T. Kloks

Advanced Graph Algorithms

February 3, 2012

Contents

1 Exponential Algorithms

Let's face it: lots of interesting problems on graphs are NP-complete. In this chapter we have a look at exponential algorithms.

Let's look at an example. Suppose we want to solve the maximum independent set problem on some graph $G = (V, E)$. An independent set is a subset $M \subseteq V$ of vertices such that no two vertices in M are adjacent. The maximum independent set problem asks for an independent set M in G such that $|M|$ is maximal. An answer to the problem is called a 'maximum' independent set. We denote the cardinality of a maximum independent set in G with $\alpha(G)$.

An easy way to solve the problem is as follows. First, make a list of all subsets of vertices. Next, check which subsets are independent sets. Then count the number of vertices in each independent set and take the largest one.

What is the time-complexity of this algorithm? Let n be the number of vertices in the graph G. Obviously, there are 2^n subsets of vertices. Let M be a subset. We need to check if M is an independent set. We assume that the graph G is represented by an adjacency matrix A. Then we can check if two vertices are adjacent in constant time. Since M has $O(n^2)$ pairs we can check if M is an independent set in $O(n^2)$ time. Thus our algorithm runs in $O(n^2 \cdot 2^n)$ time. In the next section we show that we can do better.

When we are dealing with exponential algorithm we don't care so much about the polynomial factors in the time-bound. The O^*-notation neglects the polynomial factors. So instead of $O(n^2 \cdot 2^n)$ we write $O^*(2^n)$.

The O^*-notation is pretty useful. For example, suppose our graph G is represented by adjacency lists. That is, for each vertex x in G there is a linked list $L(x)$ of its neighbors. Now, to check if two vertices x and y are adjacent we need to check if y appears in the list $L(x)$ or not. In the worst case $L(x)$ contains $n-1$ vertices, so checking if y is in this list takes more than constant time. (Of course, we can construct the adjacency matrix A in $O(n^2)$ time and then proceed as before.)

If we use the O^*-notation then we don't need to worry about such details. Let $p(n)$ be some polynomial, for example $p(n) = 10 \cdot n^5$. Suppose we can check if any subset M is an independent set in at most $p(n)$ time. Then the algorithm described above runs in $O(p(n) \cdot 2^n) = O^*(2^n)$ time.

1.1 Independent set

An independent set M in a graph $G = (V, E)$ is *maximal* if any vertex in $V \setminus M$ has at least one neighbor in M. Moon and Moser[1] have shown that any graph with n vertices has at most $3^{n/3}$ maximal independent sets. Notice that a graph which is the union of $\frac{n}{3}$ triangles achieves this bound.

There are algorithms that list all the maximal independent sets with *polynomial delay*. That means that there exists some polynomial $p(n)$ such that the algorithm spends at most $p(n)$ time before it generates the next (or the first) independent set. For example, the algorithm of Tsukiyama, *et al.*, takes $O(nm)$ time per maximal independent set, where n and m are the number of vertices and edges in the graph.[2] These two results yield the following theorem.

Theorem 1.1. *There exists an* $O^*(1.4422^n)$ *algorithm that solves the maximum independent set problem on a graph* G*, where* n *is the number of vertices in* G.

Proof. We use an algorithm which lists all maximal independent sets in G in $O(p(n) \cdot 3^{n/3})$ time for some polynomial $p(n)$. Notice that

$$O(p(n) \cdot 3^{n/3}) = O^*(3^{n/3}) = O^*(1.4422^n).$$

This proves the theorem. □

Of course, this algorithm is much better than the $O^*(2^n)$ algorithm that we started with. In the rest of this section we show that we can still do a little bit better.

[1] J. W. Moon and L. Moser, On cliques in graphs, *Israel Journal of Mathematics* **3** (1965), pp. 23–28.

[2] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM Journal on Computing* **6** (1977), pp. 505–517.

Let x be a vertex of G. As usual we use the notation $N(x)$ for the set of neighbors of a vertex x. The degree of x is $|N(x)|$. We also use the notation $N[x]$ for the *closed* neighborhood of x, which is

$$N[x] = N(x) \cup \{x\}.$$

Let x be a vertex in G. There are two types of independent sets, namely those that contain x and those that do not contain x. Consider a maximum independent set M. The next two lemmas show how to reduce the graph in each of the two cases.

Lemma 1.2. *Let M be a maximum independent set in G and let $x \notin M$. Then M is a maximum independent set in $G - x$.*

Proof. The graph $G - x$ is the subgraph of G induced by $V \setminus \{x\}$. Let M be a maximum independent set in G and let $x \notin M$. Then M is an independent set in $G - x$.

Of course, any independent set in $G - x$ is also an independent set in G. Thus $G - x$ cannot have an independent set M' with $|M'| > |M|$ since this contradicts the assumption that M is a maximum independent set in G.
This proves the lemma. □

Lemma 1.3. *Let M be a maximum independent set in G and let $x \in M$. Then $M \setminus \{x\}$ is a maximum independent set in $G - N[x]$.*

Proof. The graph $H = G - N[x]$ is the subgraph of G induced by $V \setminus N[x]$.

Let M be a maximum independent set in G and assume that $x \in M$. Notice that $M \setminus \{x\}$ is an independent set in $G - N[x]$.

Suppose that H has an independent set M' which is larger than $M \setminus \{x\}$. Since M' is an independent set in $G - N[x]$, M' contains no neighbors of x. Thus $M' \cup \{x\}$ is an independent set in G which is larger than M and this contradicts the assumption.
This proves the lemma. □

In other words, Lemmas 1.2 and 1.3 show that, for any vertex x, a maximum independent set M can be derived from a maximimum independent set in $G - x$ or from a maximum independent set in $G - N[x]$.

The algorithm builds a rooted binary tree T as follows. The root of T corresponds with the graph G. If the graph has only one vertex, then T consists of the root only. Otherwise, choose a vertex x in G. The root has two children.

The left child is the root of a binary tree which corresponds with the graph $G - x$. The right child is the root of a binary tree which corresponds with the graph $G - N[x]$.

Our algorithm computes a maximum independent set in G as follows. If G has only one vertex then $\alpha(G) = 1$. Otherwise, the algorithm recursively computes the maximum independent set in the left subtree and in the right subtree. By the two lemmas above,

$$\alpha(G) = \max \{ \, \alpha(G - x), \; 1 + \alpha(G - N[x]) \, \}. \tag{1.1}$$

We need to make a remark here. If x is the only vertex in G then $G - x$ is not a graph since, by definition, a graph has at least one vertex. (The 'empty graph' is the graph without any edge.) In Formula (1.1), if $V = \{x\}$ then we define $\alpha(G - x) = 0$. A similar situation occurs when x is adjacent to all other vertices, *i.e.*, when $N[x] = V$. In that case we define $\alpha(G - N[x]) = 0$.

For example, assume that the graph G is a clique. A clique is a subset C of vertices such that every pair of vertices in C is adjacent. In other words, a clique in the graph is an independent set in the complement $\bar{G}$ of the graph G and *vice versa*.

Since G is a clique, any maximal independent set in G has only one vertex. Thus G has exactly n maximal independent sets. In this case, every pair of vertices in G is adjacent, thus $N[x] = V$ for any vertex x. If x is the only vertex in G then $\alpha(G - x) = 0$ and otherwise $\alpha(G - x) = 1$.

In order to obtain a good timebound we like to reduce the graph in at least one of the two branches as much as possible. One branch only removes the vertex x and this reduces the graph by one vertex. The other branch removes $|N[x]|$ vertices from the graph. To make this graph as small as possible we choose the vertex x such that it has the largest degree.

Lemma 1.4. *Let* G *be a graph and assume that every vertex of* G *has degree at most 2. Then a maximum independent set in* G *can be computed in linear time.*

Proof. Since every vertex has degree at most 2, the graph is the union of a collection of paths and cycles.

To compute the maximum independent set of G we can compute the maximum independent set in each of the (connected) components of G and add them up. We leave it as an exercise to check the following claims, for example by using (1.1).

The *length* of a path or cycle is the number of edges in the path or the cycle. An *isolated vertex* in G is a vertex without neighbors.

(1) If C is a cycle of length $2k$ then $\alpha(C) = k$.
(2) If C is a cycle of length $2k + 1$ then $\alpha(C) = k$.
(3) If P is a path of length $2\ell \geqslant 0$ then $\alpha(P) = \ell + 1$.
(4) If P is a path of length $2\ell + 1 > 0$ then $\alpha(P) = \ell + 1$.

Thus, in order to compute $\alpha(G)$ it suffices to compute the lengths of the components of G, and this can be done in linear time. □

We now change the algorithm a little bit, as follows. As long as there exists a vertex x in the graph with degree at least three, then the algorithm chooses such a vertex to build the tree T. When, at some point, a reduced graph H has no more vertices of degree more than two then the algorithm uses the linear-time algorithm described in Lemma 1.4 to compute $\alpha(H)$.

Let $T(n)$ be the worst-case timebound that the algorithm needs to compute $\alpha(G)$ for a graph G with n vertices. Then, by the Formula (1.1) and by Lemma 1.4 we have the following recurrence relation for $T(n)$.

$$T(n) \leqslant T(n-1) + T(n-4) + O(n+m). \tag{1.2}$$

Perhaps you wonder why we write $T(n-4)$. If x is a vertex of degree i then the Formula (1.1) says $T(n-i-1)$ instead of $T(n-4)$. However, notice that $T(n)$ is a non-decreasing function (see Exercise 1.3), and since the degree of x is at least three

$$i \geqslant 3 \quad \Rightarrow \quad T(n-i-1) \leqslant T(n-4).$$

It is an easy exercise to check that $T(n) \leqslant \omega^n \cdot p(n)$, where $p(n)$ is some polynomial and ω is the largest real root of the equation

$$\omega^4 = \omega^3 + 1.$$

Some calculations yield $\omega \approx 1.3803$. This proves the following theorem.

Theorem 1.5. *There exists an* $O^*(1.3803^n)$ *algorithm which solves the maximum independent set problem on a graph* G*, where* n *is the number of vertices in* G.

Remark 1.6. The technique that we used is sometimes called a 'pruned search tree technique.' Pruning means that the search tree is cut short. In our case the leaves of the search tree are graphs in which every vertex has degree at most two. There are many ways to prune the search tree further (see, *e.g.*, Exercise 1.2). It is one of the well-known techniques that are used in the design of exponential algorithms.

Remark 1.7. Very often the timebound for an exponential algorithm follows from a recurrence relation like Formula (1.2). If you want to study exact algorithms it's a good idea to obtain some basic knowledge about solving recurrence relations.

Remark 1.8. The best time-bound for an algorithm that solves the maximum independent set problem seems to be Robson's algorithm.[3] This algorithm uses exponential space. Robson claims that he improved his algorithm such that it runs in $O^*(1.1844^n)$. This algorithm uses a computer-generated case analysis with thousands of different cases. Recently, Fomin and Kratsch developed an algorithm for maximum independent set which runs in $O^*(1.2786^n)$.[4] This algorithm uses only polynomial space.

1.2 Chromatic number

Definition 1.9. *The chromatic number of a graph* G *is the minimal number of colors needed to color the vertices of* G *such that no two adjacent vertices in* G *have the same color.*

Usually, one denotes the chromatic number of a graph G by $\chi(G)$.

The chromatic number problem is one of the most studied problems in graph algorithms. The problem is NP-complete for graphs in general. Even the problem to color a *planar* graph with three colors is NP-complete. (By the 4-color theorem every planar graph can be colored with four colors, but checking if a planar graph can be colored with three colors is NP-complete.) In this section we look at an exact algorithm for the chromatic number problem.

Consider a coloring of a graph $G = (V, E)$ with k colors. Consider a set T of vertices that all have the same color. Then, by definition, T is an independent set. Notice also that we may assume that T is maximal. This can be seen as follows. Suppose T is contained in a maximal independent set T'. Then we can re-color the vertices of $T' \setminus T$ such that all the vertices of T' have the same color. Thus we obtain a coloring of G with k colors and now T' is maximal. This proves that the chromatic number of G is determined by the following formula.

$$\chi(G) = 1 + \min \; \{ \, 1 + \chi(G - T) \mid T \text{ is a maximal independent set in } G \, \}. \quad (1.3)$$

When G is an independent set then $V - T = \varnothing$ and then $G - T$ is not a graph. In that case we define $\chi(G - T) = 0$.

[3] J. M. Robson, Algorithms for maximum independent sets, *Journal of Algorithms* **7** (1986), pp. 425–440.

[4] F. Fomin and D. Kratsch, *Exact exponential algorithms*, Springer, 2010.

Recall the result of Moon and Moser: every graph with n vertices has at most $3^{n/3}$ maximal independent sets and these can be listed in $O(p(n) \cdot 3^{n/3})$ time, where $p(n) = n^c$ is some polynomial (in Tsukiyama's algorithm $c \leqslant 3$).

Our algorithm considers all possible subsets S of vertices in G and it colors $G[S]$. Here, $G[S]$ is the graph induced by S. The subsets are processed in order of increasing cardinality. Thus, when S is considered by the algorithm, all the subsets of S have already been colored. When S has ℓ vertices, then the algorithm lists all the maximal independent sets in $G[S]$ in $O(p(\ell) \cdot 3^{\ell/3})$ time.

By Formula (1.3) the algorithm computes $\chi(G[S])$ in $O(p(\ell) \cdot 3^{\ell/3})$ time, since all the values of

$$G[S] - T = G[S \setminus T]$$

were determined earlier (because $S \setminus T \subset S$), for all maximal independent sets T in $G[S]$.

When G has n vertices, then there are $\binom{n}{\ell}$ subsets S with ℓ vertices. This shows that our algorithm has a running time proportional to

$$\sum_{\ell=0}^{n} \binom{n}{\ell} p(\ell)\, 3^{\ell/3} \leqslant p(n) \sum_{\ell=0}^{n} \binom{n}{\ell} 3^{\ell/3} = p(n)(1 + 3^{1/3})^n.$$

An easy calculation shows that $1 + 3^{1/3} \approx 2.4422$ and this proves the following theorem.

Theorem 1.10. *There exists an* $O^*(2.4422^n)$ *algorithm to compute the chromatic number of a graph* G*, where* n *is the number of vertices in* G.

Remark 1.11. Notice that this algorithm uses exponential space.

Remark 1.12. Recently, some improvements were obtained by Björkland, *et al.*[5] They show that the chromatic number problem can be solved in $O^*(2^n)$.

1.2.1 Three-coloring

In this section we consider the problem whether $\chi(G) \leqslant 3$ for a graph G. Lawler describes the following algorithm, which is a pruning of the search tree.[6]

Consider a 3-coloring of G. Notice that the graph induced by vertices of any two colors is bipartite. A graph is bipartite when it has a 2-coloring.

[5] A. Björklund, T. Husfeldt and M. Koivisto, Set partitioning via inclusion-exclusion, *SIAM Journal on Computing* **39** (2009), pp. 546–563.

[6] E. L. Lawler, A note on the complexity of the chromatic number problem, *Information Processing Letters* **5** (1976), pp. 66–67.

Lawler's algorithm is very simple. Generate all maximal independent sets in G and check if $G - S$ is bipartite for some maximal independent set S. It is easy to see that one can check if a graph is bipartite in linear time. Thus the time needed to check if a graph G can be colored with three colors is simply the time needed to list all the maximal independent sets, *i.e.*, $O^*(1.4422^n)$.

Theorem 1.13. *There exists an* $O^*(1.4422^n)$ *algorithm which checks if a graph* G *with* n *vertices can be colored with three colors.*

Remark 1.14. Until now, the best algorithm for solving the 3-coloring problem is an algorithm by Eppstein.[7] This algorithm runs in $O^*(1.3289^n)$.

1.3 Domatic partition

Definition 1.15. *A dominating set in a graph* $G = (V, E)$ *is a set* $D \subseteq V$ *of vertices such that every vertex* $x \in V \setminus D$ *has at least one neighbor in* D.

In other words, a set $D \subseteq V$ is a dominating set in $G = (V, E)$ if and only if

$$N[x] \cap D \neq \varnothing \quad \text{for every vertex } x \in V,$$

where $N[x]$ is the closed neighborhood of x.

By definition, if $G = (V, E)$ is a graph, then $V \neq \varnothing$. It follows that the empty set is not a dominating set in G. On the other hand, for any graph $G = (V, E)$, the set V is a dominating set. Also, any maximal independent set in G is a dominating set.

The dominating set problem asks for a dominating set of minimal cardinality. Usually, one denotes the minimal cardinality of a dominating set in G by $\gamma(G)$.

Remark 1.16. It is easy to see that the dominating set problem can be solved in $O^*(2^n)$ time, by simply testing every subset of V. In Section 1.6 we show that the problem can be solved in $O^*(1.7088^n)$. In their recent book, Fomin and Kratsch improve this. They show that the dominating set problem can be solved in $O^*(1.5259^n)$ time.[8]

[7] D. Eppstein, Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction, *Proceedings of the* 12th *ACM-SIAM Symposium on Discrete Algorithms* SIAM, 2001.

[8] Corollary 6.11 in: F. Fomin and D. Kratsch, *Exact exponential algorithms*, Springer, 2010.

The dominating set problem is NP-complete. Notice that it is easy to check whether a set D is a dominating set in G or not, namely, simply check if each vertex $x \in V \setminus D$ has a neighbor in D. Since there are 2^n subsets of V, there is an easy $O^*(2^n)$ algorithm which solves the dominating set problem.

Definition 1.17. *A domatic partition of a graph* $G = (V, E)$ *is a partition*

$$\{ D_1, \ldots, D_k \}$$

of V *such that each* D_i *is a dominating set in* G.

Remark 1.18. The set $\{D_1, \ldots, D_k\}$ is a partition of V if

(i) for each $1 \leqslant i \leqslant k$, $D_i \neq \varnothing$, and
(ii) for all $1 \leqslant i < j \leqslant k$, $D_i \cap D_j = \varnothing$, and
(iii) $\cup_{i=1}^{k} D_i = V$.

The domatic partition problem asks for a domatic partition of the graph with a maximal number of dominating sets.

The domatic partition problem is NP-complete. In this section we show that the domatic partition problem can be solved in $O^*(3^n)$ time on graphs with n vertices.

Let $G = (V, E)$ be a graph and let $X \subseteq V$ be some subset of vertices. An X-partition is a partition

$$\{ D_1, \ldots, D_\ell \}$$

of X such that each D_i is a dominating set in G. Notice that there can be an X-partition only if X is a dominating set in G.

If X is a dominating set in G then let $f(X)$ denote the maximal ℓ for which there is an X-partition into ℓ dominating sets. If X is not a dominating set then let $f(X) = 0$. Notice that $f(V)$ solves the domatic partition problem.

Our algorithm computes the maximal ℓ for which there is an X-partition into ℓ dominating sets for each $X \subseteq V$.

Consider subsets X of V in order of increasing cardinality. Let X be a subset of cardinality k. We assume that for all subsets X' with $|X'| < |X|$ the algorithm has determined $f(X')$. By definition, if X is not a dominating set then $f(X) = 0$. Otherwise,

$$f(X) = \max \{ 1 + f(X \setminus Y) \mid Y \subseteq X \quad \text{and } Y \text{ is a dominating set in } G \}. \tag{1.4}$$

This simple observation gives us the following theorem.

Theorem 1.19. *There exists an* $O^*(3^n)$ *algorithm which solves the domatic partition problem for graphs with* n *vertices.*

Proof. The timebound is upperbounded by

$$\sum_{k=0}^{n}\binom{n}{k}\sum_{\ell=0}^{k}\binom{k}{\ell}=\sum_{k=0}^{n}\binom{n}{k}2^k=3^n.$$

□

Remark 1.20. Notice that the algorithm described above uses exponential space. The result can be improved to $O^*(2.8718^n)$ time and polynomial space.[9]

1.4 The traveling salesman problem

Suppose there are n cities, numbered $1,\dots,n$. A traveling salesman has to visit the cities $1,\dots,n$. He starts in city 1 and travels through the remaining cities in arbitrary order and at the end he returns to city 1. The distance between city i and j is denoted by $d(i,j)$. The problem is to minimize the total travel length.

The traveling salesman problem is NP-hard.

For each subset $S \subseteq \{2,\dots,n\}$ and for each $i \in S$ let $f(S,i)$ denote the length of a shortest path that starts in city 1, then visits the cities in $S \setminus \{i\}$ in some order and then stops in city i.

We now have

$$f(S,i)=\begin{cases} d(1,i) & \text{if } S=\{i\} \text{ and} \\ \min\{\, f(S\setminus\{i\},j)+d(j,i) \mid j\in S\setminus\{i\}\,\} & \text{otherwise.}\end{cases} \tag{1.5}$$

The following theorem was proved by Held and Karp.[10]

Theorem 1.21. *There exists an* $O^*(2^n)$ *algorithm which computes the minimum length of a traveling salesman tour.*

[9] Proposition 8 in: A. Björklund, T. Husfeldt and M. Koivisto, Set partitioning via inclusion-exclusion, *SIAM Journal on Computing* **39** (2009), pp. 546–563.

[10] M. Held and R. Karp, A dynamic programming approach to sequencing problems, *Journal of SIAM* **10** (1962), pp. 196–210.

Proof. The algorithm processes the subsets $S \subseteq \{2, \ldots, n\}$ in order of increasing cardinality.

The values $f(S, i)$ can be computed by Formula (1.5) in $O(n)$ time. Notice that the solution to the traveling salesman problem is given by the value of

$$\min \{ \, f(\{2, \ldots, n\}, j) + d(j, 1) \mid 2 \leqslant j \leqslant n \, \}. \tag{1.6}$$

There are $O(n \cdot 2^n)$ pairs (S, i) where S is a subset of $\{2, \ldots, n\}$ and $i \in S$. The computation of each $f(S, i)$ takes $O(n)$ time and so the overall complexity is bounded by $O(n^2 \cdot 2^n) = O^*(2^n)$ time. □

Remark 1.22. As far as I know, there is no algorithm that solves the traveling salesman problem in time $O^*(c^n)$ for $c < 2$.

1.5 Set cover

Let U be a finite set and let $n = |U|$. Let

$$\mathcal{S} = \{ \, S_1, \; \ldots, \; S_m \, \}$$

be a collection of subsets of U. A subset $\mathcal{S}' \subseteq \mathcal{S}$ is a cover for U if

$$\bigcup_{S \in \mathcal{S}'} S = U.$$

The set cover problem asks for a cover $\mathcal{S}'$ such that $|\mathcal{S}'|$ is minimal.

Of course, a cover can exist only if every element of U is an element of at least one subset S_i.

Consider the sets

$$\mathcal{S}_i = \{S_1, \ldots, S_i\} \quad \text{for } i = 1, \ldots, m.$$

Let $U' \subseteq U$ and let $f(U', i)$ be the minimal cardinality of a cover for U' with subsets from $\mathcal{S}_i$. If

$$U' \not\subseteq \bigcup_{S \in \mathcal{S}_i} S$$

then define $f(U', i) = \infty$.

The values $f(U', i)$, for $U' \subseteq U$ and $i = 1, \ldots, m$, can be computed via the following formulas. First consider the case $i = 1$. Then

$$f(U',1) = \begin{cases} 1 & \text{if } U' \subseteq S_1 \text{, and} \\ \infty & \text{otherwise.} \end{cases} \tag{1.7}$$

The value of $f(U', i+1)$ follows from the following formula.

$$f(U', i+1) = \begin{cases} \infty & \text{if } \cup_{S \in \mathcal{S}_{i+1}} S \neq U' \text{, and} \\ \min \{\, f(U', i), 1 + f(U' \setminus S_{i+1}) \,\} & \text{otherwise.} \end{cases} \tag{1.8}$$

Here is the proof that Formula (1.8) is correct. Either the set S_{i+1} is used to cover U' and then $U' \setminus S_{i+1}$ has to be covered with sets from $\mathcal{S}_i$, or the set S_{i+1} is not used in the cover, and then U' has to be covered with subsets from $\mathcal{S}_i$.

Theorem 1.23. *There exists an* $O^*(2^n)$ *algorithm for the set cover problem.*

Proof. The algorithm processes the subsets $U' \subseteq U$ in order of increasing cardinality. For each U' the algorithm computes the values $f(U', i)$ for increasing $i = 1, \ldots, m$ via the Formulas (1.7) and (1.8).

There are $O(m \cdot 2^n)$ pairs (U', i). First consider Formula (1.7). It can be checked in $O(n)$ time if U' is covered by S_1 or not. Therefore, the values $f(U', 1)$ can be determined in $O(n \cdot 2^n)$ time.

Now consider the computation of $f(U', i+1)$ via Formula (1.8). In this case the algorithm needs to compute $U' \setminus S_{i+1}$, which takes $O(n)$ time. Thus the total time complexity is bounded by $O(nm \cdot 2^n) = O^*(2^n)$.
This proves the theorem. □

1.6 Dominating set

Recall Definition 1.15. Let $G = (V, E)$ be a graph. A dominating set for G is a set $D \subseteq V$ such that every vertex $x \in V \setminus D$ has at least one neighbor in D. The dominating set problem asks for a dominating set of minimal cardinality.

The dominating set problem can be reduced to the set cover problem as follows. Let $U = V$ and let

$$\mathcal{S} = \{\, N[x] \mid x \in V \,\}.$$

A solution for this set cover problem corresponds to a solution for the dominating set problem. This can be seen as follows.

Assume that

$$\mathcal{S}' = \{ N[x] \mid x \in V' \},$$

is a solution for the set cover problem, for some $V' \subseteq V$. Then V' is a dominating set in G because every vertex $y \in V \setminus V'$ is covered by some set in $\mathcal{S}'$ and so $y \in N[z]$ for some $z \in V'$.

To see the converse, let D be a dominating set in G. Then let

$$\mathcal{S}' = \{ N[x] \mid x \in D \}.$$

Since every vertex $y \in V \setminus D$ has a neighbor in D, there exists some $z \in D$ such that $y \in N[z]$. Thus $\mathcal{S}'$ covers V.

Let I be a maximal independent set and let $W = V \setminus I$. Our algorithm finds for every subset $D' \subseteq W$ an *extension* $\mathcal{E}(D') \subseteq I$ such that $D' \cup \mathcal{E}(D')$ is a dominating set and such that $|\mathcal{E}(D')|$ is minimal. Notice that, when we have an extension for every $D' \subseteq W$, then this solves the domination problem, namely by taking the set $D' \subseteq W$ which minimizes

$$|D' \cup \mathcal{E}(D')| = |D'| + |\mathcal{E}(D')|.$$

Let $D' \subseteq W$ and define

$$I(D') = \{ x \in I \mid x \text{ has no neighbor in } D' \}.$$

Then any extension of D' contains $I(D')$.

The vertices that have no neighbors in $D' \cup I(D')$ are the vertices of

$$X = \{ x \in W \mid N[x] \cap (D' \cup I(D')) = \varnothing \}. \tag{1.9}$$

To obtain a minimum extension of D' we need to add a minimum set Q of vertices from $I \setminus I(D')$ such that every vertex of X has a neighbor in Q.

We now show how to compute minimum extensions.
Order the vertices of I, say

$$I = \{ x_1, \ldots, x_t \}.$$

Let $X \subseteq W$. For $\ell = 1, \ldots, t$ define

$$I_\ell = \{ x_1, \ \ldots, \ x_\ell \}.$$

For each $X \subseteq W$ and for $\ell = 1, \ldots, t$, define $f(X, \ell)$ as a subset $Q_\ell(X) \subseteq I_\ell$ such that every vertex of X has a neighbor in $Q_\ell(X)$ and such that $|Q_\ell(X)|$ is minimal. We obtain formulas similar to (1.7) and (1.8).

First consider the case $\ell = 1$.

$$f(X,1) = \begin{cases} \{x_1\} & \text{if } X \subseteq N(x_1), \text{ and} \\ I & \text{otherwise.} \end{cases} \tag{1.10}$$

For $f(X, \ell+1)$ we have the following recurrence relation.

$$f(X,\ell+1) = \begin{cases} f(X,\ell) & \text{if } |f(X,\ell)| < 1 + |f(X \setminus N(x_{\ell+1}),\ell)| \\ f(X \setminus N(x_{\ell+1}),\ell) \cup \{x_{\ell+1}\} & \text{otherwise.} \end{cases} \tag{1.11}$$

According to Theorem 1.23 the sets $f(X, \ell)$ for $X \subseteq W$ and $\ell = 1, \dots, t$ can be computed in $O^*(2^{|W|})$ time.

Now let $D' \subseteq W$. Then the cardinality of an extension $\mathcal{E}(D')$ of D' is

$$|\mathcal{E}(D')| = |I(D') \cup f(X,t)|,$$

where $t = |I|$ and X is defined by Formula (1.9).

It follows that the time for solving the dominating set problem is bounded by some polynomial factor times

$$2^{|W|} + \sum_{D' \subseteq W} \binom{|W|}{|D'|} = 2^{|W|+1}. \tag{1.12}$$

Theorem 1.24. *There exists an* $O^*(1.7088^n)$ *algorithm which solves the dominating set problem on graphs with* n *vertices.*

Proof. We consider two cases. First assume that $|I| > \alpha n$, for some

$$0.2271 < \alpha < 0.22711.$$

Then, according to Formula (1.12), a minimum dominating set can be computed in

$$O^*(2^{(1-\alpha)n}) = O^*(2^{0.7729n}) = O^*(1.7088^n).$$

Now assume that $|I| \leqslant \alpha n$. Then $\gamma(G) \leqslant |I| \leqslant \alpha n$. In that case we test every subset of V with cardinality at most αn. Notice that

$$\binom{n}{\alpha n} \leqslant \binom{n}{0.22711n}.$$

By Stirling's formula one can obtain that

$$\sum_{i=0}^{\alpha n} \binom{n}{i} \leqslant 2^{h(\alpha)n} \quad \text{where} \quad h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha).$$

Some calculations show that $2^{h(\alpha)n} \leqslant 2^{0.7729n} \leqslant 1.7088^n$.
This proves the theorem. □

1.7 Subset sum

Let's do an easy one.

Let $a_1, \dots, a_n$ be n natural numbers, thus each a_i is a positive integer. Let also K be a natural number. The subset sum problem asks if there is a set $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} a_i = K$.

The subset sum problem is NP-complete.

Theorem 1.25. *There exists an* $O(K \cdot n)$ *algorithm which solves the subset sum problem.*

Proof. Define

$$S_i = \{\, 1,\ \dots,\ i \,\} \quad \text{for } i = 1, \dots, n.$$

For $i = 1, \dots, n$, our algorithm computes a set

$$\Omega_i = \{\, 0 \leqslant t \leqslant K \mid \text{there exists some } I \subseteq S_i \text{ such that } \sum_{i \in I} a_i = t \,\}.$$

Thus Ω_i contains all the sums that can be made with numbers from $\{a_1, \dots, a_i\}$ and which are at most K.

First consider $i = 1$. Then,

$$\Omega_1 = \begin{cases} \{\, 0,\ a_1 \,\} & \text{if } a_1 \leqslant K, \text{ and} \\ \{\, 0 \,\} & \text{if } a_1 > K. \end{cases} \tag{1.13}$$

Next, the set Ω_{i+1} can be computed via the following formula.

$$\Omega_{i+1} = \Omega_i \cup \{\, 0 \leqslant t \leqslant K \mid \text{there exists a } t' \in \Omega_i \text{ such that } t' + a_{i+1} = t \,\}. \tag{1.14}$$

This can be seen as follows. Let t' be a sum which can be made with numbers from $\{a_1, \dots, a_i\}$. Then t' and $t' + a_{i+1}$ can be made with numbers from $\{a_1, \dots, a_{i+1}\}$.

To see the converse, let t be a sum that can be made with numbers from $\{a_1, \dots, a_{i+1}\}$. If a_{i+1} is used to obtain the sum t, then $t' = t - a_{i+1}$ can be made with numbers from $\{a_1, \dots, a_i\}$. If a_{i+1} is not used to obtain the sum t then t is a sum with numbers from $\{a_1, \dots, a_i\}$.

It is easy to see that Ω_n can be computed via Formulas (1.13) and (1.14) in $O(K \cdot n)$ time. The answer to the subset problem is YES if $K \in \Omega_n$, and NO otherwise. □

1.8 Problems

1.1. In the maximum independent set algorithm we used a linear-time algorithm for the case where every vertex in G has degree at most two. Suppose that, instead, we keep branching until every vertex has degree at most one. Show that this changes Formula (1.2) on Page 5 into

$$T(n) \leqslant T(n-1) + T(n-2) + O(n+m). \tag{1.15}$$

Show that the solution of Formula (1.15) is the n^{th} Fibonacci number times some polynomial. Use the exact formula for the Fibonacci numbers to show that this gives

$$T(n) = O^*(1.6181^n).$$

1.2. Let x and y be two vertices of a graph G and assume that

$$N[x] \subseteq N[y].$$

Notice that this implies that x and y are adjacent. Show that

$$\alpha(G) = \alpha(G - y).$$

This is one of the prunings that is used in the algorithm of Fomin and Kratsch.

1.3. Let $T(n)$ be the worst-case time-bound for any algorithm which solves the maximum independent set problem on graphs with n vertices. In this exercise we show that

$$T(n-1) \leqslant T(n) \quad \text{for all } n > 1.$$

Suppose that $T(n-1) > T(n)$ for some $n > 1$. We obtain a contradiction as follows.

(a) Let G be a graph with $n-1$ vertices. Let G' be the graph obtained from G by adding an isolated vertex to G. Prove that

$$\alpha(G') = \alpha(G) + 1.$$

(b) Show that this proves that the maximum independent set problem for G can be determined in $T(n)$ time. This contradicts the assumption that $T(n-1) > T(n)$.

1.4. Find a linear-time algorithm which checks if a graph is bipartite.

1.5. Find an exact algorithm for 4-coloring.

1.6. Check Formula (1.4) on Page 9.

1.7. Check Formula (1.5) on Page 10.

1.8. Let $G = (V, E)$ be a graph.

(a) A Hamiltonian cycle is a cycle in G which contains all vertices. Show that there is an $O^*(2^n)$ algorithm which solves the Hamiltonian cycle problem. `Hint:` Reduce the Hamiltonian cycle problem to the traveling salesman problem. For any two vertices i and j in G define

$$d(i,j) = \begin{cases} 1 & \text{if } \{i,j\} \in E, \text{ and} \\ \infty & \text{if } \{i,j\} \notin E. \end{cases} \tag{1.16}$$

(b) A Hamiltonian path is a path in G which contains all vertices. The difference with the Hamiltonian cycle problem is that the two endpoints of the path are not necessarily adjacent. Design an $O^*(2^n)$ algorithm that solves the Hamiltonian path problem on graphs with n vertices.

1.9. Check the Formulas (1.10) and (1.11) and show that the sets $f(X, \ell)$, as defined by these formulas, can be computed in $O^*(2^{|W|})$.

1.10. The subset sum problem described in Section 1.7 is NP-complete. Theorem 1.25 on Page 15 shows that it can be solved in $O(K \cdot n)$ time. Does this prove that $P = NP$?

2 Graph Classes

In this chapter we have a close look at a few important graph classes and we look at some NP-complete problems that become polynomial when the graphs are restricted to these.

A graph class, or a class of graphs, is simply a set of graphs. For example

$$\mathcal{G} = \{\ G \mid G \text{ is a planar graph }\} \tag{2.1}$$

is the class of all planar graphs. A class of graphs may be finite or infinite. The class above, of all planar graphs, is of course infinite (it contains an infinite number of elements).

Obviously, many NP-complete problems can become polynomial when one restricts the graphs to some special graph class. For example, the four-coloring problem is NP-complete but it can be solved trivially when one restricts the graphs to the class of planar graphs.

For algorithmic problems one considers usually only infinite classes of graphs. The reason is that for finite classes of graphs most problems can be solved in constant time by exhaustive search.

Usually, one restricts the research on infinite classes of graphs to classes that are *hereditary*. A class $\mathcal{G}$ of graphs is hereditary if $G \in \mathcal{G}$ implies that every induced subgraph of G is also in $\mathcal{G}$. For example, the class of planar graphs is hereditary, since if G is planar then so is every induced subgraph of G. All the classes that we study in this chapter are hereditary.

When one studies some graph class $\mathcal{G}$ then the membership of graphs in $\mathcal{G}$ is an important issue. One refers to this as the recognition problem for the class $\mathcal{G}$:
Input: A graph G.
Question: Is $G \in \mathcal{G}$?

For example, the planar graphs are recognizable in linear time, but there are many classes of graphs for which the recognition problem is not clear. For example, consider the class

$$\mathcal{H} = \{ G \mid G \text{ is a planar graph and } \chi(G) \leqslant 3 \}.$$

The recognition problem for $\mathcal{H}$ is NP-complete, since the 3-coloring problem is NP-complete for planar graphs.

A lot of research is done on subclasses of perfect graphs. All classes that we study in this chapter are perfect. We introduce the class of perfect graphs in the next section.

2.1 Perfect graphs

Definition 2.1. *A graph* G *is perfect if for every induced subgraph* H *of* G

$$\boxed{\chi(H) = \omega(H),}$$

where $\chi(H)$ *is the chromatic number of* H *and* $\omega(H)$ *is the clique number of* H.

If one wants to color a graph G such that adjacent vertices have different colors, then all vertices of a clique in G must receive different colors. Thus for all graphs we have

$$\chi(G) \geqslant \omega(G). \tag{2.2}$$

For perfect graphs equality holds, not only for the graph itself but also for all induced subgraphs of it. Notice that the class of perfect graphs is hereditary, simply by definition.

Perhaps we should emphasize this. Assume that for some graph G,

$$\chi(G) > \omega(G).$$

For example, if G is an odd cycle of length more than 3 then

$$\chi(G) = 3 \quad \text{and} \quad \omega(G) = 2.$$

It is easy to construct a graph G' such that $\chi(G') = \omega(G')$ by adding a clique of size at least $\chi(G)$ to G. The graph G' is of course not perfect, since G is an induced subgraph of G' and equality in (2.2) does not hold for G.

Since the structure of G' is not essentially different from the structure of G it cannot be expected that there are many problems that are easier to solve for

G' than for G. For example, a dominating set for G' consists of a dominating set in G plus one vertex in the clique that is added to G. Thus the dominating set problem for G' is just as hard as it is for G.

The complement of a graph G is the graph $\bar{G}$ with the same set of vertices as G, and with two vertices in $\bar{G}$ adjacent if and only if they are not adjacent in G. Concerning perfect graphs one of the first and most important theorems was proved by Lovász in 1972.[1]

Theorem 2.2 (The perfect graph theorem). *The complement of a perfect graph is perfect.*

Thus, if G is perfect then for every induced subgraph H of G we have that

$$\boxed{\alpha(H) = \kappa(H),} \tag{2.3}$$

where $\kappa(H) = \chi(\bar{H})$ is the smallest number of cliques that partition the set of vertices and $\alpha(H) = \omega(\bar{H})$ is the cardinality of a largest independent set in H.

In Exercise 2.2 we ask you to prove that bipartite graphs are perfect. By Theorem 2.2 also the complements of bipartite graphs are perfect. (Can you prove this without using Theorem 2.2?)

Another important example of perfect graphs is the set of linegraphs of bipartite graphs. The linegraph $L(G)$ of a graph G has as its vertices the edges of G and as its edges those pairs of edges in G that share an endpoint. As an introductory example, let us prove that linegraphs of bipartite graphs are perfect.

Lemma 2.3. *Let* G *be bipartite. Then* $L(G)$ *is perfect.*

Proof. First of all, it is sufficient to prove Equation (2.3) for $L(G)$, since removing an edge from a bipartite graph leaves it bipartite. (That is, the class of linegraphs of bipartite graphs is hereditary.)

Notice that an independent set in $L(G)$ is a set of edges in G of which no two share an endpoint, *i.e.*,

$$\alpha(L(G)) = \nu(G),$$

where $\nu(G)$ is the cardinality of a maximum matching in G.

One of the earliest results in graph theory is the Theorem of König-Egerváry:[2]

[1] L. Lovász, Normal hypergraphs and the perfect graph conjecture, *Discrete Mathematics* **2** (1972), pp. 253–267.

[2] D. Kőnig, Graphen und Matrizen, *Math. Lapok* **38** (1931), pp. 116–119.

If G is bipartite then $\nu(G) = \tau(G)$, where $\tau(G)$ is the cardinality of a smallest vertex cover in G.

A vertex cover is a set C of vertices such that every edge in G has at least one endpoint in C.

Let $C = \{x_1, \dots, x_s\}$ be a vertex cover of G. The set L_i of edges that have the vertex x_i in common forms a clique in $L(G)$. Since C is a vertex cover, every edge in G is in some L_i. Thus $\{L_1, \dots, L_s\}$ is a clique cover in $L(G)$. Possibly some pairs L_i and L_j are not disjoint, but it is easy to change the clique cover into a partition of the vertices of $L(G)$ into s cliques.

Thus, if C is a minimum vertex cover then

$$\alpha(L(G)) = \nu(G) = \tau(G) = |C| \geqslant \kappa(L(G)) \geqslant \alpha(L(G)),$$

since, by (2.2), $\alpha(H) \leqslant \kappa(H)$ for any graph H.

Thus $\alpha(L(G)) = \kappa(L(G))$ and by Theorem 2.2 this proves the lemma. □

Obviously, if a graph G is perfect then it cannot have an induced odd cycle of length at least five. By Theorem 2.2, when G is perfect it cannot have the complement of an induced cycle of length at least five as an induced subgraph. Claude Berge conjectured in 1961 that this characterizes perfect graphs.[3] The proof of the conjecture sent a shock wave through the graph theory community.[4]

Definition 2.4. *Let G be a graph. A hole in G is an induced cycle of length at least five. An antihole in G is an induced subgraph of G which is isomorphic to the complement of a cycle of length at least five. An odd hole in G is a hole of odd length. An odd antihole is an antihole of odd cardinality.*

Theorem 2.5 (The strong perfect graph theorem). *A graph is perfect if and only if it has no odd hole and no odd antihole.*

Theorem 2.5 was proved using a certain decomposition of the graph into four basic classes of perfect graphs, namely,

(1) bipartite graphs,
(2) complements of bipartite graphs,
(3) linegraphs of bipartite graphs, and
(4) complements of linegraphs of bipartite graphs.

[3] C. Berge, Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind, *Wiss. Zeitschr. Martin-Luther Univ. Halle-Wittenberg* **10** (1961), pp. 114–115.
[4] M. Chudnovsky, N. Robertson, P. Seymour and R. Thomas, The strong perfect graph theorem, *Annals of Mathematics* **164** (2006), pp. 51–229.

This decomposition theorem for perfect graphs also led to a polynomial algorithm for recognizing perfect graphs.[5]

Theorem 2.6. *There exists an* $O(n^9)$ *algorithm which tests if a graph with* n *vertices is perfect.*

One of the reasons for the popularity of perfect graphs is the following theorem.[6]

Theorem 2.7. *There exist polynomial algorithms to compute* $\omega(G)$ *and* $\chi(G)$ *for graphs* G *that satisfy* $\omega(G) = \chi(G)$.

The Shannon capacity of the complement of a graph is a graph parameter which is sandwiched between the clique number and chromatic number. In turn, the Lovász number is sandwiched between the Shannon capacity and the clique cover number. Thus, if we write $\Theta(G)$ for the Shannon capacity and $\vartheta(G)$ for the Lovász number, then

$$\omega(\bar{G}) = \alpha(G) \leqslant \Theta(G) \leqslant \vartheta(G) \leqslant \kappa(G) = \chi(\bar{G}). \tag{2.4}$$

The theorem above was proved by showing that the Lovász number $\vartheta(G)$ of a graph G is computable in polynomial time.[7]

2.2 Cographs

One of the most elegant classes of graphs is the class of cographs.

Definition 2.8. *A graph* G *is a cograph if it has no induced* P_4*, which is a path with four vertices.*

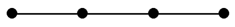

Fig. 2.1. A P_4 is a path with four vertices. Notice that $\bar{P_4} = P_4$.

By definition, the class of cographs is hereditary, namely, if G has no induced P_4 then no induced subgraph of G has an induced P_4.

[5] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour and K. Vušković, Recognizing Berge graphs, *Combinatorica* **25** (2005), pp. 143–186.

[6] M. Grötschel, L. Lovász and A. Schrijver, Polynomial algorithms for perfect graphs. In (Berge, Chvátal eds.): *Topics on perfect graphs*, North-Holland Math. Stud. **88** (1984), pp. 325–356.

[7] L. Lovász, On the Shannon capacity of a graph, *IEEE Transactions on Information Theory* **25** (1979), pp. 1–7.

Lemma 2.9. *If* G *is a cograph then* G *is perfect.*

Proof. Let G be a cograph. We show that G nor $\bar{G}$ has a hole (an induced cycle of length at least 5).

The graph G has no hole, otherwise it has an induced P_4. Notice that $\bar{P_4}$ is isomorphic to P_4. Thus if G is a cograph then $\bar{G}$ is also a cograph. Thus G has no antihole.

Since G has no hole it has no odd hole and the same holds true for $\bar{G}$. The claim now follows from Theorem 2.5. □

Theorem 2.10. *A graph* $G = (V, E)$ *is a cograph if and only if for every induced subgraph* H *of* G *one of the following properties holds.*

(a) H *has only one vertex, or*
(b) H *is disconnected, or*
(c) $\bar{H}$ *is disconnected.*

Proof. Notice that $\bar{P_4}$ is isomorphic to P_4. This implies that if G is a cograph then $\bar{G}$ is a cograph.

A graph with less than four vertices is a cograph. Therefore, since the class of cographs is hereditary, it is sufficient to prove that G or $\bar{G}$ is disconnected when G is a cograph with at least two vertices.

It is easy to check that the claim holds true when G has at most three vertices.

Assume that G has at least four vertices. Assume also that G is connected. Let x be a vertex of G. By induction $G - x$ or the complement of $G - x$ is disconnected.

Assume that $G - x$ is disconnected and let $C_1, \ldots, C_t$ be the components of $G - x$. Assume that x has a neighbor and a nonneighbor in C_1. Then there exist vertices a and b in C_1 such that $[a, b, x]$ is an induced P_3 in G. Since G is connected, x has a neighbor c in C_2. Now $[a, b, x, c]$ is an induced P_4 which is a contradiction. Thus x is adjacent to all other vertices in G. Then $\bar{G}$ is disconnected, since $\{x\}$ is a component of $\bar{G}$.

Assume that the complement of $G - x$ is disconnected. (We now 'copy' the argument above.) Let $C'_1, \ldots, C'_s$ be the components of the complement of $G - x$. Assume that x has a neighbor a' and a nonneighbor b' in C'_1. Since $\bar{G}[C'_1]$ is connected and since there is no induced P_4, a' and b' are nonadjacent in G.

If x is adjacent to all vertices in C'_2 then $\bar{G}$ is disconnected with a component C'_2. Thus x has a nonneighbor c' in C_2. Now $[x, a', c', b']$ is an induced P_4, which is a contradiction.

Thus we may assume that x is not adjacent to any other vertex in G, and so G is disconnected.
This proves the theorem. □

2.2.1 Cotrees

Let G be a cograph. By Theorem 2.10 we can build a decomposition tree for the graph G.

The decomposition tree is a pair (T, f) where T is a binary tree and f is a bijection from the leaves of T to the vertices of G. Each internal node, including the root, is labeled with a $\otimes$ - or an $\oplus$-operator. Consider an internal vertex t. Let V_1 and V_2 be the two sets of vertices in G that are mapped to the leaves of the left - and right subtree. If t is labeled with $\otimes$ then every vertex of V_1 is adjacent to every vertex of V_2. If t is labeled by $\oplus$ then no vertex of V_1 is adjacent to any vertex of V_2.

Let G be a cograph. We can build a decomposition tree as follows. If G has only one vertex, the tree consists of a single leaf, which is mapped by f to the vertex of G.

Otherwise, by Theorem 2.10, either G or $\bar{G}$ is disconnected.

Assume that G is disconnected and let $C_1, \ldots, C_t$ be the components of G. Group the components into two nonempty sets, say A and B. Recursively, build decomposition trees (T_1, f_1) and (T_2, f_2) for $G[A]$ and $G[B]$. Create a new root, and make it adjacent to the root of T_1 and to the root of T_2. Label the new root by $\oplus$.

Assume that $\bar{G}$ is disconnected. In that case, build a decomposition tree $(\bar{T}, \bar{f})$ for $\bar{G}$ as described above. Change all labels from $\oplus$ to $\otimes$ and *vice versa*.

A decomposition tree for cographs as described above is called a cotree. Notice that a graph G is a cograph if and only if it has a cotree (see Exercise 2.6).

Corneil, Perl and Stewart proved the following theorem.[8]

Theorem 2.11. *There exists a linear-time algorithm that recognizes cographs. When* G *is a cograph then this algorithm builds a cotree for* G.

2.2.2 Finding cliques in cographs

To illustrate the usefulness of cotrees, we show how to use it for the computation of the clique number.

Theorem 2.12. *There exists a linear-time algorithm that computes the clique number* $\omega(G)$ *of a cograph* G.

[8] D. Corneil, Y. Perl and L. Stuwart, A linear recognition algorithm for cographs, *SIAM Journal on Computing* **14** (1985), pp. 926–934.

Proof. let $G = (V, E)$ be a cograph. First, the algorithm builds a cotree (T, f) for G. By Theorem 2.11 this step takes linear time.

First assume that G has only one vertex. Then $\omega(G) = 1$.

Now assume that $|V| \geqslant 2$. For an internal node p of T let V_1 and V_2 be the two sets of vertices that are mapped to the leaves in the left - and right subtree. Let $G_i = G[V_i]$ for $i \in \{1, 2\}$ and let

$$G_p = G[V_1 \cup V_2].$$

First assume that the label of p in T is $\otimes$. Then G_p is the *join* of G_1 and G_2, that is, every vertex of G_1 is adjacent to every vertex of G_2. Notice that

$$\omega(G_p) = \omega(G_1) + \omega(G_2). \tag{2.5}$$

Now assume that the label of p is $\oplus$. Then G_p is the *union* of G_1 and G_2, that is, no vertex of G_1 is adjacent to any vertex of G_2. Now

$$\omega(G_p) = \max\{\, \omega(G_1),\ \omega(G_2)\,\} \tag{2.6}$$

Assume that p is the root of T. The algorithm recursively computes $\omega(G_1)$ and $\omega(G_2)$. Since p is the root, $G_p = G$ and $\omega(G)$ follows from Formulas (2.5) and (2.6).

The amount of work in each internal node takes $O(1)$ time. So, in total the algorithm runs in time $O(n)$, where $n = |V|$, since the depth of the cotree is at most n. In other words, when the cotree is a part of the input this algorithm runs in $O(n)$ time.
This proves the theorem. □

2.3 Distance-hereditary graphs

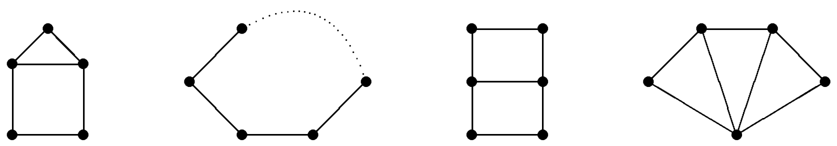
Fig. 2.2. A graph is distance hereditary if it has no induced house, hole, domino or gem.

Edward Howorka introduced distance-hereditary graphs.[9]

[9] E. Howorka, A characterization of distance-hereditary graphs, *The Quarterly Journal of Mathematics* **28** (1977), pp. 417–420.

Definition 2.13. *A graph* G *is distance hereditary if for every pair of nonadjacent vertices* x *and* y *and for every connected, induced subgraph* H *of* G *which contains* x *and* y*, the distance between* x *and* y *in* H *is the same as the distance between* x *and* y *in* G.

In other words, a graph G is distance hereditary if for every nonadjacent pair x and y of vertices, all chordless paths between x and y in G have the same length. (A path P is chordless if $G[P]$ is a path; that is, P has no shortcuts.)

Notice that, by definition the class of distance-hereditary graphs is hereditary.

There are various characterizations of distance-hereditary graphs. One of them states that a graph is distance hereditary if and only if it has no induced house, hole, domino or gem.

Distance-hereditary graphs are also characterized by the property that every induced subgraph has either an isolated vertex, or a pendant vertex, or a twin. An isolated vertex is a vertex with no neighbors. A pendant vertex is a vertex with exactly one neighbor. A twin is a pair of vertices x and y such that either

$$N(x) = N(y) \quad \text{or} \quad N[x] = N[y].$$

Let G be distance hereditary and let G' be the graph obtained from G by adding an isolated vertex or a pendant vertex, or a twin x of some vertex y in G. In Exercise 2.10 we ask you to check that G' is also distance hereditary.

Theorem 2.14. *Distance-hereditary graphs are perfect.*

Proof. Let $G = (V, E)$ be distance hereditary. Then G has no hole. It remains to show that G has no odd antihole.

Let $x \in V$ be a vertex which is either isolated, or a pendant adjacent to some vertex y, or a twin of some vertex y. Consider $G - x$. By induction we may assume that $G - x$ has no odd hole or odd antihole.

Assume that G has an odd antihole H. Let V' be the vertex set of H. Then $x \in V'$. Since H is connected, x is not isolated. Since H is biconnected, x is not a pendant vertex. Thus x is a twin of some vertex y. Notice that $y \notin V'$ is not a vertex of H, since H has no twins. Let

$$V'' = (V' \setminus \{x\}) \cup \{y\}.$$

Then V'' induces an odd antihole in $G - x$ which is a contradiction.
This proves the theorem. □

2.3.1 Decomposition trees for DH-graphs

A decomposition tree for a graph $G = (V, E)$ is a pair (T, f) consisting of a rooted binary tree T and a bijection f from V to the leaves of T.

When G is distance hereditary it has a decomposition tree (T, f) with the following three properties.[10]

Consider an edge $e = \{p, c\}$ in T where p is the parent of c. Let $W_e \subset V$ be the set of vertices of G that are mapped by f to the leaves in the subtree rooted at c. Let $Q_e \subseteq W_e$ be the set of vertices in W_e that have neighbors in $G - W_e$. The set Q_e is called the twinset of e. The first property is that the subgraph of G induced by Q_e is a cograph for every edge e in T.

Consider an internal vertex p in T. Let c_1 and c_2 be the two children of p. Let $e_1 = \{p, c_1\}$ and let $e_2 = \{p, c_2\}$. Let Q_1 and Q_2 be the twinsets of e_1 and e_2. The second property is that there is a join- or a union-operation between Q_1 and Q_2. Thus either all vertices of Q_1 are adjacent to all vertices of Q_2, or they are not adjacent to any vertex of Q_2. As in cotrees, there is a label $\oplus$ or $\otimes$ at the vertex p in T that indicates which operation is performed on Q_1 and Q_2.

Remark 2.15. Notice the difference with the labels in cotrees. The $\oplus$- or $\otimes$-operator in the decomposition tree for distance-hereditary graphs works on the twinsets, and not, as in the cotrees, on all the vertices of W_{e_1} and W_{e_2}.

Let p be an internal vertex of T which is not the root. Let e be the line that connects p with its parent. Let Q_e be the twinset of e. Let c_1 and c_2 be the two children of p in T. Let $e_1 = \{p, c_1\}$ and let $e_2 = \{p, c_2\}$. Let Q_i be the twinset of e_i, for $i \in \{1, 2\}$. The third, and final, property is that

$$Q_e = \varnothing \quad \text{or} \quad Q_e = Q_1 \quad \text{or} \quad Q_e = Q_2 \quad \text{or} \quad Q_e = Q_1 \cup Q_2.$$

The vertex p in T has an extra label that indicates which of these four operations that define Q_e occur.

Notice that the first property is a consequence of the other two. As an example, notice that cographs are distance hereditary. A cotree is a decomposition tree for a cograph with the three properties mentioned above.

Lemma 2.16. *Let G be distance hereditary. Then G has a decomposition tree as described above.*

[10] P. Hammer and F. Maffray, Completely separable graphs, *Discrete Applied Mathematics* **27** (1990), pp. 85–99.

Proof. Let $G = (V, E)$ be distance hereditary. We use the property that G has an isolated vertex, a pendant vertex or a twin.

Let x be an isolated vertex, a pendant vertex with neighbor y, or a twin of a vertex y. Let $G' = G - x$. By induction G' has a decomposition tree (T', f') as described above. We now show how to construct a decomposition tree (T, f) for G.

First assume that x is a twin of a vertex y. The vertex y is mapped by f' to some leaf ℓ of T'. Create two leaves ℓ_1 and ℓ_2 and let ℓ be the parent of ℓ_1 and ℓ_2 in T. Let f map x to ℓ_1 and y to ℓ_2, and let f be the same as f' for all other vertices $z \in V \setminus \{x, y\}$. If x and y are adjacent, then the new internal node ℓ receives an $\otimes$-operator and otherwise it receives an $\oplus$-operator. Finally, update the twinsets by adding x to all twinsets that contain y. The edges $\{\ell, \ell_1\}$ and $\{\ell, \ell_2\}$ have twinsets $\{x\}$ and $\{y\}$.

Assume that x is a pendant vertex with a neighbor y. The tree T and the map f are obtained in the same manner as described above. The internal node ℓ receives an $\otimes$-operator since x and y are adjacent. Finally, x appears in only one twinset, namely in the twinset of the new edge $\{\ell, \ell_1\}$ in T.

Assume that x is isolated in G. Choose an arbitrary vertex y in $V \setminus \{x\}$. Create the tree T and the map f as above. In this case, the vertex x appears in no twinset.
This proves the lemma. □

When G is distance hereditary then a tree-decomposition for G with the three properties described above can be obtained in linear time.

2.3.2 Feedback vertex set in DH-graphs

To illustrate the usefulness of the decomposition tree for DH-graphs we show how it is used to solve the feedback vertex set problem.

Definition 2.17. *Let $G = (V, E)$ be a graph. A set $F \subseteq V$ is a feedback vertex set if $G - F$ has no cycles, that is, $G - F$ is a forest.*

Let G be a graph. The feedback vertex set problem asks for a feedback vertex set in G of minimal cardinality. The feedback vertex set problem is NP-complete.

A graph G is a join of two graphs G_1 and G_2 if G is obtained from the graphs G_1 and G_2 by making every vertex of G_1 adjacent to every vertex of G_2.

A graph G is the union of two graphs G_1 and G_2 if

1. the vertex set of G is the union of the vertex sets of G_1 and G_2, and
2. the edge set of G is the union of the edge sets of G_1 and G_2.

Lemma 2.18. *If G is the join of G_1 and G_1 and if F is a feedback vertex set of G then*

$$|V_1 - F| \leqslant 1 \quad \textit{or} \quad |V_2 - F| \leqslant 1 \quad \textit{or both.}$$

Proof. Otherwise $G - F$ contains a 4-cycle. □

Lemma 2.19. *Let G be a cograph which is the join of cographs G_1 and G_2. Let F be a feedback vertex set of G. Assume that $|V_1 - F| = 1$. Then $G_2 - F$ is an independent set.*

Proof. Otherwise $G - F$ contains a triangle. □

Theorem 2.20. *There exists a linear-time algorithm that solves the feedback vertex set problem on distance-hereditary graphs.*

Proof. Let $G = (V, E)$ be distance hereditary. First construct a decomposition tree (T, f) for G. This takes linear time.

Extend the tree with a new root r' and make the parent of the old root r this new root r'. Define the twinset of the edge $\{r, r'\}$ as $\varnothing$. For ease of description, call this new decomposition tree again (T, f).

Let p be an internal vertex of T which is not the root r' and let c_1 and c_2 be the two children of p. Let W_1 and W_2 be the two sets of vertices that are mapped to the leaves in the subtrees at c_1 and c_2 respectively and let $W = W_1 \cup W_2$. Let $Q_1 \subseteq W_1$ and $Q_2 \subseteq W_2$ be the two twinsets of $\{p, c_1\}$ and $\{p, c_2\}$. The $\otimes$ or $\oplus$ label at p indicates whether there is a join or a union of the two twinsets Q_1 and Q_2.

Let e be the edge in T that connects p with its parent p' (possibly $p' = r'$) and let $e_i = \{p, c_i\}$ for $i \in \{1, 2\}$. The twinset Q_e is either one of Q_1 or Q_2, or it is $Q_1 \cup Q_2$, or it is the empty set.

In our dynamic programming algorithm we maintain the following four values for each edge e in T with twinset Q:

(1) the minimal cardinality of a feedback vertex set of $G[W]$;
(2) the minimal cardinality of a feedback vertex set F of $G[W]$ such that

$$Q - F \quad \text{induces an independent set;}$$

(3) the minimal cardinality of a feedback vertex set F of $G[W]$ such that

$$|Q - F| = 1 \quad \text{and}$$

(4) the minimal cardinality of a feedback vertex set F of $G[W]$ with

$$Q \subseteq F.$$

It is easy to see that these four values for an edge $e = \{p, p'\}$ in T can be obtained from the values at the edges $\{p, c_1\}$ and $\{p, c_2\}$ (see Exercise 2.11).

The minimal cardinality of a feedback vertex set for G can be read from the first value *i.e.*, Item (1) above, at the root-edge $\{r, r'\}$ of the binary tree-decomposition.
The completes the proof. □

2.4 Chordal graphs

One of the oldest classes of graphs that have been studied in great detail is the class of chordal graphs.

Definition 2.21. *A graph is chordal if it has no induced cycle of length more than three.*

For example, the class of chordal graphs contains all trees.

Notice that, by definition, the class of chordal graphs is hereditary. Let's first prove that chordal graphs are perfect.

Theorem 2.22. *Chordal graphs are perfect.*

Proof. Let $G = (V, E)$ be chordal. Then, by definition, G has no holes. We show that G has no antiholes. Since $\bar{C}_5 = C_5$ any antihole must have at least six vertices.

Notice that any cycle of length at least six has an induced $2K_2$, which is the complement of a 4-cycle. Thus G has no antihole. □

Our first characterization of chordal graphs is in terms of minimal separators.

Definition 2.23. *Let* $G = (V, E)$ *be a graph and let* x *and* y *be nonadjacent vertices. A set*

$$S \subseteq V \setminus \{x, y\}$$

is an x, y*-separator if* x *and* y *are in different components of* $G - S$.

Definition 2.24. *An x, y-separator S is a minimal x, y-separator if no proper subset of S is an x, y-separator.*

Definition 2.25. *A set S is a minimal separator if there exist nonadjacent vertices x and y such that S is a minimal x, y-separator.*

Remark 2.26. Notice that one minimal separator may properly contain another minimal separator. For example, consider a 4-cycle $[a, b, c, d]$. Add a pendant vertex e adjacent to c. Then $\{a, c\}$ is a minimal b, d-separator and $\{c\}$ is a minimal a, e-separator.

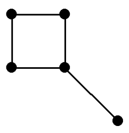

Fig. 2.3. This graph has a minimal separator contained in another one.

Theorem 2.27. *A graph is chordal if and only if every minimal separator is a clique.*

Proof. Assume that $G = (V, E)$ is chordal. Let S be a minimal x, y-separator for nonadjacent vertices x and y in G. Let C_x and C_y be the components of $G - S$ that contain x and y.

Notice that every vertex of S has at least one neighbor in C_x and at least one neighbor in C_y. To see this, assume some $z \in S$ has no neighbors in C_x. Then $S \setminus \{z\}$ is also a minimal x, y-separator. This contradicts the minimality of S.

Now assume that S is not a clique. Then S contains two vertices a and b that are not adjacent. Consider two chordless paths P_x and P_y from a to b. One with internal vertices in C_x and the other with internal vertices in C_y.

A chordless path is a path without a chord, that is, the path is induced. (In Exercise 2.13 we ask you to prove that P_x and P_y exist.) The two paths together form an induced cycle of length at least four.
This proves the theorem. □

Our second characterization of chordal graphs is in terms of simplicial vertices.

Definition 2.28. *Let* $G = (V, E)$ *be a graph. A vertex* x *in* G *is simplicial if* $N(x)$ *induces a clique in* G.

Theorem 2.29. *A graph is chordal if and only if every induced subgraph has a simplicial vertex.*

Proof. Let $G = (V, E)$ be a graph. First assume that every induced subgraph of G has a simplicial vertex. Let Ω be a subset of vertices such that $G[\Omega]$ is a cycle of length at least four. Then $G[\Omega]$ is an induced subgraph of G without simplicial vertex. This is a contradiction.

Now assume that G is chordal. Let x be a vertex such that the largest component C of $G - N[x]$ is as large as possible. Let $S \subseteq N(x)$ be the set of neighbors of x that have a neighbor in C. Then S is a minimal x, y-separator for any $y \in C$. Thus S is a clique.

We claim that every vertex of

$$V \setminus (C \cup S)$$

is adjacent to all vertices in S.

Clearly, x is adjacent to every vertex in S since $S \subseteq N(x)$. Assume that there exists a vertex

$$z \in V \setminus (S \cup C \cup \{x\})$$

which is not adjacent to some vertex $s \in S$. Then $C \cup \{s\}$ is contained in a component of $G - N[z]$ since $G[C]$ is connected and s has a neighbor in C. Thus $G - N[z]$ has a component which is larger than C. This contradicts the choice of x.

Let

$$G' = G - (S \cup C).$$

By induction we may assume that G' has a simplicial vertex z. Let $N'(z)$ be the neighborhood of z in G'. Then $N'(z)$ is a clique. Notice that

$$N(z) = N'(z) \cup S.$$

Now $N(z)$ is simplicial, since

(i) S is a clique, and
(ii) $N'(z)$ is a clique, and
(iii) every vertex of $N'(z)$ is adjacent to every vertex of S.

This proves that z is simplicial in G. □

Definition 2.30. *Let* $G = (V, E)$ *be a chordal graph. A perfect elimination ordering for* G *is an ordering of the vertices*

$$[x_1, \ldots, x_n]$$

such that for $i = 1, \ldots, n$ *the vertex* x_i *is simplicial in the subgraph of* G *induced by* $\{x_i, \ldots, x_n\}$.

Theorem 2.31. *A graph is chordal if and only if it has a perfect elimination ordering.*

Proof. This is an immediate consequence of Theorem 2.29. □

In Exercise 2.17 we ask you to prove that a perfect elimination ordering in a chordal graph can be obtained in linear time. One of the oldest and easiest algorithms to do this is an algorithm of Tarjan and Yannakakis.[11] Their algorithm computes an ordering $[x_1, \ldots, x_n]$ of the vertices in any graph. This ordering is a perfect elimination ordering if and only if the graph is chordal. Their paper describes first a linear-time algorithm that computes an ordering and next it describes a linear-time test to see if the ordering is a perfect elimination ordering.

Their algorithm computes an ordering as follows. It labels the vertices one by one. In each step, the unlabeled vertex that has the most labeled neighbors is labeled next. Ties are broken arbitrarily (so, the first vertex to get a label is arbitrary). This produces the perfect elimination ordering backwards, *i.e.*, the last vertex that gets a label is the first vertex in the perfect elimination ordering.

Lemma 2.32. *Every chordal graph has at most* n *maximal cliques, where* n *is the number of vertices in the graph.*

Proof. Let $G = (V, E)$ be a chordal graph. Let x be a simplicial in G. The only maximal clique in G that contains x is $N[x]$. Thus all other maximal cliques in G are maximal cliques in $G - x$. The claim follows by induction. □

[11] R. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* **13** (1984), pp. 566–579.

2.4.1 Clique trees

Chordal graphs have a special decomposition tree, which is called a clique tree. These clique trees can be defined in two ways. We describe both.

Definition 2.33. *Let* $G = (V, E)$ *be a graph. Let* $(T, \mathcal{S})$ *be a pair where* T *is a tree and* $\mathcal{S}$ *is a collection of subsets of* V *which are in 1-1 correspondence with the vertices of* T*. For a vertex* i *in* T *Let* $S_i \in \mathcal{S}$ *be the subset that is assigned to the vertex* i*.*
The pair $(T, \mathcal{S})$ *is a clique tree for* G *if the following conditions are true.*

(a) $\mathcal{S}$ *is the set of maximal cliques in* G*, and*
(b) *for every vertex* $x \in V$*, if* x *is in two subsets* S_i *and* S_j *then* x *is in every* S_ℓ *for which* ℓ *lies on the path in* T *from* i *to* j*.*

In other words, a clique tree for G is a tree of which the vertices represent the maximal cliques in G such that, for every vertex x in G, the maximal cliques that contain x form a subtree of T.

Theorem 2.34. *A graph is chordal if and only if it has a clique tree.*

Proof. First assume that the graph $G = (V, E)$ has a clique tree $(T, \mathcal{S})$. Let ℓ be a leaf in T and let $S_\ell \in \mathcal{S}$ be the maximal clique in G which is assigned to ℓ. We claim that S_ℓ contains a vertex which is simplicial in G.

Let p be the neighbor of ℓ in T. The cliques S_ℓ and S_p are maximal cliques in G, so there must be a vertex
$$z \in S_\ell \setminus S_p.$$
The maximal cliques that contain z form a subtree of T, by definition of the clique tree. Since $z \notin S_p$, the vertex z is contained only in one maximal clique S_ℓ in G. Thus z is simplicial.

Assume that $G = (V, E)$ is a chordal graph. We may assume that G is not a clique, otherwise we are done. Let S be a minimal separator in G such that $|S|$ is as small as possible.

Let $C_1, \dots, C_t$ be the components of $G - S$. Since S is a minimal separator, there are at least two components. We claim that every vertex in S has a neighbor in each C_i. Assume that $s \in S$ has no neighbors in C_1.

Let
$$S' = S \setminus \{s\}.$$
Then C_1 is a component of $G - S'$. Thus S' is an x, y-separator for a pair
$$x \in C_1 \quad \text{and} \quad y \in \cup_{i=2}^{t} C_i.$$
This contradicts the choice of S.

Consider clique trees T_i for the subgraphs G_i of G induced by

$$C_i \cup S \quad \text{for } i \in \{1, \dots, t\}.$$

Since S is a clique, it is contained in a maximal clique M_i in G_i. We leave it as an exercise to check that $M_i \neq S$, since G is chordal.

Construct a clique tree T for G as follows. For $i = 2, \dots, t$ make the vertex in T_i that represents M_i adjacent to the vertex in T_1 that represents M_1.

We prove that this is a clique tree for G. Let $x \in C_i$. The maximal cliques that contain x are all contained in G_i. Thus these form a subtree of T_i.

Now let $x \in S$. The cliques in G_i that contain x form a subtree of T_i and M_i is one of them. By the construction, all the maximal cliques in G that contain x form a subtree of the clique tree T for G.
This proves the theorem. □

The other way to describe clique trees is as follows.

Theorem 2.35. *A graph* $G = (V, E)$ *is a chordal graph if and only if it is the intersection graph of a collection of subtrees of a tree. By that we mean that there exists a tree* T *and a collection of subtrees*

$$\{ T_x \mid x \in V \}$$

such that two vertices x *and* y *of* G *are adjacent if and only if* T_x *and* T_y *have at least one vertex of* T *in common.*

Proof. First assume that G is chordal. Consider a clique tree $(T, \mathcal{S})$ for G. Let x be a vertex of G. By definition of the clique tree, the maximal cliques in G that contain x form a subtree T_x of T.

Assume that two vertices x and y are adjacent in G. The edge $\{x, y\}$ is contained in some maximal clique in G. So $T_x \cap T_y \neq \varnothing$.

Assume that x and y are two vertices and assume that $p \in T_x \cap T_y$. The maximal clique S_p which is assigned to the vertex p in T contains x and y. Thus x and y are adjacent.

Now assume that G is the intersection graph of a collection of subtrees of a tree T. Consider a leaf ℓ of T. If ℓ is not in any subtree T_x, then we may remove the leaf from T.

If every subtree that contains ℓ contains also the neighbor of ℓ in T, then we may remove ℓ from T.

Finally, assume that there is a tree T_x which consists of the single vertex ℓ. If y and z are two neighbors of x, then T_y and T_z both contain ℓ, and so y and z are adjacent. That means that x is a simplicial vertex. Now remove x from the graph and T_x from the collection of subtrees. It follows by induction that G has a perfect elimination ordering. By Theorem 2.31, G is chordal.
This proves the theorem. □

2.4.2 Algorithms for independent set, clique and vertex coloring in chordal graphs

Theorem 2.36. *There exists a linear-time algorithm to compute a maximum independent set in chordal graphs.*

Proof. Let $G = (V, E)$ be a chordal graph. Let s be a simplicial vertex in G.

We first prove that there exists a maximum independent set I in G which contains s.

To see this, let I be a maximum independent set and assume that $s \notin I$. If

$$N(s) \cap I = \varnothing,$$

then $I \cup \{s\}$ is also an independent set, which is a contradiction.

Since s is a simplicial vertex in G, $N(s)$ induces a clique in G. Thus

$$|N(s) \cap I| = 1.$$

Let $u \in N(s) \cap I$. Then consider

$$I' = (I \setminus \{u\}) \cup \{s\}.$$

Then I' is also a maximum independent set and $s \in I'$.

The algorithm computes a maximum independent set in G as follows. Take any simplicial vertex s and start with $I = \{s\}$. Now let

$$G' = G - N[s].$$

Then G' is chordal, and by induction there exists a linear time algorithm that computes $\alpha(G')$. Then

$$\alpha(G) = 1 + \alpha(G').$$

This proves the theorem. □

Theorem 2.37. *There exists a linear-time algorithm that computes $\omega(G)$ for chordal graphs G.*

Proof. Let $G = (V, E)$ be a chordal graph. Then, by Theorem 2.31 on Page 34, G has a perfect elimination ordering

$$\pi = [x_1, \dots, x_n].$$

Let

$$N_i = |N[x_i] \cap \{x_i, \dots, x_n\}|.$$

Then

$$\omega(G) = \max \{\, N_i \mid i \in \{1, \dots, n\} \,\}.$$

This proves the theorem. □

Let $G = (V, E)$ be a chordal graph. Then G is perfect, and $\chi(G) = \omega(G)$. By Theorem 2.37 there exists a linear-time algorithm that computes $\chi(G)$. An actual coloring is also very easy to obtain, as we show next.

Theorem 2.38. *There exists a linear-time algorithm that computes a vertex coloring for* G *with* $\chi(G)$ *colors for chordal graphs* G.

Proof. Let $G = (V, E)$ be a chordal graph. Let $[x_1, \ldots, x_n]$ be a perfect elimination ordering for G. We color the vertices of G greedily with $\omega(G)$ colors from the color set

$$\Omega = \{ 1, \ldots, \omega(G) \}$$

as follows.

For $i = n$ down to 1, color the vertex x_i with an arbitrary color from Ω that is not used by vertices in

$$N(x_i) \cap \{ x_{i+1}, \ldots, x_n \}. \tag{2.7}$$

To see that this is possible, notice that the set in Formula (2.7) contains at most $\omega(G) - 1$ vertices. Thus there is a color in Ω available to color x_i.

The claim follows by induction. □

2.5 Interval graphs

As far as practical applications are concerned, the class of graphs that steals the show is the class of interval graphs. Indeed, practical applications range from archeology, sociology, scheduling classes at school, DNA-sequencing problems in biology, time-schedules for airliners, et cetera, etc, &tc.

In this section we have a short look at the definition and some of the most important properties of interval graphs.

Definition 2.39. *A graph* G *is an interval graph if it is the intersection graph of a collection of intervals on the real line. By that we mean that there is an interval* I_x *for every vertex* x *in* G *such that two vertices* x *and* y *are adjacent in* G *if and only if* $I_x \cap I_y \neq \varnothing$.

Remark 2.40. We only look at finite graphs. Then it is easy to see that we can have intervals such that no two endpoints coincide. Thus the question whether the intervals are closed or open is not an issue. Also, we may assume that all intervals are finite.

Theorem 2.41. *Interval graphs are chordal.*

Proof. It is not difficult to see that one cannot construct any cycle of length more than three as the intersection graph of intervals. If one interval I_x is completely contained in some other interval I_y then, for the corresponding vertices x and y in G we have

$$N[x] \subseteq N[y].$$

Consider a cycle $[x_1, \dots, x_k]$. Suppose it has an interval representation. By the previous observation we may assume that

$$\ell_1 < \ell_2 < r_1 < \ell_3 < r_2 < \ell_4 < r_3 < \dots,$$

where ℓ_i and r_i are the left- and right endpoint of the interval I_i that represents x_i. Now, I_k intersects I_1 and I_{k-1}, but then x_k is adjacent to all vertices in $\{x_1, \dots, x_{k-1}\}$. Thus it is a chordless cycle (that is an induced cycle) only for $k = 3$. □

Corollary 2.42. *Interval graphs are perfect.*

Let $G = (V, E)$ be a graph. An orientation directs every edge $\{x, y\}$ from x to y or from y to x. The orientation is transitive if the following holds true for every three vertices x, y and z. If there is a directed edge (x, y) from x to y and a directed edge (y, z) from y to z, then x and z are adjacent in G and the edge $\{x, z\}$ is directed from x to z.

Definition 2.43. *A graph is a comparability graph if it has a transitive orientation of its edges.*

Lemma 2.44. *If G is an interval graphs then its complement $\bar{G}$ is a comparability graph.*

Proof. Let $G = (V, E)$ be an interval graph. Consider an interval model for G. Let x and y be two nonadjacent vertices in G. Then the interval I_x lies completely to the left, or completely to the right of the interval I_y. Say we direct the edge $\{x, y\}$ of $\bar{G}$ from x to y if I_x lies to the left or I_y. We claim that this is a transitive orientation of $\bar{G}$.

We need to check that if (x, y) and (y, z) are arcs pointing from x to y and from y to z, that there is an edge $\{x, z\}$ and that it is oriented from x to z.

I'm sure you agree that this is obvious: If I_x lies left of I_y and I_y lies left of I_z then I_x lies left of I_z. □

Actually, Lemma 2.44 'almost' characterizes interval graphs. Gilmore and Hoffman proved the following characterization.[12]

[12] P. Gilmore and A. Hoffman, A characterization of comparability graphs and of interval graphs, *Canadian Journal of Mathematics* **16** (1964), pp. 539–548.

Theorem 2.45. *A graph* G *is an interval graph if and only if* G *has no induced 4-cycle and* $\bar{G}$ *is a comparability graph.*

By Theorem 2.41 every interval graph is chordal and by Theorem 2.34 on Page 35 every chordal graph has a clique tree. We show next that interval graphs have a clique tree which is a path.[13]

Theorem 2.46. *An interval graph* G *has a consecutive clique arrangement, that is, there is a linear ordering*

$$[M_1, \dots, M_t]$$

of the maximal cliques in G *such that for every vertex* x*, the maximal cliques that contain* x *form a subsequence.*

Proof. Consider an interval model for interval graph $G = (V, E)$. Scan the real line from left to right. At each point w on the real line, consider all the intervals that contain w. Let

$$M(w) = \{ x \in V \mid w \in I_x \}.$$

Then $M(w)$ is a clique in G (possibly empty).

We need to prove the 'Helly property for intervals,' that is, if M is a clique in G then there exists a $w \in \mathbb{R}$ such that each interval that represents a vertex in M contains w.

If $|M| = 3$ then this is easy to check (see Exercise 2.21).

Assume that $|M| > 3$. Let x and y be two elements of M. Replace the intervals I_x and I_y by $I_x \cap I_y$. Each pair of intervals of this new collection intersects, by the previous observation. By induction, there exists a $w \in \mathbb{R}$ which is contained in every interval of this collection. Then w is also contained in every interval that represents a vertex of M.

We now have that every maximal clique is in the set

$$\{ M(w) \mid w \in \mathbb{R} \}.$$

This defines the linear order of the maximal cliques in G: For each maximal clique M fix a real number w such that w is in each interval which represents a vertex of M. By the finiteness of the system, we can choose the real numbers such that no two maximal cliques are represented by the same real number. If $M(w_1)$ and $M(w_2)$ are two maximal cliques, then $M(w_1)$ precedes $M(w_2)$ if and only if $w_1 < w_2$. Obviously, since I_x is an interval, for every vertex x the maximal cliques that contain x are consecutive in this ordering. □

[13] R. Halin, Some remarks on interval graphs, *Combinatorica* **2** (1982), pp. 297–304.

Definition 2.47. *An asteroidal triple $\{x, y, z\}$, AT for short, in a graph G is a set of three pairwise nonadjacent vertices in G such that for every pair of them there is a path connecting them that avoids the neighborhood of the third.*

Remark 2.48. Notice that, if some x, y-path contains no neighbor of z then it also does not contain z, since z is not adjacent to x nor y.

For example, consider a 6-cycle. Let x, y and z be three vertices that are pairwise not adjacent. Then $\{x, y, z\}$ is an asteroidal triple.

Lemma 2.49. *Interval graphs are AT-free, that is, they have no asteroidal triple.*

Proof. Let $G = (V, E)$ be an interval graph. Consider an interval model for G. take three intervals I_x, I_y and I_z which pairwise do not intersect. Then one of the intervals is between the other two. Say I_x is left of I_y and I_y is left of I_z.

Let $P = [x = x_1, \dots, x_k = z]$ be an x, z-path. Assume that $y \notin P$. Then an interval I_i of some vertex x_i must intersect I_y. Thus $\{x, y, z\}$ is not an asteroidal triple. □

Theorem 2.50. *Let G be a graph. The following statements are equivalent.*

(a) G *is an interval graph.*
(b) G *is AT-free and chordal.*
(c) *For every three maximal cliques M_1, M_2 and M_3 in G there is one that separates the others. Here we say that M_1 separates M_2 and M_3 if $M_2 \setminus M_1$ and $M_3 \setminus M_1$ are contained in different components of $G - M_1$.*

2.6 Permutation graphs

A permutation diagram is obtained as follows. Let L_1 and L_2 be two horizontal lines in the plane, one above the other. Label n points on the topline and on the bottom line by $1, 2, \dots, n$. Connect each point on the topline by a straight linesegment with the point with the identical label on the bottom line. A graph G is a permutation graph if it is the intersection graph of the linesegments of a permutation diagram.[14] By that we mean that the vertices of the graph G are the linesegments of the permutation diagram and two vertices in G are adjacent if the two linesegments cross each other.

If G is a permutation graph then its complement $\bar{G}$ is also a permutation graph. This is easy to see; simply reverse the ordering of the points on one of the two horizontal lines.

[14] A. Pnueli, A. Lempel and S. Even, Transitive orientation of graphs and identification of permutation graphs, *Canadian Journal of Mathematics* **23** (1971), pp. 160–175.

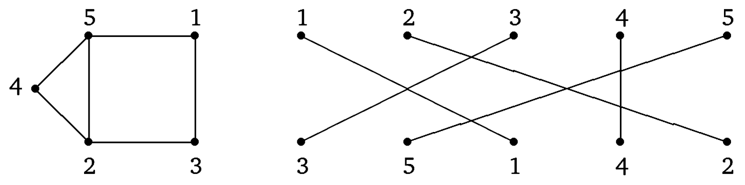

Fig. 2.4. A permutation graph and its permutation diagram

Lemma 2.51. *Permutation graphs are perfect.*

Proof. It is easy to check that one cannot construct a permutation diagram for a graph which is a cycle of length more than four. Thus permutation graphs have no holes.

A permutation graph G can also not have an antihole, since $\bar{G}$ is also a permutation graph. □

Recall Definition 2.43. A comparability graph is a graph which has a transitive orientation; so we can direct any edge $\{x, y\}$ either from x to y or from y to x such that the directed graph is transitive.

Theorem 2.52. *A graph G is a permutation graph if and only if G and $\bar{G}$ are comparability graphs.*

Proof. Let G be a permutation graph. Consider a permutation diagram for G. Notice this: for any independent set in G the corresponding linesegments are noncrossing. So they can be ordered left to right, which is of course a transitive ordering. By that we mean that if a linesegment x is left of a linesegment y and the linesegment of y is to the left of a linesegment z, then the linesegment of x is left of the linesegment z.

This shows that $\bar{G}$ is a comparability graph. Then G is also a comparability graph since the class of permutation graphs is closed under complementations.

Let G be a graph such that G and $\bar{G}$ are both comparability graphs. We show that G is a permutation graph by constructing a permutation diagram.

Let F_1 and F_2 be transitive orientations of G and $\bar{G}$. We claim that $F_1 + F_2$ is an acyclic orientation of the complete graph. Otherwise there is a directed triangle, and so two edges in the triangle are directed according to one of F_1

and F_2 and the third is directed according to the other one of F_1 and F_2. But this contradicts the transitivity of F_1 or the transitivity of F_2.

Likewise, $F_1^{-1} + F_2$ is acyclic. Order the vertices on the topline according to $F_1 + F_2$ and the vertices on the bottom line according to $F_1^{-1} + F_2$. It is easy to check that this yields the permutation diagram. □

The following characterization of permutation graphs illustrates the relation of this class of graphs to the class of interval graphs. Consider a collection of intervals on the real line. Construct a graph of which the vertices are the intervals and make two vertices adjacent if one of the two intervals contains the other. Such a graph is called an interval containment graph.

Theorem 2.53. *A graph is a permutation graph if and only if it is an interval containment graph.*

Proof. Consider a diagram of a permutation graph. When one moves the bottom line to the right of the topline then the linesegments in the diagram transform into intervals. It is easy to check that two linesegments intersect if and only if one of the intervals is contained in the other one. This proves that permutation graphs are interval containment graphs.

Now consider an interval containment graph. Construct a permutation diagram as follows. Put the left endpoints of the intervals in order on the topline and the right endpoints in order on the bottom line of the diagram. Then one interval is contained in another interval if and only if the two linesegments intersect. This proves that every interval containment graph is a permutation graph. □

Tedder, *et al.* proved the following theorem.[15]

Theorem 2.54. *Permutation graphs can be recognized in linear time. If* G *is a permutation graph then this algorithm can be used to construct a permutation diagram in linear time.*

2.6.1 Cliques and independent sets in permutation graphs

Let G be a permutation graph. Consider a permutation diagram for G. By Theorem 2.54 this diagram can be computed in linear time.

Let the vertices on the topline be numbered from left to right as $1, \dots, n$. Let

[15] M. Tedder, D. Corneil, M. Habib and C. Paul, Simpler linear-time modular decomposition via recursive factorizing permutations, *Proceedings ICALP'08*, Springer, LNCS 5125 (2008), pp. 634–645.

$$\pi = [\pi_1, \pi_2, \ldots, \pi_n]$$

be the sequence of numbers as they appear in left to right order on the bottom line.

Two vertices i and j with $i < j$ are adjacent if i appears to the right of j on the bottom line. Thus a clique in G corresponds with a decreasing sequence in π.

This shows that finding $\omega(G)$ is equivalent to finding the longest decreasing subsequence in π. Given the sequence π this can be done very efficiently.[16]

Let $G = (V, E)$ be a permutation graphs. Then $\bar{G}$ is also a permutation graph and $\alpha(G) = \omega(\bar{G})$.

Theorem 2.55. *There exist* $O(n \log \log n)$ *algorithms to compute* $\alpha(G)$ *and* $\omega(G)$ *for permutation graphs* G. *Here we assume that the permutation* π *is a part of the input.*

2.7 Problems

2.1. Let G be a 5-cycle. Is G perfect?

2.2. A graph G is bipartite if $\chi(G) \leqslant 2$. Show that G is perfect.

2.3. Prove that a graph G is bipartite if and only if all cycles in G are even. Is it also true that G is bipartite if and only if all *induced* cycles are even?

2.4. Let $G = (V, E)$ be a graph. A vertex cover in G is a set C of vertices such that every edge in G has at least one endpoint in C. Let $\tau(G)$ be the cardinality of a smallest vertex cover in G. Prove that

$$\alpha(G) + \tau(G) = n \quad \text{where } n = |V|. \tag{2.8}$$

2.5. A P_3 is a path with three vertices. Let G be a graph without induced P_3. Show that G is the union of cliques.

2.6. Show that a graph G is a cograph if and only if it has a decomposition tree.

2.7. Let $G = (V, E)$ be a graph. Two vertices x and y are twins if either

$$N[x] = N[y] \quad \text{or} \quad N(x) = N(y).$$

[16] J. Hunt and T. Szymanski, A fast algorithm for computing longest common subsequences, *Communications of the ACM* **20** (1977), pp. 350–353.

(a) Show that every cograph with at least two vertices has a twin.
(b) Show also the converse: If every induced subgraph of a graph G has either only one vertex, or else has a twin, then G is a cograph.

`Hint`: Consider a cotree (T, f) of G. Let x and y be two vertices in G that are mapped by f to two leaves in T that have the same parent in T. Prove that x and y are twins in G.

2.8. In Section 2.2.2 on Page 25 we showed how to compute the clique number in cographs in linear time. Devise linear-time algorithm to compute $\alpha(G)$, $\chi(G)$ and $\kappa(G)$.
`Hint`: Use the fact that G is perfect and that $\bar{G}$ is a cograph.

2.9. Check that the house, hole, domino and gem depicted in Figure 2.2 on Page 26 are not distance hereditary.

2.10. Let $G = (V, E)$ be distance hereditary and let G' be the graph obtained from G by adding one vertex x to G which is either isolated, or a pendant vertex with a neighbor $y \in V$, or a twin of some vertex $z \in V$. Show that G' is distance hereditary.

2.11. Finish the proof of Theorem 2.20 on Page 30 by showing how the four values, that are mentioned in the proof, are computed for an edge e in the decomposition tree T.

2.12. Which of the graphs in Figure 2.2 on Page 26 is chordal?

2.13. Finish the proof of Theorem 2.27 by showing that the paths P_x and P_y exist.
`Hint`: There is an a, b-path P_x with internal vertices in C_x since a and b both have a neighbor in C_x and $G[C_x]$ is connected. You need to show how to get rid of all the shortcuts.

2.14. Let G be a chordal graph and let S be a minimal x, y-separator for nonadjacent vertices x and y. Let C_x be the component of $G - S$ that contains the vertex x. Then C_x contains a vertex x' such that

$$S \subseteq N(x').$$

2.15. Let $G = (V, E)$ be a chordal graph and let $S \subset V$ be a separator in G. Show that S is a minimal separator if and only if there exist two components C_1 and C_2 in $G - S$ such that every vertex of S has at least one neighbor in C_1 and in C_2, that is,

$$\forall_{s \in S} \; N(s) \cap C_1 \neq \varnothing \quad \text{and} \quad N(s) \cap C_2 \neq \varnothing.$$

2.16. Consider a tree T. If T is not a clique then T has two simplicial vertices which are not adjacent. Prove the analogue for chordal graph: If G is a chordal graph and if G is not a clique, then G has two nonadjacent simplicial vertices.

2.17. Design an algorithm to compute a perfect elimination ordering of a chordal graph in linear time.
`Hint`:Number the vertices from n to 1 in decreasing order. For the next vertex to number, choose one that has the largest number of neighbors that are already numbered, braking ties arbitrarily.
This exercise is not easy; perhaps you like to have a look at Tarjan and Yannakakis' paper.

2.18. The proof of Theorem 2.37 is perhaps a bit sketchy. Rub the sleep out of your eyes and make sure that you agree with all the details.

2.19. Show that the coloring algorithm of Theorem 2.38 on Page 38 can be implemented to run in linear time.

2.20. Design a polynomial-time algorithm which check if a graph has an asteroidal triple.
`Hint`: Try all triples $\{x, y, z\}$ of pairwise nonadjacent vertices in a graph G. Notice that there is an x, z-path that avoids $N(y)$ if and only if x and z are both contained in a component of $G - N(y)$. What is the timebound of your algorithm?

2.21. Consider three intervals I_1, I_2 and I_3 on the real line. If any two have a nonempty intersection then

$$I_1 \cap I_2 \cap I_3 \neq \varnothing.$$

2.22. Prove that the Helly property also holds for a collection of subtrees of a tree. That is the following. Let T be a tree, and let $\mathcal{S}$ be a collection of subtrees of T such that

$$\forall_{T_1 \mathcal{S}} \; \forall_{T_2 \in \mathcal{S}} \; T_1 \text{ and } T_2 \text{ have at least one vertex of } T \text{ in common.}$$

Then there is a vertex of T which is a vertex of every subtree of $\mathcal{S}$.

2.23. Design an efficient algorithm to compute a feedback vertex set on interval graphs. What is the timebound for your algorithm? Extend the algorithm such that it works on chordal graphs.
`Hint`: Use the clique tree. How many vertices can a clique have that are not in the feedback vertex set?

2.24. Let $G = (V, E)$ be a chordal graph. By Theorem 2.35 there exist a tree T and a collection of subtrees

$$\mathcal{T} = \{ T_x \mid x \in V \}$$

such that and two vertices x and y in G are adjacent if and only if $T_x \cap T_y \neq \varnothing$. We say that G is the intersection graph of $\mathcal{T}$.

Make a subtree T_e for every edge $e = \{a, b\} \in E$ in G by defining

$$T_e = T_a \cup T_b.$$

Consider the intersection graph G^* of the subtrees

$$\mathcal{E} = \{ T_e \mid e \in E \}.$$

Prove that G^* is the square of the linegraph $L(G)$ of G. By that we mean that the vertices of G^* are the edges of G, and two vertices e_1 and e_2 of G^* are adjacent if and only if

$$\exists_{e \in E} \; e_1 \cap e \neq \varnothing \quad \text{and} \quad e_2 \cap e \neq \varnothing.$$

This proves that, if G is chordal then G^* is also chordal.

2.25. Prove that every cograph G is a permutation graph by showing that there exists a permutation diagram for G.

2.26. Are permutation graphs AT-free?

2.27. Consider a circle C drawn in the plane. A chord of C is a straight line-segment that connects two points on C. A circle graph G is the intersection graph of a collection of chords in a circle. By that we mean that every vertex of G is represented by a chord and that two vertices are adjacent in G if and only if the two chords intersect. The diagram that represents G is called a circle diagram.

(i) Show that every permutation graph is a circle graph.
(ii) Prove that every distance-hereditary graph is a circle graph.
`Hint:` Use the fact that a graph is distance-hereditary if and only if every induced subgraph has an isolated vertex, or a pendant vertex, or a twin. Prove by induction that a distance-hereditary graph has a circle diagram.

3

Fixed-parameter Algorithms

For most NP-complete problems one can attach a parameter to the problem and consider the parameterized version of the problem.

For example, consider the clique number $\omega(G)$ of a graph G. The clique number problem asks to find the maximal value ω for which the graph G has a clique with ω vertices.

Suppose that, instead, we ask the following.

Given a graph G and some number k.
Is $\omega(G) \geqslant k$?

Any algorithm, exponential or not, that solves this parameterized clique number problem, can be used to solve the clique number problem: Just check for $k = 1, \ldots, n$ whether G has a clique of k vertices, and determine the largest k for which it has.

Just coming up with some parameter like that doesn't help much, of course. Well, maybe it does, a little bit. Suppose that $\omega(G)$ is at most two. For example, any bipartite graph has clique number at most two, so there are lots of them.

It is easy to check if $\omega(G) \geqslant 3$: just run the following algorithm.

1. If G is an independent set then $\omega(G) = 1$. Of course, it takes only linear time to check if G has an edge or not.
2. When G has at least one edge, then the next step is to check if G has a triangle $\{x, y, z\}$. If so, then $\omega(G)$ is at least three. If not, then $\omega(G) < 3$.

   To check if G has a triangle, we can check for every three vertices x, y and z if $\{x, y, z\}$ is a triangle or not. There are $O(n^3)$ triples to check. If we use the adjacency matrix to represent G we can check if a given triple $\{x, y, z\}$

is a triangle or not in constant time, since we only need to check if the three pairs $\{x, y\}$, $\{x, z\}$ and $\{y, z\}$ are three edges or not. So we can check in $O(n^3)$ time if $\omega(G) \geqslant 3$ or not.

Of course, we can generalize this. We can check in $O(k^2 \cdot n^k)$ time if $\omega(G) \geqslant k$ or not. Namely, to check if G has a clique with k vertices we can try all subsets with k vertices and see if one of them is a clique. In a graph G with n vertices there are $O(n^k)$ subsets with k vertices. Let $\Omega = \{x_1, \dots, x_k\}$ be such a subset. To check if Ω is a clique, check if every pair in Ω is adjacent. There are $O(k^2)$ pairs, so, when we use the adjacency matrix to check adjacencies, this algorithm can be implemented to run in $O(k^2 \cdot n^k)$ time.

The above algorithm shows that for each CONSTANT k, the algorithm to check if $\omega(G) \geqslant k$ runs in polynomial time. So we have a polynomial algorithm to check if a graph G has a clique with three vertices, or a clique with 10 vertices, or even if G has a clique with one million vertices. In fact, for each CONSTANT k, the algorithm to check if $\omega(G) \geqslant k$ runs in polynomial time.

Of course, we can say that the algorithm is polynomial *only* for constant k. An algorithm that runs in $O(n^n)$, or an algorithm that runs in $O(n^{\sqrt{n}})$ is not polynomial. Even an algorithm that runs in $O(n^{\log\log(n)})$ is not polynomial.

The inverse Ackermann function $\alpha(n)$ is one of the slowest growing functions that exist. For example, when n is the number of atoms in the universe, then $\alpha(n) < 5$ (at least, that's what people think.) No computer can ever be built that contains more components than the number of atoms in the universe. An algorithm that runs in $O(n^{\alpha(n)})$ is not polynomial, since $\alpha(n)$ is a growing function of n. For any practical situation this algorithm runs in $O(n^5)$, but it is not polynomial. That's where the 'theoretical' in 'theoretical computer science' kicks in; it takes us where no man has gone before.

When we really want to run that algorithm to check if $\omega(G) \geqslant 10^6$ then we run into trouble. I mean, a polynomial-time algorithm that runs in $O(n^{10^6})$ is not very appetizing! So, "Big Deal!" you say, and you're right, so far it is not a big deal.

Here's a definition.

Definition 3.1. *A parameterized problem* (P, k) *is fixed-parameter tractable if there exist*

1. *some function* $f : \mathbb{N} \to \mathbb{R}$*, and*
2. *some constant* c*, and*
3. *some algorithm that solves* (P, k) *and that runs in* $O(f(k) \cdot n^c)$ *time.*

For example, when

$$f(k) = 2^k \quad \text{and} \quad c = 2$$

a fixed-parameter algorithm to solve the parameterized problem (P, k) runs in $O(2^k \cdot n^2)$.

Notice the difference with an algorithm that runs in $O(k^2 \cdot n^k)$. In a fixed-parameter algorithm the parameter k does not appear in the exponent of n. An algorithm that runs in $O(k^2 \cdot n^k)$ is not a fixed-parameter algorithm because the k appears in the exponent of n.

Which of the two timebounds, $O(k^2 \cdot n^k)$ or $O(2^k \cdot n^2)$ would you prefer? For constant k, say $k = 10^6$, both algorithms run in polynomial time. Also, in both cases, the algorithms are practical only for small values of k. When $k = \sqrt{n}$, the algorithms are not polynomial.

To compare them, let's fix $k = 10$. The fixed parameter algorithm runs in time $O(2^{10} \cdot n^2)$, so actually it is a quadratic algorithm. The other algorithm runs in $O(100 \cdot n^{10})$. I think everybody agrees, the fixed-parameter algorithm is better, unless we are fooled, for example by some crazy constant that is hidden in the big O.

An example of a parameterized problem (P, k) is $(\omega(G), k)$ and it asks if $\omega(G) \geqslant k$. Or, another example of a parameterized problem is whether G has a dominating set with *at most* k vertices, that is, $(\gamma(G), k)$ is the problem that asks if $\gamma(G) \leqslant k$. One more example is the problem (χ, k) which asks if $\chi(G) \leqslant k$.

Notice that you have to be a bit careful with the sign; sometimes you want to maximize the cardinality of some subset and sometimes you want to minimize it. The question whether there is a vertex coloring of G with *at least* k colors makes little sense. As long as you have enough vertices you can give each vertex a different color; a graph with n vertices has an n-coloring.

In this chapter we look at some problems for which there are fixed-parameter algorithms. Not every problem is like that. For example, it is unlikely that the parameterized clique number problem $(\omega(G), k)$ is fixed-parameter tractable. It can be shown that,

unless $P = NP$, there is no fixed-parameter algorithm that solves the parameterized clique number problem $(\omega(G), k)$ on graphs.

Unless P=NP, any algorithm that solves $(\omega(G), k)$ will have a running time where the k appears in some way in the exponent of n.

The book by Downey and Fellows examines which problems have a fixed-parameter algorithm.[1] The book contains a list of problems that are fixed-parameter tractable, and a list of problems that are not fixed-parameter tractable unless P=NP.

It is not always easy to tell. As in the usual NP-completeness theory one can show that some parameterized problem (P, k) is not fixed-parameter tractable by reducing it to some other parameterized problem (Q, k').

Suppose it is known that some parameterized problem (Q, k') has no fixed-parameter solution. To reduce a problem (P, k) to (Q, k') one first reduces P to Q in polynomial time. Furthermore, the reduction has to be so that there is some function that connects the parameters k and k', say $k = g(k')$. Just as in ordinary NP-completeness proofs, we now need that $(P, g(k'))$ has a solution if and only if (Q, k') has a solution.

Now suppose we have an algorithm that solves (P, k) in time $O(f(k) \cdot n^c)$ for some function f and some constant c. Then a parameterized reduction as above solves (Q, k') in time $O(f(g(k')) \cdot h(n)^c)$ where h is the polynomial that reduces P to Q. Since we know that (Q, k') has no fixed-parameter solution unless P=NP, any fixed-parameter solution for (P, k) implies that P=NP.

Notice that the difference with the ordinary NP-completeness reductions is the function that relates the parameters. People who are familiar with NP-completeness proofs will have little trouble doing similar proofs for parameterized problems.

We show a very easy example of a parameterized reduction at the start of the next section.

3.1 Vertex cover

Definition 3.2. *Let* $G = (V, E)$ *be a graph. A vertex cover for* G *is a set* S *of vertices such that every edge in* G *has at least one endvertex in* S.

The vertex cover problem asks for a vertex cover of smallest cardinality.

The vertex cover problem is of course equivalent to the independent set problem.

Lemma 3.3. *Let* $G = (V, E)$ *be a graph. Then* S *is a vertex cover if and only if* $V \setminus S$ *is an independent set.*

[1] R. Downey and M. Fellows, *Parameterized complexity*, Springer-Verlag, 1999.

Proof. Let S be a vertex cover. By definition, there is no edge with both ends in $V \setminus S$. So $V \setminus S$ is an independent set.

Suppose I is an independent set. Then every edge must have at least one end in $V \setminus I$. So $V \setminus I$ is a vertex cover. □

We know that the independent set problem is NP-complete, so Lemma 3.3 shows that the vertex cover problem is NP-complete as well.

As we mentioned in the introduction, there is probably no fixed-parameter algorithm that solves the parameterized clique number problem $(\omega(G), k)$.

There is a trivial parameterized reduction from $(\alpha(G), k)$ to $(\omega(G), k')$. The reduction takes the complement $\bar{G}$ of the graph G. The function that relates k and k' is just the identity function $g(k) = k$. Notice that G has a clique with at least k vertices if and only if $\bar{G}$ has an independent set with k vertices.

Since the reduction from G to $\bar{G}$ is polynomial, this shows that $(\alpha(G), k)$ is not fixed-parameter tractable, unless something funny happens.

Let's introduce some fancy notation for the vertex cover number of a graph. Denote the smallest cardinality of a vertex cover in a graph G by $\zeta(G)$. (That greek letter is "zeta," or "z," the last letter in the English alphabet. The greek α is "a," the first letter in the English alphabet.) Then, by Lemma 3.3,

$$\zeta(G) = n - \alpha(G), \quad \text{where } n \text{ is the number of vertices in } G.$$

(At least when $n = 27$ is looks OK in the English alphabet.)

Here comes the surprise.

Theorem 3.4. *The vertex cover problem is fixed-parameter tractable. The problem* $(\zeta(G), k)$ *can be solved in* $O(2^k \cdot n^2)$ *time.*

Proof. Let $G = (V, E)$ be a graph. Let $e = \{x, y\}$ be an edge in G. If Z is a vertex cover then

$$x \in Z \quad \text{or} \quad y \in Z.$$

Consider the following algorithm.

Start with $Z = \varnothing$. Build a binary tree as follows. The root consists of the pair $(G, \varnothing)$. If $G - Z$ has no edge, then we are done; Z is a vertex cover.

Now assume that $\{x, y\}$ is an edge with both ends in $G - Z$. Then construct two children in the tree. One for the pair

$$(\ G - (Z \cup \{x\}),\ Z \cup \{x\}\ )$$

and one for the pair

$$(\ G - (Z \cup \{y\}),\ Z \cup \{y\}\ ).$$

Thus in one case we put x in the vertex cover and in the other case we put y in the vertex cover. In both cases we remove the vertex from the graph that we put in the vertex cover.

Of course, we can keep growing the binary tree until each leaf is some pair $(G - Z^*, Z^*)$, where Z^* is a minimal vertex cover. When we scan all the leaves and find the smallest cardinality of the sets Z^* then we find $\zeta(G)$.

Here's the trick. We want to find a vertex cover Z of size at most k. Thus, if some node in the binary tree has a pair $(G - Z', Z')$ with

$$|Z'| = k$$

then we don't need to grow it any further. We let the algorithm check if $G - Z'$ is an independent set or not. If so, then Z' is a vertex cover with k vertices. Otherwise, we let the algorithm backtrack to its parent, and continue the search from there.

In this way, the binary tree has a depth at most k, since each branching adds one vertex to the set Z. Any full binary tree of depth k has 2^k leaves and $2^k - 1$ internal nodes.

In each vertex $(G - Z^*, Z^*)$ of the binary tree we have to check if there is an edge $\{x, y\}$ with both ends in $G - Z^*$. This we can do in $O(n^2)$ time. The total time complexity is therefore $O(2^k \cdot n^2)$. □

Remark 3.5. We have seen that $(\alpha(G), k)$ is not fixed-parameter tractable and that $(\zeta(G), k')$ is. Furthermore, we have that

$$\zeta(G) = n - \alpha(G)$$

for any graph G.

Obviously, we cannot have a parameterized reduction from $(\zeta(G), k')$ to $(\alpha(G), k)$. What is the problem?

The problem is that we need to relate the parameters in the two problems if we want to have a parameterized reduction. The parameters should relate like $k' = n - k$. There is no function doing that; there would be an "n" somewhere in that function.

Remark 3.6. The technique used in Theorem 3.4 is called a bounded search-tree technique. In this technique you build a search-tree of a size which is some function of the parameter k. The bounded search-tree technique is one of the most successful techniques that one can use to prove that some problem is fixed-parameter tractable.

3.2 A kernel for vertex cover

In this section we describe another technique which is very often useful to obtain fixed-parameter algorithms. We illustrate it for the vertex cover problem.

Lemma 3.7. *Let* G *be a graph and assume that* $\zeta(G) \leqslant k$. *Let* Z *be a vertex cover for* G *with* $|Z| \leqslant k$. *If a vertex* x *in* G *has at least* $k+1$ *neighbors then* $x \in Z$.

Proof. Suppose that $x \notin Z$. Consider $k+1$ edges

$$\{x, y_1\}, \; \dots \; , \{x, y_{k+1}\}, \quad \text{where} \quad \{y_1, \dots, y_{k+1}\} \subseteq N(x).$$

Every edge must have one endvertex in Z, thus if $x \notin Z$ then $y_i \in Z$, for all $i \in \{1, \dots, k\}$. But then $|Z| > k$, which is a contradiction. □

Lemma 3.8. *Let* G *be a graph and assume that every vertex in* G *has degree at most* k. *Furthermore, assume that* G *has no isolated vertices. Assume that* G *has a vertex cover* Z *with at most* k *vertices. Let* n *and* m *be the number of vertices and edges in* G. *Then*

$$n \leqslant k + k^2 \quad \textit{and} \quad m \leqslant 2k^2.$$

Proof. Let Z be a vertex cover in G. Since G has no isolated vertices, and since Z is a vertex cover, every vertex of $V \setminus Z$ has at least one neighbor in Z.

Since the degree of any vertex in G is at most k, any vertex in Z has at most k neighbors in $V \setminus Z$. This proves that the number of vertices is at most

$$n \leqslant |Z| + |Z| \cdot k = |Z|(k+1) \leqslant k(k+1)$$

Any edge has either two ends in Z or exactly one end in Z. Thus the number of edges in G satisfies

$$m \leqslant |Z|^2 + |Z| \cdot k \leqslant 2k^2.$$

This proves the lemma. □

Lemmas 3.7 and 3.8 can be used to reduce the graph G in polynomial time to a graph H of which the number of vertices is bounded by some function of the parameter k. Such a graph H is called a kernel for the parameterized problem. Notice that the exponential part of the algorithm runs only on the kernel.

Theorem 3.9. *Let* G *be a graph with* n *vertices. The parameterized vertex cover problem* $(\zeta(G), k)$ *is fixed-parameter tractable. There exists an algorithm for* $(\zeta(G), k)$ *which runs in* $O(n^2 + 2^k \cdot k^4)$ *time.*

Proof. Reduce G to a kernel H by removing vertices that have at least $k+1$ neighbors in G. Let Z_0 be this set of vertices. Then $Z_0 \subseteq Z$ for any vertex cover Z with at most k vertices.

If some vertices of the remaining graph $G - Z_0$ are isolated, then remove them. The subgraph H induced by the remaining vertices of $G - Z_0$ is called the kernel. The part of the algorithm described above is called the reduction to the kernel, and it runs in $O(n^2)$ time.

By Lemma 3.8 we may assume that the number of vertices in H is at most $k^2 + k$. We need to find a vertex cover in H with at most $k - |Z_0|$ vertices. By Theorem 3.4 there is an $O(2^k \cdot (k^2 + k)^2) = O(2^k \cdot k^4)$ algorithm that solves the parameterized vertex cover problem in the kernel.

In total, our algorithm runs in $O(n^2 + 2^k \cdot k^4)$ time. To see that this is a fixed-parameter algorithm, we need to show that it has the form $f(k)n^c)$ for some function f and constant c. This is obvious; namely, since $k \leqslant n$ we have that

$$n^2 + 2^k \cdot k^4 \leqslant (2^k + 1) \cdot n^4.$$

This proves the theorem. □

The following theorem is pretty useless in practice, but it may answer a question that you had in mind.

Theorem 3.10. *Any parameterized problem that is fixed-parameter tractable can be reduced to a kernel in polynomial time.*

Proof. Assume that there exists an algorithm that runs in $O(f(k) \cdot n^c)$ for some function f and some constant c. We use this algorithm to reduce the input to a kernel as follows.

1. When $f(k) \leqslant n$ then the algorithm above runs in time proportional to $f(k)n^c \leqslant n^{c+1}$, *i.e.*, it is polynomial. This part of the algorithm is the reduction to the kernel.
2. Otherwise, when $f(k) > n$, then we have a kernel of size $n < f(k)$.

This proves the theorem. □

An alternative algorithm for finding a kernel for the parameterized vertex cover problem uses a maximum matching.

Definition 3.11. *A matching in a graph* $G = (V, E)$ *is a subset* M *of edges such that no two edges in* M *have an endvertex in common.*

The maximum matching problem asks for a matching in a graph G of maximal cardinality. The maximum matching problem can be solved in $O(n^{5/2})$ on graphs G with n vertices.[2]

Lemma 3.12. *Let* $G = (V, E)$ *be a graph. Assume that* $\zeta(G) \leqslant k$. *Then*

$$\nu(G) \leqslant k,$$

where $\nu(G)$ *is the cardinality of a maximum matching in* G.

Proof. If Z is a vertex cover in G then Z contains at least one endvertex of each edge in a maximum matching M. □

Theorem 3.13. *Let* $G = (V, E)$ *be a graph and assume that* G *has no isolated vertices. There exists a kernel with at most* $3k$ *vertices for* $(\zeta(G), k)$.

Proof. The algorithm first computes a maximum matching M. By Lemma 3.12 if $|M| > k$ then any vertex cover has at least $k + 1$ vertices.

Now assume that $|M| \leqslant k$. Let $V(M)$ be the set of endvertices of edges in M and let $I = V \setminus V(M)$. Then I is an independent set.

Consider the graph B with vertex set V and edge set the edges with one endpoint in $V(M)$ and the other endpoint in I. Then B is bipartite. By the König-Egerváry theorem

$$\zeta(B) = \nu(B).$$

By the maximum matching algorithm, we can find in polynomial time a maximum matching M' in B and a minimum vertex cover Z' for B with endvertices in $V(M')$.

We now consider two cases.
First assume that Z' has at least one vertex in $V(M)$. Define

$$U = V(M) \cap Z'.$$

Define I' as the set of other endvertices of edges in M' that have one endvertex in U.

We claim that there exists a minimum vertex cover Z in G which contains U. To see this, notice that each edge $e \in M'$ must have an endvertex in Z. If this endvertex is in I', then we may replace it with the other endvertex of e in U. Then the new set of vertices is also a vertex cover.

So, in this case we can reduce the graph by removing the vertices of

[2] J. Edmonds, Paths, trees, and flowers, *Canadian Journal of Mathematics* **17** (1965), pp. 449–467.

$$U \cup I'$$

and put the vertices of U in Z. The algorithm proceeds to look for a vertex cover with at most $k - |U|$ vertices in

$$G - (U \cup I').$$

In the second case we assume that Z' has no vertices in $V(M)$. Then the minimum vertex cover Z' has only vertices in I. Since there are no isolated vertices,

$$Z' = I.$$

Then the number of vertices is at most

$$|I| + |V(M)| \leqslant k + 2k = 3k.$$

In other words, this set is a kernel. □

3.3 A better search-tree algorithm for vertex cover

In this section we develop a better search-tree algorithm for the parameterized vertex cover problem. The improvement follows from two simple observations.

Lemma 3.14. *Let $G = (V, E)$ be a graph. Let x be a vertex of degree at most one. Then there exists a minimum vertex cover Z for G with $x \notin Z$.*

Proof. Assume that x is isolated. Then x is not in any minimum vertex cover.

Assume that x has one neighbor, say y. If Z is a minimum vertex cover, and $x \in Z$, then

$$Z' = (Z \setminus \{x\}) \cup \{y\}$$

is also a minimum vertex cover.
This proves the lemma. □

Lemma 3.15. *Let G be a graph and let x be a vertex with two neighbors, say y and z. Then there exists a minimum vertex cover Z for G such that either*

(i) $\{y, z\} \subseteq Z$ *and* $x \notin Z$, *or*
(ii) $x \in Z$ *and*

$$(N(y) \cup N(z)) \setminus N(x) \subseteq Z.$$

Proof. Let $G = (V, E)$ be a graph and let Z be a minimum vertex cover for G. Let x be a vertex with two neighbors, say y and z.

Obviously, when $y \in Z$ and $z \in Z$ then $x \notin Z$, since Z is minimum.

Assume that $x \in Z$, $y \in Z$ and $z \notin Z$. Then

$$Z' = (Z \setminus \{x\}) \cup \{z\}$$

is a minimum vertex cover and $|Z'| = |Z|$.

Thus, we may assume that either both y and z are in Z or neither of them is in Z. If neither y nor z is in Z, then $x \in Z$ and

$$(N(y) \cup N(z)) \setminus N(x) \subseteq Z.$$

This proves the lemma. □

Theorem 3.16. *There exists an* $O(1.47^k \cdot n^2)$ *algorithm that solves the parameterized vertex cover problem* $(\zeta(G), k)$.

Proof. The algorithm builds a search-tree as follows. The root of the search-tree is the pair $(G, \varnothing)$.

At each vertex (G', Z') in the search tree the algorithm proceeds as follows.

(1) If there is an isolated vertex x in G', then remove it from G'.
(2) If G' has a vertex x with one neighbors y, then put y in Z' and remove x and y from the graph G'.
(3) If G' has a vertex x with at least three neighbors, the the algorithm branches; the vertex in the search tree gets two children. In one child the vertex x is put in Z' and it is removed from G'. In the other child, all the neighbors of x are put in Z' and all vertices of $N[x]$ are removed from the graph G'.
(4) Assume that G' has a vertex x with two children, y and z. Assume that neither y nor z has a neighbor which is not a neighbor of x. If y and z are not adjacent then put x in Z' and remove x, y and z from G'. If y and z are adjacent the put two of x, y and z in Z' and remove x, y and z from G'.
Otherwise, when at least one of y and z has a neighbor which is not a neighbor of x, the algorithm branches. In one child both y and z are put in Z' and they are removed from G'. In the other child, x and all vertices of
$$(N(y) \cup N(z)) \setminus N(x)$$
are put in Z' and they are removed from G'. Notice that the correctness of this step follows from Lemma 3.15.

Notice that each vertex in the search tree is a leaf or it has two children.

Let $T(k)$ be the amount of work done by the algorithm. In the first and second case, the algorithm is greedy. We may assume that these cases do not occur. In the third case, the branching gives the recurrence

$$T(k) \leqslant T(k-1) + T(k-3) + O(n^2). \tag{3.1}$$

In the fourth case, we have

$$T(k) \leqslant 2T(k-2) + O(n^2), \tag{3.2}$$

since in each of the two branches at least two vertices are put in the vertex cover.

The solution for the Recurrence (3.1) is $T(k) = O(1.47^k \cdot n^2)$. The solution for the Recurrence (3.2) is $T(k) = O(1.42^k \cdot n^2)$. (In Exercise 3.5 we ask you to check that.) Clearly, the worst case occurs when the algorithm branches all the time as in the third case. □

Remark 3.17. The best algorithm that we know of for the parameterized vertex cover problem $(\zeta(G), k)$ runs in $O^*(1.28^k)$. Here we suppressed the polynomial in n.[3] This algorithm does a more extensive case analysis.

3.4 Minimum fill-in

Recall Definition 2.21 on Page 31 of a chordal graph. A graph is chordal if it has no induced cycle of length more than three.

Of course, when a graph $G = (V, E)$ is not chordal then we can add some edges to the graph such that it becomes chordal. For example, if we add all edges between any pair of vertices x and y that are not adjacent in G, then the new graph is a clique, and a clique is obviously chordal.

Definition 3.18. *Let* $G = (V, E)$ *be a graph. A graph* $H = (V, E')$ *is a chordal embedding of* G *if*

1. G *and* H *have the same set of vertices, and*
2. H *is chordal, and*
3. $E \subseteq E'$.

A chordal embedding H is minimal if the deletion of any edge that is added to G, makes H non-chordal.

[3] J. Chen, I. Kanj, and G. Xia, Improved parameterized upperbounds for vertex cover, *Proceedings* 31th *MFCS*, Springer, LNCS 4162 (2006), pp. 238–249.

Definition 3.19. *The minimum fill-in problem asks for a chordal embedding* H *of a graph* G *such that the number of edges in* H *is minimal.*

The minimum fill-in problem is NP-complete.

Let $f(G)$ be the minimal number of edges that needs to be added to G in order to make G chordal. In this section we look at the parameterized $(f(G), k)$ problem.

For example, when G is a 4-cycle we need to add one edge. Furthermore, there are two choices to add the edge. If G is a 5-cycle, we need two edges and there are 5 choices to make G chordal by adding two edges.

If H is a t-cycle, then adding $t-3$ edges from one vertex x to all vertices of $H - N[x]$ makes the graph chordal. In fact, any minimal chordal embedding of H adds $t-3$ edges to H, although not all chordal embeddings are like the one above.

Lemma 3.20. *Let* C *be a cycle of length* t*. Then* $f(C) = t-3$*. There are*

$$\frac{1}{t-1}\binom{2(t-2)}{t-2} \leqslant 4^{t-3}$$

different embeddings of C *into a chordal graph with* $t + f(C)$ *edges.*

Proof. We leave most of the proof as an exercise.

The Catalan number C_n is defined as

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

(They are named after the Belgian mathematician E. Catalan (1814-1894). The Chinese mathematician Antu Ming discovered these numbers around 1730.)

The Catalan numbers satisfy the recurrence relation

$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n \quad \text{and} \quad C_1 = 1.$$

Notice that the bound $C_n \leqslant 4^{n-1}$ follows easily by induction from the recurrence relation:

$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n \leqslant \frac{4n+2}{n+2} 4^{n-1} \leqslant \frac{4(n+2)}{n+2} 4^{n-1} = 4^n.$$

□

Theorem 3.21. *The minimum fill-in problem is fixed-parameter tractable. The parameterized problem* $(f(G), k)$ *can be solved in* $O(4^k \cdot n^5)$ *time.*

Proof. We use the bounded search-tree technique.

First we show how to find an induced cycle of length at least four. Let x be a vertex and let y and z be two neighbors of x such that y and z are not adjacent. Then $\{x, y, z\}$ is contained in a chordless cycle of length at least four if and only if y and z are contained in a component of the graph induced by

$$(V \setminus N[x]) \cup \{y, z\}.$$

So, to look for an induced cycle of length at least four, we try all possible triples $\{x, y, z\}$ as above. For each such triple the algorithm finds a y, z-path

$$G - (N[x] \setminus \{y, z\}).$$

Adding the vertex x yields the chordless cycle of length at least four.

Notice that the number of these triples is bounded by

$$\sum_{x \in V} d(x)(d(x) - 1) \leqslant n \sum_{x \in V} d(x) = 2nm.$$

At each vertex in the search-tree T, the algorithm finds an induced cycle C of length at least four. Let t be the length of C. By Lemma 3.20 C cannot be made chordal when $t - 3 > k$. Otherwise, the search-tree branches over the $\frac{1}{t-1}\binom{2t-4}{t-2}$ possible minimal embeddings of C. The parameter k reduces by $t - 3$ since there are $t - 3$ edges added to C.

Let L_k be the number of leaves in the search-tree T. We claim that

$$L_k \leqslant 4^k.$$

We prove this by induction. If $k = 0$ then the algorithm does not branch. It checks in linear-time if the graph is chordal. If so, we are done, and if not, there is not suitable chordal embedding of G since we cannot add any edge.

By Lemma 3.20 and by induction, for $k \geqslant 1$,

$$L_k \leqslant 4^{t-3} \cdot L_{k-(t-3)} \leqslant 4^k.$$

We claim that the search-tree T has at most $2 \cdot 4^k + 1$ vertices. To see this, first note that it has at most one vertex of degree two, namely the root. All other internal vertices have one parent and at least two children, since every cycle of length at least four has at least two possible minimal embeddings.

Let n' and m' be the number of vertices and edges in T. We write L instead of L_k for the number of leaves in T. For a vertex x in T we write $d_T(x)$ for the number of neighbors of x in T. Then we have

$$\begin{aligned}2m' = 2(n'-1) &= \sum_{x,d_T(x)=1} 1 + \sum_{x,d_T(x)=2} 2 + \sum_{x,d_T(x)=3} 3 + \ldots \\ &\geqslant L + \sum_{x,d_T(x)=3} 3 + \sum_{x,d_T(x)=4} 4 + \ldots \\ &\geqslant L + 3(\sum_{x,d_T(x)=3} 1 + \sum_{x,d_T(x)=4} 1 + \ldots) \\ &\geqslant L + 3(n' - L - 1) = 3n' - 2L - 3.\end{aligned}$$

This implies that

$$n' \leqslant 2L + 1 \leqslant 2 \cdot 4^k + 1.$$

At each vertex in the search-tree we spend at most $O(n^3)$ time to look for for a suitable triple $\{x, y, z\}$ and $O(n^2)$ time to find the y, z-path. Thus, a cycle of length at least four is found in $O(n^5)$ time, if it exists. The total time that is used by this algorithm is therefore bounded by $O((2\cdot4^k+1)\cdot n^5) = O(4^k \cdot n^5)$. This proves the theorem. □

3.5 Homogeneous coloring of perfect graphs

Definition 3.22. *A homogeneous coloring of a graph is a partition of its vertices into cliques and independent sets.*

We refer to the sets of the partition as color classes. Each color class is either a clique or an independent set. The homogeneous coloring problem asks for a homogeneous coloring with a minimum number of color classes.

A (k, ℓ)-coloring of a graph is a partition of its vertices into k cliques and l independent sets, of which some may be empty.

Recall that a graph G is perfect when $\omega(H) = \chi(H)$ for every induced subgraph H of G. By the perfect graph theorem, when G is perfect, also $\bar{G}$ is perfect. Recall also Theorem 2.7 on Page 23 which shows that the coloring and clique number problem can be solved in polynomial time for G, and also for $\bar{G}$, when G is a perfect graph.

This is no longer true for homogeneous colorings. Wagner showed that finding a homogeneous coloring of a permutation graph with a minimal number of colors is NP-complete. However, for every fixed k and ℓ, the class of permutation graphs that admit a (k, ℓ)-coloring is characterized by a finite collection of forbidden induced subgraphs. This holds true even for the class of all perfect graphs.[4]

[4] K. Wagner, Monotonic coverings of finite sets, *Elektronische Informationsverarbeitung und Kybernetik* **20** (1984), pp. 633–639.
T. Feder and P. Hell, Matrix partitions of perfect graphs, *Discrete Mathematics* **306** (2006), pp. 2450–2460.

Theorem 3.23. *For every $k, \ell \in \mathbb{N}$ there exists a polynomial time algorithm that checks if there exists a (k, ℓ)-coloring of a perfect graph.*

Proof. There exists a finite set of graphs $\mathcal{F}(k, \ell)$ such that any perfect graph G has a (k, ℓ)-coloring if and only if it has no element of $\mathcal{F}(k, \ell)$ as an induced subgraph.

Let $H \in \mathcal{F}(k, \ell)$. Let c be the number of vertices of H. To check if H is an induced subgraph of $G = (V, E)$, the algorithm checks all subsets Ω with c vertices of V and checks if H and the induced subgraph $G[\Omega]$ of G are isomorphic. This algorithm can clearly be implemented to run in $O(n^{c+2})$ time, where n is the number of vertices in G.

Since $\mathcal{F}(k, \ell)$ is a finite set, it contains only a constant number of elements. Thus there are only a constant number of elements $H \in \mathcal{F}(k, \ell)$ that need to be checked. □

Although the algorithm of Theorem 3.23 is polynomial for each fixed k and ℓ, this is not a fixed-parameter algorithm. Furthermore, the theorem only shows the existence of a polynomial-time algorithm. To write down the algorithm we would need the set $\mathcal{F}(k, \ell)$. Not only contains this set a huge number of graphs but also, nobody knows what it is.

In the following theorem we show that the (k, ℓ)-coloring problem is fixed-parameter tractable on perfect graphs.

Theorem 3.24. *There exists an $O(f(k, \ell)n^c)$ algorithm which solves the (k, ℓ)-coloring problem on perfect graphs. Here*

$$f(k, \ell) = (k + \ell)^{2k \cdot \ell + \ell} \quad \textit{and } c \textit{ is a constant.}$$

Proof. Let $x_1, \ldots, x_n$ be an arbitrary ordering of the vertices and define

$$G_i = G[V_i] \quad \text{where} \quad V_i = \{x_1, \ldots, x_i\}.$$

If there exists a partition of G into k cliques and l independent sets, of which some may be empty, then such a partition exists for each of the graphs G_i.

In each step the algorithm checks if the current graph G_i has a partition of the vertices with k cliques and ℓ independent sets. Let $\mathcal{P}$ be a partition of G_{i-1} into k cliques and ℓ independent sets. Then, obviously, the graph G_i has a partition into $k + 1$ cliques and ℓ independent sets. To see this, just consider the partition

$$\mathcal{P} \cup \{\{x_i\}\}.$$

Consider a partition $\mathcal{P}$ of V_i into cliques and independent sets. Consider a bitvector L of length i where the t^{th} bit is one if the vertex x_t is in a clique of $\mathcal{P}$ and zero if the vertex x_t is in an independent set of $\mathcal{P}$.

Assume that we are given the bitvector L. Then we can check in polynomial time if it is *valid* by checking if

(a) the vertices with a 1 in L induce a perfect graph that has a clique cover with at most k cliques, and
(b) the vertices with a 0 in L induce a perfect graph that has chromatic number at most ℓ.

Let $\mathcal{P}$ and $\mathcal{Q}$ be two homogeneous colorings of G_i. Assume that $\mathcal{P}$ has k cliques and ℓ independent sets and that $\mathcal{Q}$ has k' cliques and ℓ' independent sets. Let L_P and L_Q be the bitvectors of $\mathcal{P}$ and $\mathcal{Q}$. The Hamming distance $H(L_P, L_Q)$ of L_P and L_Q is the number of bits that are different in L_P and L_Q. We claim that

$$H(L_P, L_Q) \leqslant k \cdot \ell' + k' \cdot \ell. \tag{3.3}$$

To see this, simply observe that any independent set in $\mathcal{P}$ and clique in $\mathcal{Q}$ can intersect in at most one vertex.

The algorithm runs as follows. Let $\mathcal{P}$ be a partition of G_{i-1}. Add the vertex x_i as a separate clique to $\mathcal{P}$ and let L be the bitvector of this partition. Let X be the set of vertices that have a 1 in L and let Y be the set of vertices with a 0 in L. First check if $G_i[X]$ has a clique cover with at most k cliques and if $G_i[Y]$ has chromatic number at most ℓ. If that is the case, then we are done; the algorithm outputs a clique cover and a coloring with k cliques and ℓ independent sets for the graph G_i.

Now assume that the clique cover number of $G_i[X]$ is $k+1$. Since G_i is perfect, it has an independent set S with $k+1$ vertices. Since G_i is a perfect graph it can find a maximum independent set S in polynomial time.

If there is a partition $\mathcal{Q}$ of G_i with k cliques and ℓ independent sets, then, by Equation (3.3),

$$H(L, L_Q) \leqslant \mu = 2k \cdot \ell + \ell. \tag{3.4}$$

If there exists an independent set S with $k+1$ vertices in $G_i[X]$, then there is a vertex $z \in S$ of which the corresponding bit is 0 in L_Q. For each bit in L which corresponds with a vertex z of S the algorithm does the following. It switches the bit of z in L to 0. Let L' be this bitvector. Then the algorithm recurses and tries to find a partition $\mathcal{Q}$ which is at distance at most $\mu - 1$ from L'.

The case where $G_i[Y]$ has chromatic number $k+1$ is similar.

To analyze the time complexity, observe that we may assume that $k \geqslant 1$ and $\ell \geqslant 1$, since otherwise we can just check if there is a clique cover or coloring with k or ℓ sets. The recursion tree has depth at most $2k \cdot \ell + \ell$ since each time the algorithm is recursively called the Hamming distance μ is decreased by one. Each node in the recursion tree corresponds with a clique of cardinality $\ell + 1$ or an independent set of cardinality $k+1$. Since

$$k+1 \leqslant k+\ell \quad \text{and} \quad \ell+1 \leqslant k+\ell \tag{3.5}$$

every node in the recursion tree has at most $k+\ell$ children. Thus the total number of recursions is bounded by

$$(k+\ell)^{2k\cdot\ell+\ell}. \tag{3.6}$$

The graph searches for a partition $\mathcal{P}$ in G_i for $i = 1,\dots,n$. In each transition from i to $i+1$ the graph spends $O(n^c)$ time in each node of a recursion tree with $(k+\ell)^{2k\ell+\ell}$ nodes. This proves the time bound and the theorem. □

3.6 Problems

3.1. Let's start with a problem from elementary school. Which function grows faster.

$$k^{\log(k)} \quad \text{or} \quad (\log(k))^k.$$

How about

$$2^{k^2} \quad \text{or} \quad 2^{2^{\log(k)+\log(k^2)}}.$$

A tricky one:

$$\text{the inverse Ackermann function } \alpha(k) \text{ or} \quad \log(\log(\cdots \log(k))\cdots).$$

3.2. Show that the parameterized chromatic number problem $(\chi(G), k)$ is not fixed-parameter tractable.
`Hint`: Recall that 3-coloring is NP-complete.

3.3. Let S be a set and let

$$\mathcal{S} = \{\, S_1, \,\dots,\, S_n \,\}$$

be a collection of n subsets of S such that $|S_i| = 3$ for all $i \in \{1,\dots,n\}$. The 3-hitting set problem asks for a subset $S' \subseteq S$ of minimal cardinality such that every triple S_i of $\mathcal{S}$ contains at least one element of S'. Use the bounded search-tree technique to show that the parameterized 3-hitting set problem can be solved in $O(3^k \cdot n^2)$ time.
`Hint`: Build a search-tree in which every vertex which is not a leaf has degree at most three.

3.4. Find a kernel for the 3-hitting set that we defined in Exercise 3.3.
`Hint`: Consider an element $x \in S$ that appears in at least $2k^2 + 1$ subsets S_i. Design a proof, similar to the proof of Lemma 3.8, which shows that x is in any 3-hitting set with at most k elements.

3.5. Solve the recurrences in (3.1) and (3.2).

3.6. Check that there are 14 minimal chordal embeddings of a labeled 6-cycle.

3.7. Prove Lemma 3.20 on Page 61.
`Hint`: Prove that the number of minimal chordal embeddings C_n of an $(n+2)$-cycle with vertices numbered $1, \ldots, n+2$ satisfies the Catalan recurrence relation

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}.$$

3.8. Let G be a graph and let $k \in \mathbb{N}$, $k \geqslant 3$. Design an algorithm that checks if G has a chordless cycle of length at least k and that out-puts one if it has. Is your algorithm a fixed-parameter algorithm?

3.9. Can we use the linear-time recognition algorithm for chordal graph of Tarjan and Yannakakis to find an induced cycle of length at least four?

3.10. Let H be a graph with c vertices. Design an algorithm that checks if a graph G has an induced subgraph which is isomorphic to H. What is the timebound for your algorithm?
`Hint`: This problem is not fixed-parameter tractable. That is easy to see. When H is an independent set with α vertices, the induced subgraph problem asks for an independent set in G with α vertices.

3.11. Check the details in the proof of Theorem 3.24. Especially

(a) check Formula 3.3, and
(b) check Formula 3.4, and
(c) check Formula 3.6.

4
Decomposition Trees

We have seen a few examples of decomposition trees of graphs already.

1. When G is a chordal graph it has a clique tree. We explained that concept in Section 2.4.1.
2. When G is a cograph then it has a cotree. We explained that in Section 2.2.1.
3. In Section 2.3.1 we explained the notion of a decomposition tree for distance-hereditary graph.
4. Interval graphs are chordal graphs. Interval graphs have a special clique tree, which is a path. We explained that in Theorem 2.46 on Page 40.

In Section 2.1 we hinted at a decomposition tree for perfect graphs that decomposes the graph into four basic classes of graphs. Furthermore, in Chapter 2 we have seen a few examples that show that a decomposition tree for a graph can be very useful for solving hard problems on that graph.

In this chapter we look at some parameterized decomposition trees. The advantage of the parametrization is that we no longer are restricted to some special class of graphs. Any graph can be decomposed using the tree decompositions by a suitable choice of the parameter.

We look at two kinds of decomposition trees. The first one is based on the clique trees for chordal graphs and the second one is based on the decomposition trees for distance-hereditary graphs.

Research on decomposition trees really took off with the graph minor theory. We review some of that theory in the next section.

4.1 Graph minors

Let's start with a basic lemma on natural numbers.

Lemma 4.1. *Consider a sequence of natural numbers*

$$n_1,\ n_2,\ n_3,\ \ldots \tag{4.1}$$

There exists an infinite subsequence which is nondecreasing.

Proof. Define

$$I = \{\, i \in \mathbb{N} \mid \forall_{j>i}\ n_j < n_i \,\}. \tag{4.2}$$

The subsequence of (4.1) of the number n_i with $i \in I$ is strictly decreasing. Since it is bounded from below by 1, it is finite.

Now let

$$k = \begin{cases} 0 & \text{if } I = \varnothing, \text{ and} \\ \max\{\, i \mid i \in I \,\} & \text{if } I \neq \varnothing. \end{cases} \tag{4.3}$$

Since I is finite this maximum exists. Consider the sequence

$$n_{k+1},\ n_{k+2},\ \ldots$$

For each element n_ℓ with $\ell > k$, there exists some element n_m with $m > \ell$ and $n_m \geqslant n_\ell$, since $\ell \notin I$. Then it is easy to construct an infinite non-decreasing subsequence. □

Consider an infinite sequence of graphs

$$G_1,\ G_2,\ \ldots \tag{4.4}$$

Let n_i be the number of vertices of the graph G_i, for all i. By lemma 4.1, there exists an infinite subsequence of (4.4) in which the number of vertices is nondecreasing. A similar proof shows that there is also an infinite subsequence of graphs of which the number of vertices *and* edges is nondecreasing. So we may assume that the sequence of graphs is 'nondecreasing,' if we order them by their numbers of vertices and edges.

Can we take a different ordering? Suppose that we order the set of all graphs by the induced subgraph relation. For two graphs G and H define $G \preceq H$ if the graph G is an induced subgraph of H. Notice that this relation is transitive, that is,

$$\text{if} \quad F \preceq G \quad \text{and} \quad G \preceq H \quad \text{then} \quad F \preceq H.$$

Not every pair of graphs is related by $\preceq$. We call $\preceq$ a quasi-order. If we call two graphs G and H 'the same' if $G \preceq H$ and $H \preceq G$, then the order is called a partial order. (In that case, we call two graphs that are isomorphic 'the same.')

Question:
Is it true that there always exists some integers i and j with $i < j$ such that $G_i \preceq G_j$?

Notice that, by Lemma 4.1 this is true for a sequence (4.1) of natural numbers; just take two elements of a (infinite) nondecreasing subsequence. Thus, if $\preceq$ is the ordering of graphs by numbers of vertices and edges, then the answer to the question is yes.

No; for the sequence of graphs this is not true if $\preceq$ is the ordering by induced subgraphs. An easy counterexample is

$$C_3,\ C_4,\ C_5,\ \ldots,$$

where C_i is a cycle of length i. No cycle C_i is an induced subgraph of another cycle C_j.

For the sequence of paths

$$P_1,\ P_2,\ P_3,\ \ldots,$$

where P_i is the path i vertices, the statement is true of course, since any path P_i is an induced subgraph of a path P_j with $j > i$. But for general sequences of graphs we cannot have that.

To make the statement above true for general sequences of graphs, we need to relax the condition that some graph G_i is an *induced* subgraph of another graph G_j. If we take the subgraph-relation, that is, if we only insist that G_i is a subgraph of G_j, then this is still not true. The same counterexample applies.

The way to make the statement true is to look at graph minors. To define graph minors we need the concept of an edge contraction.

Definition 4.2. *Let $G = (V, E)$ be a graph. Let $e = \{x, y\}$ be an edge in G. Contracting the edge e in G is the following operation which transforms G into a graph G'. Replace the two vertices x and y by one vertex, say xy. The neighborhood of xy in G' is the set*

$$N(x) \cup N(y) \setminus \{x, y\}.$$

Thus, contracting the edge $e = \{x, y\}$ squeezes the edge down to a single vertex xy. When z is a vertex that is adjacent to both x and y, then squeezing the edge $\{x, y\}$ to a single vertex xy, creates two edges $\{xy, z\}$ in G'. The double edge $\{xy, z\}$ is then replaced in G' by a single edge.

For example, let C be a cycle with n vertices. If we contract an edge in C we get a cycle with $n - 1$ vertices.

We now have the following. If (4.4) is an infinite sequence of graphs G_i such that each G_i is a cycle, then there exist $i < j$ such that G_i is obtained from G_j by a sequence of edge contractions. Just take two cycles G_i and G_j with $i < j$ such that G_j has at least as many vertices as G_i. We can obtain G_i from G_j by contracting sufficiently many edges in G_j (zero when $G_i = G_j$).

In other words, we now have the following. Define $G \preceq H$ if G is obtained from H by a sequence of edge contractions. If (4.4) is a sequence of cycles, then there exist $i < j$ such that $G_i \preceq G_j$.

The minor ordering is a little bit more general.

Definition 4.3. *A graph* G *is a* <u>*minor*</u> *of a graph* H *if* G *can be obtained from* H *by a sequence of operations, where each operation is one of the following.*

(i) *Deletion of a vertex.*
(ii) *Deletion of an edge.*
(iii) *Contraction of an edge.*

In other words, $G = (V, E)$ is a minor of a graph $H = (V', E')$ if for each vertex $x_i \in V$ there exists a subset of vertices $X_i \subseteq V'$ such that

(a) $X_i \cap X_j = \varnothing$, when $x_i \neq x_j$, and
(b) $H[X_i]$ is connected for all i, and
(c) Some vertex in X_i is adjacent to some vertex in X_j when $\{x_i, x_j\} \in E$.

Theorem 4.4 (The graph minor theorem). *Let* $G \preceq_m H$ *if* G *is a minor of* H. *Then in any infinite sequence of graphs*

$$G_1,\ G_2,\ \ldots$$

there exist two elements $i < j$ *with* $G_i \preceq_m G_j$.

It took Robertson and Seymour more than ten years to prove this theorem. They finished the proof in 2004, and they wrote it down in a sequence of 20 papers. The total length of the proof is more than 500 pages; so we skip it.

The first step in the proof is Kruskal's theorem. Kruskal proved the theorem in 1960 for the case where $G_1,\ G_2,\ \ldots$ is a sequence of trees.[1] Of course, for trees it is sufficient to consider only edge contractions; if T_1 and T_2 are trees then T_1 is a minor of T_2 if and only if T_1 can be obtained from T_2 by a sequence of edge contractions.

An important consequence of the graph minor theorem is the finite basis theorem.

[1] J. Kruskal, Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, *Transactions of the American Mathematical Society* **95** (1960), pp. 210–225.

Theorem 4.5. *Let $\mathcal{G}$ be a class of graphs which is closed under taking minors. That is, if $G \in \mathcal{G}$ and if H is a minor of G, then $H \in \mathcal{G}$. There exists a finite set of graphs Ω such that $G \in \mathcal{G}$ if and only if no element of Ω is a minor of G.*

Proof. Let Ω be the set of graphs that are *not* in $\mathcal{G}$ but for which every proper minor is in $\mathcal{G}$. So if $G \in \Omega$ then $G \notin \mathcal{G}$, but if we delete a vertex or an edge, or if we contract an edge in G, then the new graph is in $\mathcal{G}$.

Suppose that Ω is not finite. Then we can construct an infinite sequence

$$G_1, \; G_2, \; \ldots,$$

of graphs in Ω. Furthermore, we may assume that no two graphs G_i and G_j are the same (isomorphic). By Theorem 4.4, there exist $i < j$ such that G_i is a minor of G_j. Since G_i and G_j are not the same, G_i is a proper minor of G_j. But this is a contradiction; every proper minor of G_j is in $\mathcal{G}$, so $G_i \notin \Omega$. □

The set Ω is called the obstruction set for the class $\mathcal{G}$.

Let's look at an example. Let $\mathcal{G}$ be the class of all planar graphs. Let $G \in \mathcal{G}$. If we delete a vertex x from G, then the remaining graph $G - x$ is also planar. It is also easy to see that, if we delete an edge from G, then the remaining graph is planar. Finally, if we contract an edge in G, then it is fairly easy to see that the new graph G' is still planar. Thus the class $\mathcal{G}$ of planar graphs is closed under taking minors. By Theorem 4.5 there is a finite obstruction set Ω. For the class of planar graphs this set is

$$\Omega = \{ K_5, \; K_{3,3} \}.$$

Thus a graph is planar if and only if it has no K_5 or $K_{3,3}$ as a minor. Probably you know this theorem as Kuratowski's theorem, which was originally formulated a little bit different (without using edge contractions). Wagner showed that Kuratowski's theorem is equivalent to the formulation above.

In the next section we illustrate the power of the graph minor theorem by another example. We end this section with another important result of Robertson and Seymour.

Theorem 4.6. *Let H be a graph. There exists an $O(n^3)$ algorithm that tests if $H \preceq_m G$ for graphs G, where n is the number of vertices of G.*

In other words, the parameterized graph minor problem $(\preceq_m(G), H)$, which asks if the fixed graph H is a minor of a graph G, is fixed-parameter tractable. Notice that, when H is not fixed the problem is NP-complete. For example, when we take H a cycle with n vertices and G is a graph on n vertices, then $H \preceq_m G$ if and only if G has a Hamiltonian cycle. To test if G is Hamiltonian is NP-complete.

Notice that Theorem 4.6 provides an $O(n^3)$ algorithm to test if a graph G is planar. The theorem says we can test if G has a K_5-minor in $O(n^3)$ time. Also, we can test if G has a $K_{3,3}$-minor in $O(n^3)$ time. So, the total time to test if a graph is planar takes $O(n^3)$, because the obstruction set is finite (it has only two elements so we only need to do two tests).

Of course, we know that there is a linear-time algorithm to test planarity, but the result is much more general; it says that the recognition problem (see Page 19) for graph classes that are closed under taking minors can be solved in polynomial time.

Theorem 4.7. *Let $\mathcal{G}$ be a class of graphs which is closed under taking minors. There exists an $O(n^3)$ algorithm to test of a graph $G \in \mathcal{G}$.*

Proof. By Theorem 4.5 the class $\mathcal{G}$ has a finite obstruction set. By Theorem 4.6 we can test for each element in Ω whether it is a minor of a graph G or not. Since Ω is finite we only need to perform a constant number of these tests. This proves the theorem. □

4.2 Parameterized feedback vertex set

Recall Definition 2.17 on Page 29. A set F of vertices in a graph $G = (V, E)$ is a feedback vertex set in G if every cycle in G has at least one vertex in F. In other words, F is a feedback vertex set in G if $G - F$ is a forest.

For a graph G let $f(G)$ denote the minimal cardinality of a feedback vertex set in G. In this section we show that the parameterized feedback vertex set problem $(f(G), k)$ is fixed-parameter tractable.

Lemma 4.8. *Let $k \in \mathbb{N} \cup \{0\}$. Let $\mathcal{G}(k)$ be the class of graphs G with $f(G) \leqslant k$. The class $\mathcal{G}(k)$ is closed under taking minors.*

Proof. Let $G \in \mathcal{G}$. We prove that every minor of G is in $\mathcal{G}$.

Let F be a vertex set in $G = (V, E)$ with $|F| \leqslant k$.
Let $x \in V$ and let $G' = G - x$. If $x \in F$ then $F' = F \setminus \{x\}$ is a feedback vertex set in G', since $G' - F'$ is a forest. When $x \notin F$, then $G' - F$ is a forest, since the class of forests is minor closed.

Let $e = \{x, y\} \in E$. Let G' be the graph obtained from G by deleting the edge from E. Then F is a feedback vertex set in G', since $G' - F$ is a forest.

Let $e = \{x, y\} \in E$ and let G' be the graph obtained from G by contracting the edge $\{x, y\}$ to a single vertex xy. First assume that $x \notin F$ and that $y \notin F$. Then F is a feedback vertex set in G' since the class of forests is closed under taking minors.
Assume that $x \in F$ and that $y \notin F$. Define

$$F' = (F \setminus \{x\}) \cup \{xy\}.$$

Then F' is a feedback vertex set for G' since $G' - F' = G - F - \{y\}$ is a forest.
Assume that $x \in F$ and that $y \in F$. Define

$$F' = (F \setminus \{x, y\}) \cup \{xy\}.$$

Then F' is a feedback vertex set in G' since $G' - F' = G - F$ is a forest.

The proves the lemma, since $|F'| \leqslant k$ in all cases. □

We're done. The following theorem states the result.

Theorem 4.9. *The parameterized feedback vertex set problem* $(f(G), k)$ *is fixed-parameter tractable.*

Proof. Let $k \in \mathbb{N} \cup \{0\}$. Let $\mathcal{G}(k)$ be the class of graphs that have a feedback vertex set with at most k vertices. By lemma 4.8 the class $\mathcal{G}(k)$ is minor closed. By Theorem 4.5 there is a finite obstruction set $\Omega(k)$ such that

$$G \in \mathcal{G}(k) \quad \text{if and only if} \quad \forall_{H \in \Omega(k)}\; H \not\preceq_m G.$$

By Theorem 4.6 we can test $H \preceq_m G$ in $O(n^3)$ time, for each $H \in \Omega(k)$. Since $|\Omega(k)|$ is constant, the total time needed to test if $G \in \mathcal{G}(k)$ takes $O(n^3)$ time. □

4.3 Treewidth

Robertson and Seymour came up with the graph parameter treewidth during their research on graph minors.[2] It turns out that, if $\mathcal{G}$ is a class of graph which is closed under taking minors, and if $\mathcal{G}$ does not contain all planar graphs then there exists a $k \in \mathbb{N} \cup \{0\}$ such that all graphs in $\mathcal{G}$ have treewidth at most k.

For example, let $\mathcal{G}$ be the class of graphs that have a feedback vertex set with at most k vertices. By Lemma 4.8 this class is closed under taking minors. Furthermore, $\mathcal{G}$ does not contain all planar graphs. For example, if we take a sequence of larger and larger grids then it is easy to see that these have an

[2] N. Robertson and P. Seymour, Graph minors. II. Algorithmic aspects of tree-width, *Journal of Algorithms* **7** (1986), pp. 309–322.

increasing number of vertices in a minimum feedback vertex set. The result says that there exists a k' such that all graphs in $\mathcal{G}$ have treewidth at most k'.

In this section we take a close look at this important graph parameter.

Recall Definition 3.18 on Page 60. Let $G = (V, E)$ be a graph. A chordal embedding of G is a chordal graph $H = (V, E')$ with $E \subseteq E'$.

Definition 4.10. *Let* $G = (V, E)$ *be a graph. The treewidth of* G *is*

$$tw(G) = \min \{\, \omega(H) - 1 \mid H \textit{ is a chordal embedding of } G \,\}. \tag{4.5}$$

For example, let T be a tree. Then T has no cycles. Thus T is chordal, since it has no induced cycles of length more than three. Any chordal embedding of T has a clique number at least equal to the clique number of T. Therefore,

$$tw(T) = \begin{cases} 0 & \text{if } T \text{ has only one vertex} \\ 1 & \text{if } T \text{ has at least two vertices.} \end{cases} \tag{4.6}$$

Thus any nontrivial tree has treewidth one. This explains why we subtract one from $\omega(H)$ in Definition 4.10. Namely, in this way we have that any nontrivial tree has treewidth one, which is nice. In the following example we show the converse, that is, if the treewidth of a graph G is at most one then G is a forest.

Let's look at another example. Consider a cycle C. We claim that the treewidth of C is at least two. Assume that it is one. Let H be a chordal embedding of C and assume that $\omega(H) = 2$. Recall Theorem 2.31. Since H is chordal it has a perfect elimination ordering, say

$$[x_1, \ldots, x_n].$$

Every maximal clique in H is one of the sets

$$\Omega_i = \{\, x_j \mid j \geqslant i \quad \text{and} \quad j \in N[x_i] \,\}.$$

Since $\omega(H) = 2$, we have that

$$|\Omega_i| \leqslant 2 \quad \text{for all } i \in \{\, 1, \ldots, n \,\}.$$

Thus the perfect elimination ordering removes vertices one by one, and at each step the vertex has at most one neighbor in the remaining graph. This shows that H is a forest. Since C is a subgraph of H, also C is a forest, which is a contradiction because C is a cycle. Thus the treewidth of C is at least two.

We now show that the treewidth of C is exactly two. Let x be a vertex of C. Add edges from x to all other vertices in C. The new graph is chordal and has clique number three. This proves that $tw(C) = 2$.

In Section 4.3.1 we look at a greedy algorithm that checks if the treewidth of a graph G is at most two.

In the following theorem we prove that the class of graphs with treewidth at most k is closed under taking minors. For $k = 1$ this is true, since the class of graphs with treewidth at most one is the class of all forests.

To prove the general case we start with three easy lemmas.

Lemma 4.11. *Let* G *be a graph and let* G′ *be a subgraph of* G*. Then*

$$tw(G') \leqslant tw(G).$$

Proof. Let $G = (V, E)$ be a graph. Write $k = tw(G)$. Let $G' = (V', E')$ be a subgraph of G. Let H be a chordal embedding of G with $\omega(H) = k + 1$. Then $H[V']$ is a chordal graph, since the class of chordal graphs is hereditary. The graph $H[V']$ is a chordal embedding of G' since every edge in E' is also an edge in E, and so it is an edge in $H[V']$ since H is a chordal embedding of G. Obviously,

$$\omega(H[V']) \leqslant \omega(H) = k + 1,$$

and this implies that $tw(G') \leqslant k$.
This proves the lemma. □

Lemma 4.11 shows that the class of graphs with treewidth at most k is closed under taking subgraphs. To prove that this class is also closed under edge contractions, we prove this first for the class of chordal graphs.

Lemma 4.12. *The class of chordal graphs is closed under edge contractions.*

Proof. Let $G = (V, E)$ be a chordal graph. Let $e = \{x, y\}$ be an edge in G and let G' be the graph obtained from G by contracting the edge e to a single vertex xy. We prove that G' is chordal.

Recall Theorem 2.35 on Page 36. Since G is chordal there exists a tree T and a collection of subtrees of T

$$\{ T_x \mid x \in V \} \tag{4.7}$$

such that any two vertices a and b are adjacent if and only if $T_a \cap T_b \neq \varnothing$.

Consider the subtrees T_x and T_y. Since x and y are adjacent $T_x \cap T_y \neq \varnothing$. For the new vertex xy define the subtree

$$T_{xy} = T_x \cup T_y. \tag{4.8}$$

Then T_{xy} is a subtree of T. Any other vertex z is adjacent to xy in G' if and only if z is adjacent to x or to y in G. Any subtree T_z intersects T_x or T_y if and only if T_z intersects T_{xy}.

Thus the graph G' is the intersection graph of a set of subtrees of a tree, and so, G' is chordal. □

Lemma 4.13. *Let* $G = (V, E)$ *be a chordal graph and let* $e = \{x, y\}$ *be an edge in* G. *Let* G' *be the graph obtained from* G *by contracting the edge* e *to a single vertex* xy. *Then*

$$\omega(G') \leqslant \omega(G). \tag{4.9}$$

Proof. Let Ω' be a maximal clique in G'. If $xy \notin \Omega'$ then Ω' is a clique in G and so

$$|\Omega'| \leqslant \omega(G). \tag{4.10}$$

Now assume that $xy \in \Omega'$. If x is adjacent to all other vertices $z \in \Omega' \setminus \{xy\}$ then

$$\Omega = (\Omega' \setminus \{xy\}) \cup \{x\}$$

is a clique in G and so (4.10) holds.
Of course, by symmetry, (4.10) also holds when y is adjacent to all other vertices of $\Omega' \setminus \{xy\}$.

Assume that there exist vertices x' and y' in $\Omega' \setminus \{xy\}$ such that

$$x' \in N(x) \setminus N(y) \quad \text{and} \quad y' \in N(y) \setminus N(x). \tag{4.11}$$

Then $[x, x', y', y]$ is a 4-cycle in G which is a contradiction. □

The proof of the next theorem is now a piece of cake.

Theorem 4.14. *Let* $k \in \mathbb{N} \cup \{0\}$. *Let* $\mathcal{T}(k)$ *be the class of graphs* G *with*

$$tw(G) \leqslant k.$$

Then $\mathcal{T}(k)$ *is minor closed.*

Proof. Let $G = (V, E)$ be a graph and assume that $tw(G) \leqslant k$. Let G' be a minor of G. We prove that $tw(G') \leqslant k$.

When G' is a subgraph of G then the claim follows from Lemma 4.11.

Let $e = \{x, y\}$ be an edge in G. Assume that G' is obtained from G by contracting the edge e to a single vertex xy.

By definition, there exists a chordal embedding H of G with $\omega(H) = k + 1$. Since H is a chordal embedding of G, e is also an edge in H.

By lemma 4.12 the graph H', obtained from H by contracting the edge e in H to a single vertex xy, is chordal. Then H' is a chordal embedding of G'.

By Lemma 4.13 $\omega(H') \leqslant \omega(H)$.
This proves the theorem. □

Via the theory on graph minors we immediately obtain the following theorem.

Theorem 4.15. *The parameterized treewidth problem* $(tw(G), k)$*, which asks if a graph* G *has treewidth at most* k*, is fixed-parameter tractable. For each* $k \in \mathbb{N} \cup \{0\}$ *there exists an* $O(n^3)$ *algorithm which checks if the treewidth of a graph* G *is at most* k*.*

Proof. By Theorem 4.14 the class $\mathcal{T}(k)$ of graphs with treewidth at most k is minor closed. By Theorem 4.5 there exists a finite obstruction set $\Omega(k)$ and, by Theorem 4.6 we can test for each element $H \in \Omega(k)$ in $O(n^3)$ time if it is a minor of G. Now $tw(G) \leqslant k$ if and only if none of the graphs in $\Omega(k)$ is a minor of G. □

Remark 4.16. The theorem above only shows that there exists an $O(n^3)$ algorithm which checks if a graph has treewidth at most k. The graph minor theory does not provide the algorithm since the obstruction set is unknown. However, for each $k \in \mathbb{N} \cup \{0\}$ there is an explicit linear-time algorithm which checks if a graph has treewidth at most k.[3] In Section 4.3.3 we show that there exists an $O(n^{k+2})$ algorithm which checks if the treewidth of a graph is at most k. Although this is not a fixed-parameter algorithm, it is useful for small values of k.

4.3.1 An algorithm for treewidth two

Let's do something easy first.

In this section we show that there exists a linear-time algorithm to check if the treewidth of a graph is at most two.

We present the algorithm first. We prove that it is correct in theorem 4.18.

Let $G = (V, E)$ be a graph. The algorithm to check if the treewidth of G is at most two runs as follows.

[3] T. Kloks, *Treewidth - Computations and Approximations*, Springer-Verlag, Lecture Notes in Computer Science **842**, 1994.

1. If there exists a vertex x with at most one neighbor then delete it from the graph. Let $G' = G - x$. Check if the treewidth of G' is at most two. The treewidth of G is at most two if and only if the treewidth of G' is at most two.
2. Let x be a vertex with exactly two neighbors, y and z. Assume that y and z are adjacent. Then let $G' = G - x$. The treewidth of G is at most two if and only if the treewidth of G' is at most two.
3. Let x be a vertex with exactly two neighbors, y and z. Assume that y and z are not adjacent. Then add the edge $\{y, z\}$ to the graph and remove the vertex x. Let G' be this graph. The treewidth of G is at most two if and only if the treewidth of G' is at most two.
4. Assume that every vertex in G has at least three neighbors. Then the treewidth of G is more than two.

As you see, in the algorithm we use the fact that every graph with treewidth at most two has a vertex with at most two neighbors. We prove that in the following lemma. In Exercise 4.18 we ask you to prove a similar lemma for the graphs with treewidth at most k.

Lemma 4.17. *Let* $G = (V, E)$ *be a graph and assume that* $\text{tw}(G) \leqslant 2$. *Then every subgraph of* G *has a vertex with at most two neighbors.*

Proof. By Lemma 4.11 the class of graphs with treewidth at most two is closed under taking subgraphs. Therefore, it is sufficient to prove the claim for G.

Let H be a chordal graph embedding of G with $\omega(H) \leqslant 3$. To avoid confusion, we use the notation $N_H(x)$ to denote the neighborhood of a vertex x in the graph H.

By Theorem 2.31 on Page 34 there exists a perfect elimination ordering for H, say
$$\sigma = [x_1, \dots, x_n].$$
The vertex x_1 is a simplicial vertex in H and so its neighborhood is a clique. Since $\omega(H) \leqslant 3$ we have that $|N_H(x_1)| \leqslant 2$, and so x_1 has at most two neighbors in H. Because H is a chordal embedding of G we have that
$$N_G(x_1) \subseteq N_H(x_1),$$
and so the vertex x_1 has at most two neighbors in G.
This proves the lemma. □

We now show that the algorithm above is correct.

Theorem 4.18. *There exists an* $O(n)$ *algorithm which checks if the treewidth of a graph* G *is at most two.*

Proof. By Lemma 4.17, if G has n vertices and more than $2n$ edges then $tw(G) > 2$. Assume that G has at most $2n$ edges. Then we can compute the degree of every vertex x in $O(n)$ time. This shows that the algorithm that we described can be implemented to run in $O(n)$ time.

Let x be a vertex with at most two neighbors.

First assume that x is isolated. Let $G' = G - x$. By Lemma 4.11, if $tw(G') > 2$ then also $tw(G) > 2$. Assume that $tw(G') \leqslant 2$. Let H' be a chordal embedding for G' with $\omega(H') \leqslant 3$. Add the vertex x as an isolated vertex to H' and let H be this graph. Then H is a chordal embedding of G and $\omega(H) \leqslant 2$.

Assume that x has one neighbor, say y. Let $G' = G - x$. If $tw(G') > 2$, then also $tw(G) > 2$. Assume $tw(G') \leqslant 2$ and let H' be a chordal embedding of G' with $\omega(H') \leqslant 3$. Add the vertex x to H' and make it adjacent to y. Let H be this graph. Then H is a chordal embedding of G and $\omega(H) \leqslant 3$.

Assume that x has two neighbors, y and z, and assume that y and z are adjacent. Let $G' = G - x$. If $tw(G') > 2$ then $tw(G) > 2$ and so we may assume that $tw(G') \leqslant 2$. Let H' be a chordal embedding of G' with $\omega(H') \leqslant 3$. Let

$$\sigma' = [x_2, \ldots, x_n]$$

be a perfect elimination ordering for H'. Let H be the graph obtained from H' by adding the vertex x to H' and by making x adjacent to y and z. Consider

$$\sigma = [x, x_2, \ldots, x_n].$$

Since x is adjacent to y and z and since $\{y, z\}$ is an edge in H, the vertex x is a simplicial vertex in H. This proves that σ is a perfect elimination ordering for H. Thus, H is a chordal embedding for G and $\omega(H) \leqslant 2$.

Finally, assume that x has exactly two neighbors, say y and z, and assume that y and z are not adjacent. Let G' be the graph obtained from G by adding the edge $\{y, z\}$ and by removing the vertex x. We claim that G' is a minor of G. To see that, observe that contracting the edge $\{x, y\}$ in G produces the graph G'.

By Theorem 4.14, if $tw(G') > 2$ then $tw(G) > 2$. Assume that $tw(G') \leqslant 2$ and let H' be a chordal embedding of G' with $\omega(H') \leqslant 3$. Notice that $\{y, z\}$ is an edge in H'. Add the vertex x as a simplicial to H', by making it adjacent to y and z. Then H is a chordal embedding of G and $\omega(H) \leqslant 3$.
This proves the theorem. □

4.3.2 k-Trees

When a graph G has treewidth k then it has an embedding into a chordal graph H with clique number at most $k + 1$. In this section we show that there is a special chordal graph embedding for G.

Definition 4.19. *Let* $k \in \mathbb{N} \cup \{0\}$*. A* k*-tree is a chordal graph in which every maximal clique has cardinality* $k + 1$.

Theorem 4.20. *Let* G *be a graph and let* $k = tw(G)$*. Then there exists an embedding of* G *into a* k*-tree.*

Proof. Let H be a chordal graph embedding of G with $\omega(H) = k + 1$. Write $N_H(x)$ for the neighbors of x in the graph H.
We prove that there exists an embedding H' of H such that H' is a k-tree.

By Theorem 2.31 there exists a perfect elimination ordering for H, say

$$\sigma = [x_1, \dots, x_n].$$

For $i = n$ down to 1 add neighbors to $N_H(x_i)$ as follows. If $i \geqslant n - k$ then make x_i adjacent to all vertices of

$$\{\, x_{i+1},\ \dots,\ x_n \,\}.$$

This step makes a $(k + 1)$-clique of $\{x_{n-k}, \dots, x_n\}$.

For $i = n - k - 1$ down to 1 consider

$$N_i = N_H(x_i) \cap \{\, x_{i+1},\ \dots,\ x_n\}.$$

Then $|N_i| \leqslant k$ since it is a clique in H of cardinality at most k.

By induction, every maximal clique in

$$H'_i = H'[\{\, x_{i+1},\ \dots,\ x_n \,\}]$$

has cardinality $k + 1$. Since N_i is a clique it is contained in a maximal clique Ω_i in H'_i. Furthermore, $|N_i| \leqslant k$.

Choose a subset $N'_i \subseteq \Omega_i$ which contains N_i and which has cardinality k. Make x_i adjacent to all vertices N'_i.

The graph H' is a chordal graph embedding of G and every maximal clique in H' has cardinality $k + 1$. □

4.3.3 An $O(n^{k+2})$ algorithm for treewidth

Computing the treewidth of a graph is NP-complete. Arnborg, *et al.*, designed the first polynomial-time algorithm which checks if a graph has treewidth at most k.[4] In this section we explain their algorithm.

Recall Theorem 2.27 on Page 32 which says that a graph is chordal if and only if every minimal separator is a clique.

[4] S. Arnborg, D. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM Journal on Algebraic and Discrete Methods* **8** (1987), pp. 277–284.

Lemma 4.21. *Let* $G = (V, E)$ *be a chordal graph and let* S *be a minimal separator in* G*. Let* C *be a component of* $G - S$*. Then* C *contains a vertex* x *such that* x *is simplicial in* G*.*

Proof. Consider the graph $G[S \cup C]$. Since the class of chordal graphs is hereditary, this induced subgraph is chordal. If $G[S \cup C]$ is a clique then we are done. In that case every vertex $x \in C$ is simplicial since $N(x) \subseteq S \cup C$ and so $N(x)$ is a clique.

Otherwise, there are two nonadjacent vertices a and b in $G[S \cup C]$. Let S' be a minimal a, b-separator in $G[S \cup C]$. We claim that S' is also a minimal a, b-separator in G. To see this, let P be an a, b-path in $G - S'$. Then P must contain at least one vertex which is not in $S \cup C$. But then P contains two vertices from S, and so P has a shortcut. This proves the claim.

Consider the components of $G - S'$. Let C_a and C_b be the components of $G - S'$ which contain a and b. Since S is a clique, the set $S \setminus S'$ can have vertices in at most one of C_a and C_b. Assume that C_a contains no vertices from S. Then $C_a \subseteq C$.

Furthermore, $|C_a| < |C|$, since $a \notin C_a$. By induction C_a contains a vertex x which is simplicial in G. □

Let G be a chordal graph and let S be a minimal separator in G. Let C be a component of $G - S$. By Lemma 4.21 there exists a perfect elimination ordering for G which eliminates all vertices of C first.

The algorithm is based on the following observation.

Let H be a k-tree embedding of a graph G. Let S be a minimal separator in H. Then $|S| = k$. Furthermore, S is a separator in G. Let $C_1, \dots, C_t$ be the components of $G - S$. We show that there exists a k-tree embedding H' of G such that

1. S is a separator in H', and
2. each component C_i of $G - S$ is a component of $H' - S$, and
3. each $H'[S \cup C_i]$ is a k-tree.

Let G_i be the subgraph of G induced by $S \cup C_i$. The graphs $H[S \cup C_i]$ are chordal embeddings of $G[S \cup C_i]$. Since the treewidth of each $H[S \cup C_i]$ is at most k, there exist k-tree embeddings H_i for $H[S \cup C_i]$. Define the k-tree H' as the union of the graphs H_i.

Theorem 4.22. *Let* G *be a graph and let* $k \in \mathbb{N} \cup \{0\}$*. There exists an* $O(n^{k+2})$ *algorithm which check if the treewidth of* G *is at most* k*.*

Proof. When G has at most $k+1$ vertices then $tw(G) \leqslant k$. In that case the algorithm makes a clique of G and it reports YES. Otherwise, if G has more than $k+1$ vertices, by Theorem 4.20 the graph has $tw(G) \leqslant k$ if and only if G has a k-tree embedding. The algorithm that we describe below finds a k-tree embedding H if $tw(G) \leqslant k$.

The algorithm first makes a list of all pairs (S, C) where

(i) S is a separator in G and $|S| = k$, and
(ii) C is a component of $G - S$.

Notice that G has at most n^k separators S of cardinality k. The number of components in $G - S$ is at most n for each separator S, so the list contains at most n^{k+1} pairs (S, C).

For a pair (S, C) let $G^*(S, C)$ be the graph obtained from $G[S \cup C]$ by making a clique of S.

The algorithm checks if there exists a separator S with $|S| = k$, such that for each component C of $G - S$, $G^*(S, C)$ has a k-tree embedding. If that is the case, then the treewidth of G is at most k, and otherwise the treewidth of G is more than k.

First, the algorithm sorts the pairs (S, C) according to nondecreasing cardinalities $|C|$. It does that by bucket sort in time $O(n^{k+1})$. It processes the pairs (S, C) in that order as follows.

When there is a k-tree embedding of $G^*(S, C)$ then, by Lemma 4.21, there is a perfect elimination ordering that eliminates the vertices of C first. Let $c \in C$ be the last vertex in this elimination ordering. Then c is adjacent to all vertices of S in the k-tree embedding. The algorithm tries all vertices $c \in C$ as candidates.

Consider a pair (S, C). Let $c \in C$ and define

$$S(c) = S \cup \{c\}.$$

Let $C_1(c), \ldots, C_t(c)$ be the components of

$$G[C \setminus S(c)].$$

For each $i \in \{1, \ldots, t\}$ check if there exists a separator $S_i(c)$ such that

(i) $\{ y \mid y \text{ has a neighbor in } C_i(c) \} \subseteq S_i(c)$, and
(ii) $|S_i(c)| = k$, and
(iii) $(S_i(c), C_i(c))$ is a YES instance, that is, there exists a k-tree embedding of $G^*(S_i(c), C_i(c))$.

We claim that the pair (S, C) can be processed in $O(|C|^2)$ time. Notice that a k-tree has $O(kn)$ edges, which is $O(n)$ since k is fixed. Thus for each choice of $c \in C$ the components $C_i(c)$ can be determined in $O(|C|)$ time (see Exercise 4.18). Since C is connected, the vertex c has a neighbor in each component $C_i(c)$. There are $|C|$ choices for c and $O(k)$ candidates for the separators $S_i(c)$ because

$$c \in S_i(c) \quad \text{and} \quad |S \setminus S_i(c)| = 1.$$

For each component $C_i(c)$ it can be checked if there is a suitable separator $S_i(c)$ (which reports YES) in constant time. This proves the claim.

Notice that this shows that the overall time complexity of the algorithm is bounded by $O(n^{k+2})$, since there are $O(n^k)$ separators S of cardinality k and the summation of $|C|^2$ over all components C of $G - S$ is bounded by $O(n^2)$.

If there exist a vertex $c \in C$ and a collection of separators $S_i(c)$ such that the answer is YES for all $G^*(S_i(c), C_i(c))$ then the algorithm answers YES for $G^*(S, C)$. Otherwise, it answers NO for $G^*(S, C)$.

If there exists a separator S with $|S| = k$ such that for all components C of $G - S$ the algorithm above reports a YES for the pair (S, C), then the treewidth of G is at most k. Otherwise the treewidth is more than k. □

4.3.4 Maximum clique in graphs of bounded treewidth

As an example, we show that, for every $k \in \mathbb{N} \cup \{0\}$ there exists a linear-time algorithm to compute the clique number for graphs of treewidth at most k.

The usual strategy to solve NP-complete problems for graphs of bounded treewidth is dynamic programming on the clique tree of the chordal embedding. For the clique number problem there is an easier algorithm, that we describe below.

Let $k \in \mathbb{N} \cup \{0\}$ and let G be a graph with treewidth at most k. Then there exists a chordal embedding H of G with $\omega(H) \leqslant k + 1$. We mentioned that there exists a linear-time algorithm to construct the graph H, although we did not give you the details of this algorithm.

Theorem 4.23. *Let $k \in \mathbb{N} \cup \{0\}$. There exists an $O(n)$ algorithm to compute $\omega(G)$ for graph $G \in \mathcal{T}(k)$.*

Proof. Let $G = (V, E)$ be a graph and assume that $tw(G) \leqslant k$. We assume that we have a chordal embedding H of G with $\omega(H) \leqslant k + 1$.

Obviously, if M is a maximal clique in G then M is a clique in H. Let $\mathcal{M}$ be the set of all maximal cliques in H. Then

$$\omega(G) = \max \{ \, |W| \mid \exists_{M \in \mathcal{M}} \; W \subseteq M \quad \text{and} \quad G[W] \text{ is a clique} \, \}. \tag{4.12}$$

By Lemma 2.32 on Page 34, H has at most n maximal cliques, where n is the number of vertices of H.

Let

$$\sigma = [x_1, \dots, x_n] \tag{4.13}$$

be a perfect elimination ordering for H. This can be obtained in linear time by the algorithm of Tarjan and Yannakakis. Notice that H has at most $n \cdot k$ edges, and so this algorithm runs in $O(n \cdot k) = O(n)$ time, since k is a constant.

Define

$$N_i = \{\, x_j \mid j \geqslant i \quad \text{and} \quad x_j \in N[x_i] \,\}. \tag{4.14}$$

Then Formula (4.12) becomes

$$\omega(G) = \max \{\, |W| \mid W \subseteq N_i \quad i \in \{1, \dots, n\} \text{ and } G[W] \text{ is a clique } \}. \tag{4.15}$$

Each N_i has at most $k+1$ vertices, so the number of subsets of N_i is bounded by 2^{k+1}. Using a suitable datastructure we can check in constant time if two vertices are adjacent in G. So, for each subset $W \subseteq N_i$ we can check in $O(k^2)$ time if $G[W]$ is a clique or not. Thus the overall time complexity of this algorithm is $O(2^k \cdot k^2 \cdot n) = O(n)$, since k is a constant. □

4.3.5 Chromatic number for graphs of bounded treewidth

In this section we illustrate the standard technique, namely dynamic programming on the decomposition tree. As an example, we show how to compute the chromatic number for graphs of bounded treewidth.

Theorem 4.24. *Let $k \in \mathbb{N} \cup \{0\}$. There exists a linear-time algorithm that computes $\chi(G)$ for graphs $G \in \mathcal{T}(k)$.*

Proof. Let $k \in \mathbb{N} \cup \{0\}$ and let $G = (V, E)$ be a graph in $\mathcal{T}(k)$. There exists a linear-time algorithm which computes a chordal embedding H of G with $\omega(H) \leqslant k+1$. Since H is perfect $\chi(H) \leqslant k+1$ and so, since G is a subgraph of H also $\chi(G) \leqslant k+1$.

Consider a clique tree $(T, \mathcal{S})$ for H. For a vertex p in T let $S_p \in \mathcal{S}$ be the maximal clique of H that is assigned to p.

Root T at some arbitrary vertex r. For a vertex p in T let T_p be the subtree of T which is rooted at p. Thus $T_r = T$.

For a vertex p in T let

$$V_p = \{\, x \in V \mid \exists_i \; i \text{ is a vertex in } T_p \text{ and } x \in S_i \,\}. \tag{4.16}$$

Consider all vertex colorings of S_p. A coloring of S_p with ℓ colors is a partition

$$\mathcal{P} = \{\, Q_1,\ \dots,\ Q_\ell \,\}$$

of S_p into ℓ color classes. Some of these color classes may be empty.

Since $|S_p| \leqslant k+1$, there are at most ℓ^{k+1} different partitions of S_p, namely, each vertex in S_p has one of ℓ different colors.

A partition $\mathcal{P}$ of S_p is valid when no two vertices in the same color class of P are adjacent in G.

Let $1 \leqslant \ell \leqslant k+1$. Let $\mathcal{P}$ be a valid partition of S_p with ℓ color classes. Let $b_p(\mathcal{P}, \ell)$ be a boolean variable which is TRUE if and only if there exists a coloring of $G[V_p]$ with ℓ colors such that the vertices of S_p are colored as in $\mathcal{P}$. For each vertex p the algorithm determines all values $b_p(\mathcal{P}, \ell)$ as follows.

First assume that p is a leaf. Then $V_p = S_p$. The algorithm determines all values $b_p(\mathcal{P}, \ell)$ by trying all partitions of S_p. If $\mathcal{P}$ has ℓ color classes (possibly some empty), and if each color class is an independent set in $G[S_p]$, then $b_p(\mathcal{P}, \ell)$ is TRUE. Otherwise it is FALSE.

Let p be an internal vertex of T and let $c_1, \dots, c_t$ be the children of p in T. For each $i \in \{1, \dots, t\}$ let $S_i \in \mathcal{S}$ be the maximal clique of H which is assigned to the vertex c_i in T.

The important observation is that no two vertices

$$x \in S_i \setminus S_p \quad \text{and} \quad y \in S_j \setminus S_p$$

are adjacent when $i \neq j$. (This follows from Definition 2.33 on Page 35 of the clique tree.)

Let $\mathcal{P} = \{Q_1, \dots, Q_\ell\}$ be a valid partition of S_p. Then $b_p(\mathcal{P}, \ell)$ is TRUE if and only if for each child c_i of p there exists a valid partition $\mathcal{P}' = \{Q'_1, \dots, Q'_\ell\}$ such that

(i) $b_{c_i}(\mathcal{P}', \ell)$ is TRUE, and
(ii) if $Q'_j \in \mathcal{P}'$ then

$$Q'_j \cap S_p \subseteq Q_j.$$

The graph G has a coloring with ℓ colors if and only if there exists a valid partition $\mathcal{P}$ with ℓ color classes at the root r, such that $b_r(\mathcal{P}, \ell)$ is TRUE.
This proves the theorem. □

4.4 Rankwidth

Recall the definition of a decomposition tree (T, f) of a distance-hereditary graph $G = (V, E)$ as described in Section 2.3.1. Thus T is a rooted binary tree and f is a bijective map from the vertices of G to the leaves of T.

For each edge $\{p, c\}$ in T where c is the child of p, let W_e be the set of vertices of G that are mapped to leaves in the subtree of c. The twinset Q_e is the subset of W_e that have neighbors in $V \setminus W_e$.

By definition of the decomposition tree, all vertices of Q_e have the same neighbors in $V \setminus W_e$.

Consider the adjacency matrix A of G. This is an $n \times n$ matrix with diagonal elements $A_{i,i} = 0$ for all i. For $i \neq j$, $A_{i,j} = 1$ if the vertices x_i and x_j are adjacent in G and $A_{i,j} = 0$ if x_i and x_j are not adjacent in G.

Let $e = \{p, c\}$ be an edge in T. The cutmatrix C_e is the submatrix of A of which the rows are the vertices of W_e and the columns are $V \setminus W_e$.

Since all vertices of Q_e have the same neighbors in $V \setminus W_e$, the cutmatrix C_e has a shape

$$C_e = \begin{pmatrix} J & 0 \\ 0 & 0 \end{pmatrix} \tag{4.17}$$

where J is a matrix with all ones. The first set of rows of C_e correspond with the vertices of Q_e and the first set of columns correspond with the neighbors of vertices in Q_e in $V \setminus W_e$.

When $Q_e = \varnothing$, this matrix becomes the zero matrix. If $Q_e = W_e$ then the shape of the matrix becomes

$$\begin{pmatrix} J & 0 \end{pmatrix}.$$

When the vertices of Q_e are adjacent to all vertices of $V \setminus W_e$ then the matrix becomes

$$\begin{pmatrix} J \\ 0 \end{pmatrix}.$$

So in general, the cutmatrix is a submatrix of the matrix in (4.17).

The binary field GF[2] has two elements, 0 and 1, and the addition and multiplication are defined by

$$0 + 0 = 0 \quad \text{and} \quad 0 + 1 = 1 + 0 = 1 \quad \text{and} \quad 1 + 1 = 0$$

and

$$0 \cdot 0 = 0 \quad \text{and} \quad 0 \cdot 1 = 1 \cdot 0 = 0 \quad \text{and} \quad 1 \cdot 1 = 1.$$

Column vectors can be added by using the addition rules of GF[2] entrywise. The multiplication of a vector with a scalar $\alpha \in \{0, 1\}$ is done entrywise, with the rules for multiplication of GF[2].

The rank of a matrix over GF[2] is the number of linearly independent columns. A set of column vectors $\mathbf{a}_1, \ldots, \mathbf{a}_t$ is linearly dependent if there exist scalars $\alpha_1, \ldots, \alpha_t$, with $\alpha_i \in \{0, 1\}$ for each $i \in \{1, \ldots, t\}$, such that

$$\alpha_1 \cdot \mathbf{a}_1 + \cdots + \alpha_t \cdot \mathbf{a}_t = \mathbf{0}$$

where $\mathbf{0}$ is the all-zero vector.

The rank of a matrix is computed using the well-known Gauss elimination method.

Definition 4.25. *A graph G has rankwidth k if there exists a decomposition tree (T, f) such that every cutmatrix has rank over GF[2] at most k.*

Here, a decomposition tree (T, f) is defined as above, so T is a rooted binary tree and f is a bijection from the vertices of G to the leaves of T.

We denote the class of graphs with rankwidth k by $\mathcal{R}(k)$.

Since we are only interested in matrices that have small rank, the following observation is useful. It shows that, we can avoid the Gaussian elimination and instead just look at the number of different rows or columns.

Lemma 4.26. *Let A be a matrix and let k be its rank over GF[2]. Then A has at most 2^k different columns.*

Proof. Since the rank over GF[2] of A is k there is a basis $\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$. Then every column $\mathbf{c}$ of A can be written as a linear combination,

$$\mathbf{c} = \alpha_1 \cdot \mathbf{a}_1 + \cdots + \alpha_k \cdot \mathbf{a}_k.$$

The scalars $\alpha_i \in \{0, 1\}$, for all $i \in \{1, \ldots, k\}$. There are at most 2^k different linear combinations and so there are at most 2^k different columns in A. $\square$

Lemma 4.27. *Let G be a graph with rankwidth k. Then the rankwidth of its complement $\bar{G}$ is at most $k + 1$.*

Proof. Let (T, f) be a decomposition tree for G such that the rank over GF[2] of every edge in T is at most k.

For $\bar{G}$ we use the same decomposition tree. We write J for the all-one matrix. Each cutmatrix C_e changes to $J + C_2$, which switches the zeroes and ones in C_e into ones and zeroes.

Since the rank over GF[2] of C_e is k, there is a basis $\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ for the column space of C_e. Then every column of $J + C_e$ can be written as

$$\mathbf{j} + \mathbf{c} = \mathbf{j} + \alpha_1 \cdot \mathbf{a}_1 + \cdots + \alpha_k \cdot \mathbf{a}_k,$$

where $\mathbf{j}$ is the all-one vector. This shows that the dimension of the columns space of $J + C_e$ is at most $k + 1$. Here we use the linear algebra property that $\{\mathbf{j}, \mathbf{a}_1, \ldots, \mathbf{a}_k\}$ contains a basis for the columns of $J + C_e$, with a minimal number of elements. $\square$

Let's look at an example.

Lemma 4.28. *A graph is a cograph if and only if it has a decomposition tree such that every cutmatrix or its transpose has a shape*

$$\begin{pmatrix} J & 0 \end{pmatrix}$$

or a submatrix of that. Here J *is the all-ones matrix. Consequently, cographs have rankwidth one.*

Proof. By Theorem 2.10, a graph $G = (V, E)$ is a cograph if and only if it has a cotree (see Section 2.2.1). Any binary tree with at least two vertices has two leaves that have the same parent. Let x and y be two vertices of G that are mapped to sibling leaves. We claim that x and y are twins.

To see that, notice that x is adjacent to $z \neq x$ if and only if their common ancestor is labeled with an $\otimes$-operator. Let $z \in V \setminus \{x, y\}$. then the common ancestor of x and z is the same as the common ancestor of y and z. This proves the claim.

Actually, a graph is a cograph if and only if every induced subgraph with at least two vertices has a twin. Namely, if H is a cograph, then the operation which creates a twin of some vertex x in H does not create an induced P_4. Thus the class of cographs is closed under creating twins.

We prove the lemma by induction on the number of vertices. Consider a cotree (T, f) for the cograph G. Let x and y be twins, mapped to sibling leaves of T. Remove the vertex x and let $G' = G - x$.

A decomposition tree (T', f') for G' is obtained from the decomposition tree (T, F) for G by removing the leaf that is mapped to x, and by contracting the edge in T that connects y to its parent. The $\otimes$ or $\oplus$ label of the parent of x and y in T disappears.

By induction, the cutmatrix of every edge in T' has the shape as claimed in the lemma. Now consider the edges of T. For every edge e in T which is not incident with x or y, the cutmatrix C_e is obtained from the cutmatrix C'_e in T' by making a copy of the row or column that represents the vertex y since x and y have the same neighbors.

Consider an edge e of T which is incident with a leaf. Let a be the vertex that is mapped to that leaf. Then the cutmatrix of e is

$$\begin{pmatrix} 1 \cdots 1 & 0 \cdots 0 \end{pmatrix},$$

where the single row represents a. The first set of columns, those with a 1, are the neighbors of a and the final set of columns, those with a 0, are the nonneighbors of a. Thus in any decomposition tree, the cutmatrix of an edge which is incident with a leaf, has the desired shape.
This proves the lemma. □

In the same manner as in the proof of Lemma 4.28 one can show that every cutmatrix in the decomposition tree of a distance-hereditary graph, has a shape

$$\begin{pmatrix} J & 0 \\ 0 & 0 \end{pmatrix}.$$

We ask you to prove that in Exercise 4.20.

Many NP-complete problems can be solved in polynomial time for graphs of $\mathcal{R}(k)$. The reason is this. Consider a decomposition tree (T, f) for a graph in $\mathcal{R}(k)$. Let $e = \{p, c\}$ be an edge in T, where p is the parent of c. Let W_e be the set of vertices that are mapped to leaves in the subtree rooted at c. By Lemma 4.26 the vertices of W_e have at most 2^k different neighborhoods in $V \setminus W_e$. In the next section we discuss a class of problems that can be solved in polynomial time for graph in $\mathcal{R}(k)$.

Algorithms for graphs in $\mathcal{R}(k)$ use the dynamic programming strategy on the decomposition tree. Luckily, this decomposition tree can be obtained in $O(n^3)$ time.[5] We omit the description of this algorithm.

We have seen in Theorem 4.14 that the class $\mathcal{T}(k)$ of graphs with treewidth at most k is closed under taking minors. As a consequence, this class of graphs is characterized by a finite obstruction set.

Unfortunately, this is not true for the class $\mathcal{R}(k)$ of graphs with rankwidth at most k. For example, consider a clique with 5 vertices. This graph has rankwidth one (it is a cograph). Now remove edges such that a 5-cycle remains. The 5-cycle is not distance-hereditary; its rankwidth is two. Thus the 5-cycle is a minor of the 5-clique but the rankwidth of the 5-cycle is bigger than the rankwidth of the 5-clique. Actually, this example shows that the class $\mathcal{R}(k)$ is not even closed under taking subgraphs.

For the class $\mathcal{R}(k)$ we define a different ordering.

Definition 4.29. *Let* $G = (V, E)$ *be a graph. Let* $x \in V$. *The local complementation at* x *is the operation which replaces all edges in* $N(x)$ *by nonedges and all nonedges in* $N(x)$ *by edges.*

Definition 4.30. *A graph* H *is a vertex-minor of a graph* G *if* H *can be obtained by a sequence of operations, each of which is either*

(a) *a deletion of a vertex, or*
(b) *a local complementation.*

In Exercise 4.21 we aks you to prove that $\mathcal{R}(k)$ is closed under taking vertex-minors.

Oum and Symour proved the following theorem.

[5] P. Hliněný and S. Oum, Finding branch-decompositions and rank-decompositions, *SIAM Journal on Computing* **38** (2008), pp. 1012–1032.

Theorem 4.31. *Let* $k \in \mathbb{N} \cup \{0\}$ *and let*

$$G_1,\ G_2,\ \ldots \tag{4.18}$$

be an infinite sequence of graphs in $\mathcal{R}(k)$. *There exist indices* $i < j$ *such that* G_i *is a vertex-minor of* G_j.

As a consequence of this theorem the graphs in $\mathcal{R}(k)$ are characterized by a finite collection of forbidden vertex-minors. To prove this, Oum and Seymour showed that, for $k \geqslant 1$, each graph in the obstruction set has at most

$$\frac{6^{k+1}-1}{5}$$

vertices.

This finite obstruction set for $\mathcal{R}(k)$ leads to a polynomial recognition algorithm. However, this is non-constructive since the obstruction set is unknown. Furthermore, the timebound for this algorithm is much worse than the constructive $O(n^3)$ algorithm by Hliněný and Oum, mentioned above. When H is a fixed graph, then checking if a graph G contains H as a vertex-minor is quite complicated.

4.5 Monadic second-order logic

The most natural way to express and classify graph-theoretic problems is by means of logic.

In monadic second order logic a finite sentence is a formula that uses quantifiers $\forall$ and $\exists$. The quantification is over vertices, edges, and subsets of vertices and edges. Relational symbols are $\neg$, $\in$, $=$, and, or, $\subseteq$, $\cup$, $\cap$, and $\Rightarrow$. Some of these are superfluous.

Although the minimization or maximization of the cardinality of a subset is not part of the logic, one usually includes them.

As an example we show how the feedback vertex set can be formulated in monadic second-order logic. Obviously, we can formulate that $V \setminus F$ is a forest by a sentence which expresses that every subset W of $V \setminus F$ has a vertex of degree at most one but the following method to formulate that a graph is a forest is more informative.

First we show that the property that a graph is connected can be formulated in this logic. A graph is *disconnected* if the vertex set V has a partition $\{V_1, V_2\}$ such that there is no edge between a vertex in V_1 and a vertex in V_2. Note that this property can be formulated in monadic second-order logic.

Next we show that we can formulate the property that a graph has no induced cycle of length more than three in this logic. A graph has an induced cycle of length at least four if there exist three vertices w, w_1, and w_2 such that w is adjacent to w_1 and w_2, and w_1 and w_2 are not adjacent, and such that the following holds. Let

$$V' = (V \setminus N[w]) \cup \{w_1, w_2\}.$$

Obviously, there exists an induced cycle containing $\{w, w_1, w_2\}$ if and only if w_1 and w_2 are contained in a component of $G[V']$.

Checking for a triangle is easy and so, one can formulate the property that a graph is a forest in monadic second-order logic. Finally, the fact that a subset F is a feedback vertex set can be formulated by stating that $V \setminus F$ induces a forest.

Another easy example is this. Fix some graph H. Then one can formulate in monadic second-order logic if a graph G contains H as an induced subgraph.

One more example. It is only a little bit more complicated to show that for every graph H there is a monadic second-order formula that expresses that a graph G contains H as a minor. We leave it as an exercise.[6]

Let $k \in \mathbb{N} \cup \{0\}$ and let $\Omega(k)$ be the finite obstruction set for $\mathcal{T}(k)$. Then one can formulate the question if the treewidth of a graph G is at most k by a finite monadic second-order formula. Namely, write down the monadic second order formula which checks if some $H \in \Omega(k)$ is a minor of G.

Courcelle popularized monadic second-order logic by proving that any graph-theoretic problem that can be formulated in monadic second-order logic can be solved in linear time for graphs of bounded treewidth.[7]

Theorem 4.32. *Let* $k \in \mathbb{N} \cup \{0\}$. *Any problem that can be formulated in monadic second-order logic can be solved in linear time for graphs in* $\mathcal{T}(k)$.

A restricted form of this logic is where one does not allow quantification over subsets of edges. The C_2MS-logic is such a restricted monadic second-order logic where one can furthermore use a test whether the cardinality of a subset is even or odd. Using this logic one can formulate for example if a fixed graph H is a vertex-minor of a graph G.

[6] `Hint:` Use the alternative formulation of a minor, which appears after Definition 4.3.

[7] B. Courcelle and S. Oum, Vertex-minors, monadic second-order logic, and a conjecture by Seese, *Journal of Combinatorial Theory, Series B* **97** (2007), pp. 91–126.

Remark 4.33. Notice the difference. One can formulate that a graph is Hamiltonian by expressing this as the existence of a suitable subset of the edges. However, this formulation is not valid in C_2MS-logic.

The class of graphs that have rankwidth at most k is much larger than the class of graphs with treewidth at most k. (There exists a function

$$f : \mathbb{N} \to \mathbb{N}$$

such that if a graph has treewidth at most k then its rankwidth is at most $f(k)$. The converse is of course not true; any clique has rankwidth one, but its treewidth is the number of vertices minus one.) The set of problems that can be solved in polynomial time for graphs of bounded rankwidth is consequently a bit smaller.

Theorem 4.34. *Let $k \in \mathbb{N} \cup \{0\}$. Any problem that can be formulated in C_2MS-logic can be solved in $O(n^3)$ time for graphs in $\mathcal{R}(k)$. When a decomposition tree for the graph is a part of the input, then these algorithms run in linear time.*

4.6 Problems

4.1. Consider an infinite sequence of trees

$$T_1, \; T_2, \; \ldots \tag{4.19}$$

Let $\preceq$ denote the induced subgraph relation. It it true that for any sequence of trees, as in (4.19) there always exist integers $i < j$ such that $T_i \preceq T_j$?
`Hint`: Consider a sequence of paths $P_1, P_2, \ldots$ In each path P_i, say with ends a_i and b_i, add two leaves, a_i^* and b_i^*. Make a_i^* adjacent to $N(a_i)$ and make b_i^* adjacent to $N(b_i)$ (in other words; create a twin of each end).

4.2. Prove the alternative definition of a minor, which appears after Definition 4.3.

4.3. Prove that, for two trees T_1 and T_2, if $T_1 \preceq_m T_2$ then there exist a sequence of edge contractions in T_2 that produces T_1.

4.4. Check that the class of all planar graphs is closed under taking minors.

4.5. Let $\mathcal{T}$ be the class of all forests.

(a) Prove that $G \in \mathcal{T}$ if and only if G has no triangle as a minor.
(b) Prove that you can test in $O(n^3)$ time if a graph G is a forest.
(c) Can you do better? (This is a joke.)

4.6. A graph is outerplanar if it has a plane embedding such that all vertices are on the outerface. Let $\mathcal{O}$ be the class of outerplanar graphs.

(i) Prove that $\mathcal{O}$ is closed under taking minors.
(ii) Prove that a graph is outerplanar if and only if it has no K_4 nor $K_{2,3}$ as a minor. Thus the obstruction set for $\mathcal{O}$ is

$$\Omega = \{ K_4, \; K_{2,3} \}.$$

4.7. Consider graphs of treewidth two.

(1) Prove that a graph has treewidth at most two if and only if it has no K_4 as a minor.
(2) Use Problem 4.6 to show that every outerplanar graph has treewidth at most two.
(3) Give an example of a graph which has treewidth two but which is not outerplanar.

4.8. Consider the class of graphs that you can draw on the surface of a torus (that is a donut) without crossing lines. The graphs in this class are called toroidal. Show that this class of graphs is minor closed.
The obstruction set is finite, but it seems that it contains at least 16000 elements. Can you think of one element?
`Hint`: It is not difficult to show that you can draw K_5 on a torus. Also, K_6 and K_7 are toroidal. In general, if you can draw a graph G in the plane with at most one crossing then G is toroidal. Toroidal graphs have chromatic number at most 7; thus K_8 is not toroidal.

4.9. Show that $K_{3,3}$ is a minor of the Petersen graph.

4.10. Let $k \in \mathbb{N} \cup \{0\}$. Let $\mathcal{C}(k)$ be the class of graphs G that have at most k vertex-disjoint cycles.

(i) Show that $\mathcal{C}$ is closed under taking minors.
(ii) What is the obstruction set $\Omega(k)$?
`Hint`: Consider the graph Q which is the union of $k + 1$ triangles. Prove that $\Omega(k) = \{Q\}$.
(iii) Let $\mathcal{G}(k)$ be the class of graphs that have a feedback vertex set with at most k vertices. Show that $\mathcal{G}(k) \subseteq \mathcal{C}(k)$. Can you think of a graph that is in $\mathcal{C}(k)$ but not in $\mathcal{G}(k)$?
`Hint`: Consider the 5-wheel, that is, a 5-cycle plus one vertex adjacent to all vertices in the cycle. How many vertex-disjoint cycles are there in the 5-wheel? What is the minimal cardinality of a feedback vertex set?

4.11. Let $k \in \mathbb{N} \cup \{0\}$. Let $\mathcal{K}(k)$ be the class of graphs that have a vertex cover with at most k vertices.

(a) Show that $\mathcal{K}(k)$ is closed under taking minors.
(b) Consider the graph H which is the union of $k + 1$ edges. Then $H \notin \mathcal{K}(k)$.
(c) Can you think of another graph than Q which is in the obstruction set of $\mathcal{K}(k)$?

4.12. Let $k \in \mathbb{N} \cup \{0\}$. Let $\mathcal{D}(k)$ be the class of graphs $G = (V, E)$ for which there is a subset D of vertices with $|D| \leqslant k$ such that every vertex of $G - D$ has degree at most two in $G - D$.

(1) Show that $\mathcal{D}(k)$ is minor closed.
(2) Show that there exists an $O(n^3)$ algorithm to check if $G \in \mathcal{D}(k)$.

4.13. Let T be a tree and let

$$\{ T_x \mid x \in V \} \tag{4.20}$$

be a collection of subtrees of T. Define a graph $G = (V, E)$ by

$$\{x, y\} \in E \quad \text{if and only if} \quad |T_x \cap T_y| \geqslant 2. \tag{4.21}$$

I. What can you say about G?
`Hint`: Show that any graph is the edge-intersection graph of a collection of subtrees of a star (a tree of diameter two).
II. Let T be a star and fix the degree of T by some $k \in \mathbb{N}$. Let $\mathcal{G}(k)$ be the class of graphs that are edge-intersection graphs of T. What can you say about $\mathcal{G}(k)$?
III. Is the class $\mathcal{G}(k)$ minor closed?[8]

4.14. Let G be a graph. Prove that

$$\omega(G) \leqslant tw(G) + 1 \quad \text{and} \quad \chi(G) \leqslant tw(G) + 1.$$

4.15. Let G be a k-tree and consider a vertex coloring of G with $k + 1$ colors such that no two adjacent vertices have the same color. Let $1 \leqslant p \leqslant k + 1$ and let C_p any subset of p colors. Let G_p be the subgraph of G induced by the vertices that have colors in C_p. Prove that G_p is a $(p - 1)$-tree.
`Hint`: Use a perfect elimination ordering for G and prove the claim by induction.

4.16. Let G be a chordal graph. Prove that if G is not a clique then it has two simplicial vertices that are not adjacent.
`Hint`: Use Lemma 4.21.

4.17. Let G be a k-tree. Prove that every minimal separator in G has cardinality k and it is the intersection of two maximal cliques of cardinality $k + 1$.

4.18. Let G be a graph and let $k = tw(G)$. Prove that every induced subgraph of G has a vertex with at most k neighbors.

[8] J. Gramm, J. Guo, F. Hüffner and R. Niedermeier, Data reduction, exact, and heuristic algorithms for clique cover, *proceedings* 8th *ALENEX06*, SIAM (2006), pp. 86–94.

4.19. Prove that the number of edges in a k-tree is

$$\binom{k+1}{2} + (n-k-1)k = nk - \binom{k+1}{2}.$$

4.20. Let (T, f) be a decomposition tree for a distance-hereditary graph. Prove that the cutmatrix of every edge in T has a shape

$$\begin{pmatrix} J & 0 \\ 0 & 0 \end{pmatrix}.$$

`Hint:` A distance-hereditary graph has an isolated vertex, or a pendant vertex, or a twin. Use this fact to prove that for every edge e in T the vertices of W_e have only two different neighborhoods in $V \setminus E_e$, namely, all vertices of the twinset Q_e have the same neighbors in $V \setminus W_e$ and all vertices of $W_e \setminus Q_e$ have the same neighbors (zero) in $V \setminus W_e$.

4.21. Let $k \geqslant 1$. Prove that the class $\mathcal{R}(k)$ of graphs of rankwidth at most k is closed under taking vertex-minors.

4.22. Let $G = (V, E)$ be a graph. Let $S \subseteq V$ be a subset of vertices. A switch of G with respect to S is the following operation. Change all edges with one endvertex in S and the other in $V \setminus S$ into nonedges, and change all nonedges with one endvertex in S and the other in $V \setminus S$ into edges.

Let $\mathcal{G}$ be the class of graphs that can be switched to a cograph. In other words, $G \in \mathcal{G}$ if there exists a set of vertices in G such that the switch of G with respect to S changes G into a cograph.

Prove that graphs in $\mathcal{G}$ have rankwidth at most two.

4.23. Consider the following class $\mathcal{G}$ of graphs G for which there is a coloring of the vertices with colors black and white, such that for every induced subgraph H of G, one of the following holds.

(i) H has one vertex (either black or white).
(ii) There exists a partition $\{V_1, V_2\}$ of the vertices of H such that either
   (a) every vertex x of V_1 is adjacent to all vertices of V_2 that have the same color as x, or
   (b) every vertex x of V_1 is adjacent to all vertices of V_2 that have the opposite color of x.

(1) Prove that the graphs in $\mathcal{G}$ have rankwidth at most two.
(2) Prove that there exists a polynomial-time algorithm to check if a graph G is in $\mathcal{G}$.

4.24. Let $k \in \mathbb{N} \cup \{0\}$. Prove that the chromatic number problem can be solved in linear time on graphs with treewidth at most k by showing that the problem can be formulated in monadic second-order logic.
`Hint:` Use the result of Exercise 4.14, that is,

$$tw(G) \leqslant k \quad \text{implies that} \quad \chi(G) \leqslant k+1.$$

4.25. Prove that the domatic partition problem can be solved in linear time on graphs of bounded treewidth.
`Hint`: First prove that the domatic number of a graph $G \in \mathcal{T}(k)$ is bounded by $k+1$.

References

1. R. Niedermeier, *Invitation to fixed-parameter algorithms*, Oxford Lecture Series in Mathematics **31**, 2006.
2. F. Fomin and D. Kratsch, *Exact exponential algorithms*, Springer-Verlag, 2010.
3. M. Golumbic, *Algorithmic graph theory and perfect graphs*, Elsevier, Annals of Discrete Mathematics **57**, 2004.
4. R. Diestel, *Graph theory*, Springer-Verlag, Graduate Text in Mathematics **173**, 2010.
5. T. Kloks, *Treewidth – computations and approximations*, Springer-Verlag, Lecture Notes in Computer Science **842**, 1994.

Index