex set, not an edge set.

That is, the input of the Directed Feedback Arc Set Compression problem is a directed graph $G$, an integer $k$, and a feedback vertex set $W$ of $G$, the task is to find a feedback arc set $S$ of size at most $k$. We show, by a reduction to Skew Edge Multicut, that this problem is FPT parameterized by $|W|$ and $k$.

**Lemma 3.40.** Directed Feedback Arc Set Compression can be solved in time $O(|W|! \cdot 4^k \cdot |V(G)| \cdot k(|V(G)| + |E(G)|))$. 

3.6 Directed Feedback Vertex Set

Fig. 3.9: Solving Directed Feedback Arc Set Compression in the proof of Lemma 3.40. Every directed cycle in $G'$ has an $s_i \rightarrow t_j$ subpath for some $i \geq j$.

**Proof.** The task is to find a set $S$ of edges where $G \setminus S$ has no directed cycle, or equivalently, has a topological ordering. Every topological ordering induces an ordering on the set $W$ of vertices. Our algorithm starts by guessing an ordering $W = \{w_1, \ldots, w_{|W|}\}$ and the rest of the algorithm works under the assumption that there is a solution compatible with this ordering. There are $|W|!$ possibilities for the ordering of $W$, thus the running time of the rest of the algorithm has to be multiplied by this factor.

Every directed cycle contains a $w_i \rightarrow w_j$ walk for some $i \geq j$: the cycle contains at least one vertex of $W$ and has to go “backwards” at some point. Therefore, cutting all walk paths (which is a problem similar to Skew Edge Multicut) gives a feedback arc set.

We build a graph $G'$ the following way. We replace every vertex $w_i$ with two vertices $s_i, t_i$, and edge $(t_i, s_i)$ between them. We define $E_W = \{(t_i, s_i) \mid 1 \leq i \leq |W|\}$. Then we replace every edge $(a, w_i)$ with $(a, t_i)$, every edge $(w_i, a)$ with $(s_i, a)$, and every edge $(w_i, w_j)$ with $(s_i, t_j)$ (see Figure 3.9) Let us define the Skew Edge Multicut instance $(G', (s_1, t_1), \ldots, (s_{|W|}, t_{|W|}), k)$. The following two claims establish a connection between the solutions of this instance and our problem.

**Claim 3.41.** If there is a set $S \subseteq E(G)$ of size at most $k$ such that $G \setminus S$ has a topological ordering inducing the order $w_1, \ldots, w_{|W|}$ on $W$, then the Skew Edge Multicut instance has a solution.

**Proof.** For every edge $e \in S$, there is a corresponding edge $e'$ of $G'$; let $S'$ be the set corresponding to $S$. Suppose that $G' \setminus S'$ has a directed $s_i \rightarrow t_j$
path $P$ for some $i \geq j$. Path $P$ may go through several edges $(t_i, s_i)$, but it has an $s_i \to t_{i'}$ subpath $P'$ such that $i' \geq j'$ and $P'$ has no internal vertex of the form $t_i, s_i$. If $i' = j'$, then the edges of $G$ corresponding to $P'$ give directed cycle $C$ in $G$ disjoint from $S$, a contradiction. If $i' > j'$, then there is a directed $w_{i'} \to w_{j'}$ path $P$ of $G$ corresponding to $P'$ that is disjoint from $S$, contradicting the assumption that $w_{i'}$ is later than $w_{j'}$ in the topological ordering of $G \setminus S$. \hfill \qed

**Claim 3.42.** Given a solution $S'$ of the Skew Edge Multicut instance, one can find a feedback arc set $S$ of size at most $k$ for $G$.

**Proof.** Suppose that the Skew Edge Multicut instance has a solution $S'$ of size at most $k$. For every edge of $E(G') \setminus E_W$, there is a corresponding edge of $G$; let $S$ be the set of at most $k$ edges corresponding to $S' \setminus E_W$. Suppose that $G \setminus S$ has a directed cycle $C$. This cycle $C$ has to go through at least one vertex of $W$. If $C$ goes through exactly one vertex $w_i$, then the edges of $G'$ corresponding to $C$ give a directed $t_i \to s_i$ path disjoint from $S'$, a contradiction (note that this path is certainly disjoint from $E_W$). If $C$ goes through more than one $w_i$, then $C$ has a directed $w_i \to w_j$ path for some $i > j$ such that the internal vertices of $P$ are not in $W$. Then the edges of $G'$ corresponding to $P$ give a directed $s_i \to t_j$ path in $G'$ disjoint from $S'$ (and from $E_W$), again a contradiction. \hfill \qed

Suppose that there is a feedback arc set $S \subseteq E(G)$ of size at most $k$. A topological ordering of $G \setminus S$ induces an ordering on $W$ and, as the algorithm tries every ordering of $W$, eventually this ordering is reached. At this point, Claim 3.41 implies that the constructed Skew Edge Multicut instance has a solution $S'$ and we can find such a solution using the algorithm of Theorem 3.38. Then Claim 3.42 allows us to find a solution of Directed Feedback Arc Set Compression. The algorithm tries $|W|!$ orderings of $W$ and uses the $4^k \cdot k \cdot (|V(G)| + |E(G)|)$ time algorithm of Theorem 3.38, every other step can be done in linear time. \hfill \Box

Finally, let us see how iterative compression can be used to solve Directed Feedback Arc Set using Directed Feedback Arc Set Compression.

**Theorem 3.43.** There is a $O(4^k \cdot (k + 1)! \cdot k \cdot |V(G)|(|V(G)| + |E(G)|)$ time algorithm for Directed Feedback Arc Set.

**Proof.** Let $v_1, \ldots, v_n$ be an arbitrary ordering of the vertices of $G$ and let $G_i = G[\{v_1, \ldots, v_i\}]$. For $i = 1, 2, \ldots, n$, we compute a feedback arc set $S_i$ of size at most $k$ for $S_i$ for $G_i$. For $i = 1$, $S_1 = \emptyset$ is a trivial solution. Suppose now that we have already computed $S_i$; then $S_{i+1}$ can be computed as follows. Let $W_i \subseteq V(G_i)$ be the head of each edge in $S_i$. Clearly, $|W_i| \leq k$ and $G \setminus W_i$ is acyclic (since $G \setminus S_i$ is acyclic). Moreover, $W_i \cup \{v_{i+1}\}$ has size at most $k + 1$ and $G_{i+1} \setminus (W_i \cup \{v_{i+1}\}) = G_i \setminus W_i$, hence it is acyclic.
Therefore, we may invoke the algorithm of Lemma 3.40 to find a feedback arc set \( S_{i+1} \) of size at most \( k \) for \( G_{i+1} \). If the algorithm answers that there is no feedback arc set of size at most \( k \) for \( G_{i+1} \), then there is no such feedback arc set for the supergraph \( G \) of \( G_{i+1} \) and hence we can return a negative answer for Directed Feedback Arc Set on \( (G, k) \). Otherwise, if the algorithm returns a feedback arc set \( S_{i+1} \) for \( G_{i+1} \), then we can continue the iteration with \( i + 1 \).

As \( |W_i| \leq k + 1 \) in each call of the Directed Feedback Arc Set Compression algorithm of Lemma 3.40, its running time is \( \mathcal{O}((k+1)! \cdot 4^k \cdot |V(G)| \cdot k(|V(G)| + |E(G)|)) \). Taking into account that we call this algorithm at most \( |V(G)| \) times, the claimed running time follows.

Using the simple reduction from the vertex version to the edge version (Proposition 3.39), we obtain the fixed-parameter tractability of Directed Feedback Vertex Set.

**Corollary 3.44.** There is a \( \mathcal{O}((k+1)! \cdot 4^k \cdot k|V(G)|(|V(G)| + |E(G)|)) \) time algorithm for Directed Feedback Vertex Set.

### 3.7 Vertex-deletion problems

All the problems treated so far in the chapter were defined in terms of removing edges of a graph (with the exception of Directed Feedback Vertex Set, which was quickly reduced to Directed Feedback Arc Set). In this section, we introduce the definition needed for vertex-removal problems and state (without proofs) the analogs of the edge-results we have seen in the previous sections.

A technical issue specific to vertex-removal problems is that we have to define whether a vertex set separating \( X \) and \( Y \) is allowed to contain a vertex of \( X \setminus Y \). If so, then \( X \) and \( Y \) need not be disjoint (but then the separator has to contain \( X \cap Y \)). In a similar way, we have to define whether, say, Vertex Multiway Cut allows the deletion of the terminals. Note that in an edge-deletion problem we can typically make an edge “undeletable” by replacing it with \( k + 1 \) parallel edges (as we did in the proof of Lemma 3.40). Similarly, one may try to make a vertex undeletable by replacing it with a clique of size \( k + 1 \) (with the same neighborhood), but this can be notationally inconvenient, especially when terminals are present in the graph. Therefore, we state the definitions and the results in a way that we assume that the (directed or undirected) graph \( G \) is equipped with a set \( V^\sim(G) \subseteq V(G) \) of undeletable vertices. Given two (not necessarily disjoint) sets \( X, Y \subseteq V(G) \) of vertices, an \( (X, Y) \)-separator is a set \( S \subseteq V(G) \setminus V^\sim(G) \) of vertices such that \( G \setminus S \) has no \( (X \setminus S) - (Y \setminus S) \) path. We define minimal and minimum \( (X, Y) \)-separators the usual way. We can characterize minimal \( (X, Y) \)-separators as the neighborhood a set of vertices.
Proposition 3.45. If $S$ is a minimal $(X,Y)$-separator, then $S = N(R)$ (or $S = N^+(R)$ in directed graphs), where $R$ is the set of vertices reachable from $X \setminus S$ in $G \setminus S$.

The results of Section 3.1 can be adapted to $(X,Y)$-separators, we omit the details. Important $(X,Y)$-separators are defined analogously to important $(X,Y)$-cuts.

Definition 3.46. Let $G$ be a directed or undirected graph and let $X, Y \subseteq V(G)$ be two sets of vertices. Let $S \subseteq V(G) \setminus V^\infty(G)$ be an $(X,Y)$-separator and let $R$ be the set of vertices reachable from $X \setminus S$ in $G \setminus S$. We say that $S$ is an important $(X,Y)$-separator if it is inclusionwise minimal and there is no $(X,Y)$-separator $S' \subseteq V(G) \setminus V^\infty(G)$ with $|S'| \leq |S|$ such that $R \subseteq R'$, where $R$ is the set of vertices reachable from $X \setminus S$ in $G \setminus S$.

The analog of Proposition 3.47 holds for $(X,Y)$-separators.

Proposition 3.47. Let $G$ be a directed or undirected graph and $X, Y \subseteq V(G)$ two sets of vertices. Let $S \subseteq V(G) \setminus V^\infty(G)$ be an $(X,Y)$-separator and let $R$ be the set of vertices reachable from $X \setminus S$ in $G \setminus S$. Then there is an important $(X,Y)$-separator $S' = N(R)$ (or $S' = N^+(R)$ if $G$ is directed) such that $|S'| \leq |S|$ and $R \subseteq R'$.

The algorithm for enumerating important $(X,Y)$-cuts and the combinatorial bounds on their number can be adapted to important $(X,Y)$-separators.

Theorem 3.48. Let $X, Y \subseteq V(G)$ be two sets of vertices in a (directed or undirected) graph $G$, let $k \geq 0$ be an integer, and let $S_k$ be the set of all $(X,Y)$-important cuts of size at most $k$. Then $|S_k| \leq 4^k$ and $S_k$ can be constructed in time $|S_k| \cdot k \cdot (|V(G)| + |E(G)|)$.

Lemma 3.49. Let $G$ be a (directed or undirected) graph and let $X, Y \subseteq V(G)$ be two sets of vertices. If $S$ is the set of all important $(X,Y)$-cuts, then $\sum_{S \in S} 4^{-|S|} \leq 1$.

Let $G$ be a graph and $T \subseteq V(G)$ be a set of terminals. A vertex multiway cut is a set $S \subseteq V(G) \setminus V^\infty(G)$ of vertices such that every component of $G \setminus S$ contains at most one vertex of $T \setminus S$. Given an undirected graph $G$, an integer $k$, a set $T \subseteq V(G)$ of terminals, the VERTEX MULTIWAY CUT problem asks for vertex multiway cut of size at most $k$. The analog of the Pushing Lemma for EDGE MULTIWAY CUT (Lemma 3.14) can be adapted to the vertex-deletion case.

Lemma 3.50 (Pushing Lemma for VERTEX MULTIWAY CUT). Let $t \in T$ be a terminal that is not separated from $T \setminus t$ in an undirected graph $G$. If $G$ has a vertex multiway cut $S \subseteq V(G) \setminus V^\infty(G)$, then it also has a vertex multiway cut $S^* \subseteq V(G) \setminus V^\infty(G)$ with $|S^*| \leq |S|$ such that $S^*$ contains an important $(t, T \setminus t)$-separator.
Theorem 3.48, Lemma 3.49, and Lemma 3.50 allow us to solve \textsc{Vertex Multiway Cut} using a branching strategy identical to the one used in Theorem 3.16.

**Theorem 3.51.** \textsc{Vertex Multiway Cut} can be solved in time $O(4^k \cdot k \cdot (|V(G)| + |E(G)|))$.

The result of Theorem 3.18 was actually proved in the literature for the (more general) vertex version, hence \textsc{Vertex Multicut} is also fixed-parameter tractable parameterized by $k$.

**Theorem 3.52** ([29, 165]). \textsc{Vertex Multicut} can be solved in time $2^{O(k^3)} \cdot n^O(1)$.

**Exercises**

3.1. Given an undirected graph $G$ and $S \subseteq V(G)$, let $i_G(S)$ be number of edges induced by $S$ (that is, $i_G(S) = |E(G[S])|$). Is $i_G$ submodular?

3.2. Prove that a set function $f : 2^{V(G)} \rightarrow \mathbb{R}$ is submodular if and only if

$$f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B) \quad (3.3)$$

holds for every $A \subseteq B \subseteq V(G)$ and $v \in V(G)$. Informally, inequality (3.3) says that the marginal value of $v$ with respect to the superset $B$ (that is, the increase of value if we extend $B$ with $v$) cannot be larger than with respect to a subset $A$.

3.3. Give a polynomial-time algorithm for finding the sets $R_{\min}$ and $R_{\max}$ defined in Theorem 3.4 using only the facts that such sets exists and that a minimum $(X, Y)$-cut can be found in polynomial time.

3.4. Let $G$ be an undirected graph and let $A, B \subseteq V(G)$ be two disjoint sets of vertices. Let $\Delta(R^{AB}_{\min})$ and $\Delta(R^{BA}_{\max})$ be the minimum $(A, B)$-cuts defined by Theorem 3.4, and let and let $\Delta(R^{BA}_{\min})$ and $\Delta(R^{AB}_{\max})$ be the minimum $(B, A)$-cuts defined by Theorem 3.4 (reversing the role of $A$ and $B$). Show that $R^{AB}_{\min} = V(G) \setminus R^{BA}_{\max}$ and $R^{BA}_{\min} = V(G) \setminus R^{AB}_{\max}$.

3.5. Prove Proposition 3.9.

3.6. Give an example where the algorithm of Theorem 3.11 finds an $(X, Y)$-cut that is not an important $(X, Y)$-cut (and hence throws it away in the filtering phase). Can such a cut come from the first recursive branch of the algorithm? Can it come from the second?

3.7. Let $G$ be an undirected graph, let $X, Y \subseteq V(G)$ be two disjoint sets of vertices, and let $\lambda$ be the minimum $(X, Y)$-cut size. Let $S$ be the set of all important $(X, Y)$-cuts. Prove that $\sum_{S \in S} 2^{-|S|} \leq 2^{-\lambda}$ holds.

3.8. Show that \textsc{Edge Multiway Cut} is polynomial-time solvable on trees.

3.9. Show that \textsc{Edge Multicut} is NP-hard on trees.

3.10. Give a $2^k \cdot n^{O(1)}$ time algorithm for \textsc{Edge Multicut} on trees.
3.11. Reduce **Directed Edge Multicut** with $\ell = 2$ to **Directed Edge Multiway Cut** with $|T| = 2$.

3.12. Reduce **Directed Feedback Arc Set** to **Directed Feedback Vertex Set** in polynomial time.

3.13. Show that **Vertex Multicut** is polynomial-time solvable on trees.

**Hints**

3.1 The function $i_G$ is not supermodular, but supermodular: the left-hand side of Inequality (3.1) is always at most the right-hand side.

3.4 If $\Delta(R)$ is a minimum $(A, B)$-cut, then $\Delta(R) = \Delta(V(G) \setminus R)$ and it is also a minimum $(B, A)$-cut. The equalities then follow simply from the fact that minimizing $R$ is the same as maximizing $V(G) \setminus R$.

3.2 To see that (3.3) is a necessary condition for $f$ being submodular, observe that it is the rearrangement of (3.1) applied for $A \cup \{v\}$ and $B$. To see that it is a necessary condition, let us build $X \cup Y$ from $X \cap Y$ by first adding the elements of $X \setminus Y$ one by one and then adding the elements of $Y \setminus X$ one by one. Let us compare the total marginal increase of adding these elements to the total marginal increase of building $X$ from $X \cap Y$, plus the total marginal increase of building $Y$ from $X \cap Y$. The submodularity follows by observing that the marginal increase of adding elements of $Y \setminus X$ cannot be larger if $X \setminus Y$ is already present in the set.

3.6 It is possible that the first branch gives an important $(X, Y)$-cut $S$ in $G \setminus xy$ such that $S \cup \{xy\}$ is not an important $(X, Y)$-cut in $G$. For example, let $X = \{x\}$, $Y = \{y\}$, a suppose that graph $G$ has the edges $xa$, $xb$, $ab$, $ay$, $by$, $cy$. Then $R_{\text{max}} = \{x\}$. In graph $G \setminus xb$, the set $\{ay, by\}$ is an important $(X, Y)$-cut. However, $\Delta_G(\{x, a\}) = \{ay, by, xb\}$ is not an important $(X, Y)$-cut in $G$, as $\Delta_G(\{x, a, b, c\}) = \{ay, by, cy\}$ has the same size.

One can show that every $(X,Y)$-cut returned by the second branch is an important $(X,Y)$-cut in $G$.

3.7 We need to show that in each of the two branches of the algorithm in Theorem 3.11, the total contributions of the enumerated separators to the sum is at most $2^{-\lambda}/2$. In the first branch, this is true because we augment each cut with a new edge. In the second branch, this is true because $\lambda$ strictly increases.

3.8 Solution 1: Dynamic programming. Assume that the tree is rooted and let $T_v$ be the subtree rooted at $v$. Let $A_v$ be the minimum cost of separating the terminals in $T_v$, and let $B_v$ the minimum cost with the additional constraint that $v$ is separated from every terminal in $T_v$. Give a recurrence relation for computing $A_v$ and $B_v$ if these values are known for every child of $v$.

Solution 2: We may assume that every leaf of the tree contains a terminal, otherwise the leaf can be removed without changing the problem. Select any leaf and argue that there is a solution containing the edge incident to this leaf.

3.9 Reduction from **Vertex Cover**: the graph is a star, edges play the role of vertices, and the terminal pairs play the role of edges.

3.10 Assume that the tree is rooted and select a pair $(s_i, t_i)$ such that the topmost vertex $v$ of the unique $s_i - t_i$ path $P$ has maximum distance from the root. Argue that there is a solution containing an edge of $P$ incident to $v$. 
3.11 Introduce two terminals $v_1, v_2$, and add the edges $(v_1, s_1), (t_1, v_2), (v_2, s_2), (t_2, v_1)$.

3.12 Subdivide each edge and replace each original vertex with an independent set of size $k + 1$.

3.13 Assume that the tree is rooted and select a pair $(s_i, t_i)$ such that the topmost vertex $v$ of the unique $s_i - t_i$ path $P$ has maximum distance from the root. Argue that there is a solution containing vertex $v$.

Bibliographic notes

There is a large literature on polynomial-time algorithms for minimum cut problems, maximum flow problems, and their relation to submodularity. The monumental work of Schrijver [187] and the recent monograph of Frank [107] give detailed overview of the subject.

The Ford-Fulkerson algorithm for finding maximum flows (which is the only maximum flow algorithm we needed in this chapter) was published in 1956 [105]. For other, more efficient, algorithms for finding maximum flows, see any standard algorithm textbooks such as Cormen, Leiserson, Rivest, Stein [50].

Marx [156] defined the notion of important separators and proved the fixed-parameter tractability of VERTEX MULTIWAY CUT using the branching approach of Theorem 3.16. The $4^k$ bound and the proof of Theorem 3.11 is implicit in the work of Chen, Liu, and Lu [43]. Using different approaches, one can obtain better than $4^k \cdot n^{O(1)}$ time algorithms: Xiao [197] gave a $2^k \cdot n^{O(1)}$ time algorithm for EDGE MULTIWAY CUT, Cao, Chen, and Fan gave a $1.84^k \cdot n^{O(1)}$ time algorithm for EDGE MULTIWAY CUT, and Cygan, Pilipczuk, Pilipczuk, and Wojtaszczyk [66] gave a $2^k \cdot n^{O(1)}$ time algorithm for VERTEX MULTIWAY CUT.

The NP-hardness of EDGE MULTICUT on trees was observed in by Garg, Vazirani, and Yannakakis [109]. The $2^k \cdot n^{O(1)}$ algorithm of Exercise 3.10 is by Guo and Niedermeier [113]. The polynomial-time algorithm for solving VERTEX MULTICUT on trees (Exercise 3.13) was described first by Clinescu, Fernandes, and Reed [34].

The NP-hardness of EDGE MULTIWAY CUT with $|T| = 3$ was proved by Dalhaus et al. [67]. Interestingly, for a fixed number $\ell$ of terminals, they also show that the problem can be solved in time $n^{O(\ell)}$ time on planar graphs. The running time for planar graphs was improved to $2^{O(\ell)} \cdot n^{O(\sqrt{\ell})}$ by Klein and Marx [136].

The fixed-parameter tractability of EDGE MULTICUT and VERTEX MULTICUT parameterized by the solution size $k$ was proved independently by Bousquet [29] and Marx and Razgon [165]. The paper of Marx and Razgon introduced the random sampling of important separators technique. This technique was used then by Lokshtanov and Marx [149] to prove the fixed-parameter tractability of $(p,q)$-CLUSTER parameterized by $q$. Lokshtanov and Marx proved the problem is FPT also with parameter $p$, and studied other variants of the problem, such as when the requirement $|C| \leq p$ is replaced by requiring that the graph induced by $C$ has at most $p$ nonedges.

The random sampling of important separators technique was used by Lokshtanov and Ramanujan [152] to solve PARITY MULTIWAY CUT and by Chitnis, Egri, and Marx [48] to solve a certain list homomorphism problem with vertex removals. The technique was generalized to directed graphs by Chitnis, Hajiaghayi, and Marx [46] to show the fixed-parameter tractability of DIRECTED EDGE MULTIWAY CUT and DIRECTED VERTEX MULTIWAY CUT parameterized by the size of the solution. The directed version was used by Kratsch, Pilipczuk, Pilipczuk, and Wahlström [141] to prove that EDGE MULTICUT is FPT parameterized by $p$ and $\ell$ on directed acyclic graphs and by Chitnis, Cygan, Hajiaghayi, and Marx [47] to prove that DIRECTED SUBSET FEEDBACK VERTEX SET is FPT.
The fixed-parameter tractability of Skew Edge Multicut, Directed Feedback Vertex Set, and Directed Feedback Arc Set were shown by Chen et al. [44].