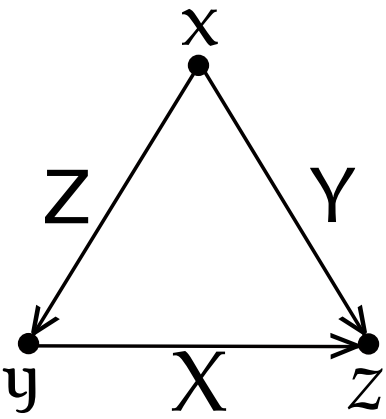
Ton Kloks and Yue-Li Wang

Advances in Graph Algorithms

October 10, 2013



Preface

We prepared this book as a course textbook for our students in Taiwan. Our aim was to write a book about some currently popular topics such as exponential algorithms, fixed-parameter algorithms and algorithms using decomposition trees of graphs. Especially for this last topic we found it necessary to include a chapter on graph classes. The chapter on decomposition trees includes some basics of the graph minor theory and such topics as tree decompositions and rank decompositions. To explain these concepts we found it beneficial to include a chapter which explains the classes of chordal graphs and distance-hereditary graphs.

In the chapter on exponential algorithms we start with some introductory examples on independent sets, chromatic number, domatic partitions, set cover, etc. The highlight of this chapter is the method based on the inclusion - exclusion principle. We exemplify this method by algorithms for chromatic number and triangle partitions.

In the chapter on graph classes we put an emphasis on perfect graphs. We explain the perfect graph theorem and the strong perfect graph theorem. We discuss the special classes of chordal graphs and interval graphs, comparability graphs, cographs and distance-hereditary graphs. We put an emphasis on the clique trees of chordal graphs, cotrees for cographs and the split-decomposition trees for distance-hereditary graphs. We briefly explain the use of these data structures by some elementary examples.

In the chapter on fixed-parameter algorithms we explain the basic notions of the search tree technique and the concept of a kernelization. We explain the search tree technique by some basic examples such as vertex cover and minimum fill-in. As an example of a kernelization we show how a matching can be used to find a kernel for vertex cover. We explain the iterative compression techniques by the fixed-parameter algorithm for graph bipartization, feedback vertex set and homogeneous colorings of perfect graphs.

In the chapter on decomposition trees we start with an explanation of the graph minor theory. As a basic example we show that this implies that feedback vertex set is fixed-parameter tractable. Next, we introduce treewidth as a parametrization of chordal graphs. We show that the class of graphs of bounded treewidth is closed under minors and, as a consequence, that the class can be recognized in $O(n^2)$ time. We give an easy linear-time algorithm for the recognition of graphs with treewidth two. For general k we explain the historic $O(n^{k+2})$ algorithm of Arnborg, Corneil and Proskurowski, for the recognition of partial k-trees. We introduce tree decompositions as the clique trees of chordal embeddings of a graph. Likewise, we introduce rank decompositions as a parametrization of the decomposition trees for distance-hereditary graphs. We close the chapter with a brief discussion of monadic second-order logic.

After each chapter we included some basic exercises. For experienced students these exercises are probably too easy. We made the decision to concentrate on elementary exercises in order not to distract the student too much from the main topics. The exercises are primarily meant as a check for the students that they understand the material of the chapter.

One of us, Ton Kloks, taught this course during the spring semester of 2012 at the Department of Computer Science of National Tsing Hua University. We are indebted to our students for their enthusiasm. Special thanks go to Ching-Hao Liu, who operated as a teaching assistant. He spotted numerous typos in the original manuscript. Thanks to him this text has greatly improved.

Ton Kloks also wishes to express his gratitude towards the Department of Computer Science of National Tsing Hua University for their hospitality and for giving him the opportunity to teach this course. He is indebted to the National Science Council of Taiwan for their financial support.

Ton Kloks
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan.

Yue-Li Wang
Department of Information Management
National Taiwan University of Science and Technology
No. 43, Sec. 4, Keelung Rd., Taipei, 106, Taiwan
ylwang@cs.ntust.edu.tw

Contents

1 Introduction 1

2 Exponential Algorithms 5
2.1 Independent set 6
2.2 Chromatic number 10
2.2.1 Three-coloring 12
2.3 Domatic partition 13
2.4 The traveling salesman problem 15
2.5 Set cover 16
2.6 Dominating set 17
2.6.1 Dominating set in bipartite graphs 21
2.7 Inclusion – exclusion 21
2.7.1 Triangle partition 25
2.7.2 An improved algorithm for triangle partition 27
2.8 Subset sum 30
2.9 Problems 31

3 Graph Classes 35
3.1 Perfect graphs 36
3.2 Comparability graphs 40
3.2.1 Recognition of comparability graphs 47
3.3 Cographs 53
3.3.1 Cotrees 55
3.3.2 Finding cliques in cographs 56
3.4 Distance-hereditary graphs 57
3.4.1 Decomposition trees for DH-graphs 60
3.4.2 Feedback vertex set in DH-graphs 62
3.5 Chordal graphs 64
3.5.1 Clique trees 68
3.5.2 Algorithms for independent set, clique and vertex coloring in chordal graphs 71

3.6 Interval graphs 72
3.7 Permutation graphs 80
3.7.1 Interval containment graphs 82
3.7.2 Cliques and independent sets in permutation graphs . . . 83
3.8 Problems 84

4 Fixed-parameter Algorithms 89
4.1 Vertex cover 92
4.2 A kernel for vertex cover 95
4.3 A better search-tree algorithm for vertex cover 98
4.4 Minimum fill-in 101
4.5 Odd cycle transversals 104
4.5.1 Feedback vertex set 108
4.6 Homogeneous coloring of perfect graphs 114
4.7 Color coding 116
4.8 Problems 118

5 Decomposition Trees 121
5.1 Graph minors 121
5.2 Parameterized feedback vertex set 127
5.3 Treewidth 128
5.3.1 An algorithm for treewidth two 133
5.3.2 k-Trees 135
5.3.3 An $O(n^{k+2})$ algorithm for treewidth 137
5.3.4 Maximum clique in graphs of bounded treewidth 139
5.3.5 Chromatic number for graphs of bounded treewidth 141
5.3.6 Disjoint cycles 142
5.4 Pathwidth 145
5.5 Rankwidth 150
5.6 Monadic second-order logic 155
5.7 Problems 158

References 163

Index 165

1

Introduction

This book is about graphs with an emphasis on algorithms. The choice of our topics is personal. The topics that appear as chapters in this book are a selection of the currently 'hot' research areas in graph theory and graph algorithms.

We expect that our students are familiar with the basic notions of graphs and algorithms. In this chapter we review some of our notations.

A graph G is an ordered pair of sets, $G = (V, E)$. The elements of the set V are called the vertices of G. Sometimes we call the vertices 'points.' We require that the set V is nonempty. The elements of the set E are called the edges of G. Sometimes we call an edge a 'line.' An edge is a set of two vertices. When the graph G is not clear from the context we use $V(G)$ and $E(G)$ to denote its set of vertices and its set of edges. The two vertices of an edge are called the endpoints of the edge. Two vertices x and y are adjacent in G if $\{x, y\} \in E$; they are nonadjacent if $\{x, y\} \notin E$. When $e = \{x, y\} \in E$ we say that e is an edge which runs between x and y or, simpler, we say that e is an edge between x and y.

The neighborhood of a vertex x is the set of vertices that are adjacent to x. We use the notation $N(x)$ to denote the neighborhood of x. When the graph G is not clear from the context we use the notation $N_G(x)$. When two vertices are adjacent we also call them neighbors of each other. When two vertices are not adjacent we call them nonneighbors. We use $N[x] = \{x\} \cup N(x)$ as a notation for the 'closed neighborhood' of a vertex x. The degree of a vertex x is the number of its neighbors, that is, $|N(x)|$. For a subset $C \subseteq V$ of vertices we write

$$N(C) = \{\, y \mid y \in V \setminus C \quad \text{and} \quad N(y) \cap C \neq \varnothing \,\}.$$

Thus, for $C \subseteq V$, $N(C)$ is the set of vertices that are not in C and that have a neighbor in C. We also write $N[C] = C \cup N(C)$ for the closed neighborhood of a subset C.

All our graphs will be finite, that means that V is a finite set. A graph $G = (V, E)$ is called empty if $E = \varnothing$.

Let $G = (V, E)$ be a graph. A graph $G' = (V', E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. The complement of a graph $G = (V, E)$ is the graph $\bar{G}$ with

$$V(\bar{G}) = V \quad \text{and} \quad E(\bar{G}) = \{\, \{x, y\} \mid x \in V \text{ and } y \in V \text{ and } \{x, y\} \notin E \,\}.$$

Let $G = (V, E)$ be a graph and let $S \subseteq V$ be some nonempty subset of vertices. The graph $G[S]$ induced by S has the set S as its set of vertices. As its set of edges it has those elements of E of which both endpoints are in S. If $G = (V, E)$ is a graph and if $S \subset V$ is a set of vertices such that $V \setminus S \neq \varnothing$ then we write $G - S$ for the subgraph of G induced by $V \setminus S$. If $S = \{x\}$ then we write $G - x$ instead of $G - \{x\}$. If $F \subseteq E$ is a set of edges then we write $G - F = (V, E \setminus F)$. If $F = \{e\}$ then we write $G - e$ instead of $G - \{e\}$.

Let $G = (V, E)$ be a graph. The linegraph $L(G)$ of G is the graph which has the edges of G as its vertices. Two vertices of $L(G)$ are adjacent if the corresponding edges in G have a vertex in common.

Let $G = (V, E)$ be a graph. A path is an ordered sequence of distinct vertices. A path has at least one vertex. We use the notation

$$P = [x_1, \ldots, x_t] \tag{1.1}$$

to denote a path P. The set of vertices of P is $\{x_1, \ldots, x_t\} \subseteq V$. There is an edge between any two vertices in P if and only if the vertices are consecutive in the ordering. In some cases we allow edges between vertices that are not consecutive in the ordering. In that case, those edges are called chords. When we want to emphasize that a path has no chords, we will call it 'chordless.'

The vertices x_1 and x_t of a path P as in (1.1) are called the endvertices, or endpoints of P. The length of a path is the number of edges in the path. When the endvertices of a path P are a and b then we refer to P as an a, b-path.

A cycle in $G = (V, E)$ is an ordered sequence of distinct vertices. We use the notation

$$C = [c_1, \ldots, c_t] \tag{1.2}$$

to denote a cycle C. A cycle has at least three vertices. There is an edge between consecutive vertices and between the first and last vertex of C; these edges are called the edges of C. A cycle is even or odd if the number of edges of C is even or odd. If there are no other edges between the vertices of C, we say that C is chordless. If there are other edges between vertices of C then these edges are called chords. The length of a cycle is the number of vertices

in it. If the cycle is chordless then the length is also the number of edges in it. When the set of vertices of a chordless path or a chordless cycle is the set of vertices V of G, we say that G is a path or a cycle. Notice that we call a graph a path or cycle only when it is chordless.

A graph $G = (V, E)$ is connected if there is a path between any two vertices. Being connected by a path defines an equivalence relation on the vertices of G. The equivalence classes are called the components of G. If a component consists of only one vertex then this vertex is called an isolated vertex of G.

A tree is a connected graph without any cycles. When $T = (V, E)$ is a tree then $|E| = |V| - 1$. Especially when T is a tree we call its vertices points and its edges lines. When G is a graph in which all components induce trees in G we call G a forest. When $T = (V, E)$ is a tree then a vertex of degree one is called a leaf of T.

A clique in a graph $G = (V, E)$ is a nonempty set of vertices Ω such that every pair of vertices in Ω is adjacent. When Ω is a clique with a maximal number of elements among all cliques in G we say that Ω is a maximum clique in G. A maximal clique in G is a clique Ω such that every vertex $z \in V \setminus \Omega$ has at least one nonneighbor in Ω.

An independent set in a graph $G = (V, E)$ is a nonempty set of vertices A such that there is no edge between any pair of vertices in A. In other words, A is an independent set in G if A induces a clique in $\bar{G}$. An independent set A in a graph G is maximum if it has the largest cardinality among all independent sets in G. Of course, there could be more than one maximum independent set. An independent set A in G is maximal if every vertex of $V \setminus A$ has at least one neighbor in A.

A coloring of a graph $G = (V, E)$ is a partition of its vertices such that each part of the partition is an independent set. The parts of the partition are called the color classes. The minimum number of color classes in a coloring of G is the chromatic number of G. We denote the chromatic number of G by $\chi(G)$. A graph is bipartite if its chromatic number is at most two. A graph is bipartite if and only if it has no cycles with an odd length.

An edge coloring of a graph $G = (V, E)$ is a coloring of its linegraph $L(G)$. Each color class is a matching in G, that is, it's a set of edges in G such that no two edges in the set have a vertex in common. The chromatic index of G is the chromatic number of $L(G)$.

A clique cover of a graph $G = (V, E)$ is a coloring of its complement $\bar{G}$. Thus each color class induces a clique in G. The minimal number of color

classes in a clique cover of G is called the clique cover number; we denote it by $\kappa(G)$. Sometimes we allow the cliques that form the color classes of a clique cover to overlap. It is always easy to change a clique cover with overlapping cliques into one, with at most as many cliques, where the cliques form a partition.

Usually we denote the number of vertices and edges of a graph $G = (V, E)$ as n and m. We express the worst-case runtime of an algorithm on G in the number of vertices of G. An algorithm on G is polynomial if it can be implemented to run in $O(p(n))$ time, for some polynomial $p(n)$. Usually we express the runtime only in the number of vertices. We make an exception for the linear-time algorithms. Such algorithms run in time $O(n+m)$. A quadratic algorithm is an algorithm that runs in $O(n^2)$ time. A cubic time algorithm runs in $O(n^3)$ time, etc.

In our book we won't go into complexity theory. We assume that our reader is familiar with the basic notion of NP-completeness. Primarily, we will be concerned with the question whether there exists a polynomial-time algorithm for a problem or whether the problem is NP-complete.

We won't go into many details about data structures. Usually it will be sufficient to assume that the graph is represented by adjacency lists, or an adjacency matrix, or a mix of these. The adjacency matrix of a graph $G = (V, E)$ with n vertices is an $n \times n$ matrix. The rows and columns are indexed by the vertices. A matrix entry with row x and column y is 1 if $\{x, y\} \in E$ and otherwise the entry is 0. The adjacency matrix needs $O(n^2)$ space. The advantage of the adjacency matrix is that one can test in $O(1)$ time if two vertices are adjacent or not. Adjacency lists require only linear space, that is $O(n + m)$ space. A clever mix of the two data structures, which uses the adjacency matrix without initializing it, makes it possible to test if two vertices are adjacent in $O(1)$ time while using only linear space.

Many other concepts will pass the revue. We will explain them when we meet them, as clearly as we can. Let's just get started. We hope the reader will enjoy the choice of our topics and the way we present them in this book.

2

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.

2.1 Independent set

An independent set M in a graph $G = (V, E)$ is *maximal* if every 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 2.1. *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 2.2. *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 2.3. *Let M be a maximum independent set in G and let $x \in M$. Then $M = \{x\}$ or $M \setminus \{x\}$ 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, unless $M = \{x\}$, $M \setminus \{x\}$ 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 2.2 and 2.3 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.

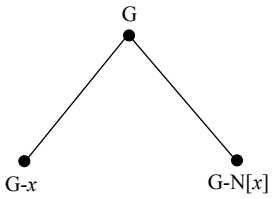


Fig. 2.1. This figure illustrates the search tree for maximum independent set.

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,

$$\alpha(G) = \max\{\ \alpha(G - x),\ 1 + \alpha(G - N[x])\ \}. \tag{2.1}$$

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 (2.1), 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.

An *isolated vertex* in G is a vertex without neighbors. A pendant vertex is a vertex with exactly one neighbor. The next lemma deals with graphs in which every vertex has degree at most two.

Lemma 2.4. *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 (2.1).

The *length* of a path or cycle is the number of edges in the path or the cycle.

(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 grow 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 2.4 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 (2.1) and by Lemma 2.4 we have the following recurrence relation for $T(n)$.

$$T(n) \leqslant T(n-1) + T(n-4) + O(n+m). \tag{2.2}$$

Perhaps you wonder why we write $T(n-4)$. If x is a vertex of degree i then the Formula (2.1) says $T(n-i-1)$ instead of $T(n-4)$. However, notice that $T(n)$ is a non-decreasing function (see Exercise 2.3), 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 (or a calculator) yield $\omega \approx 1.3803$. This proves the following theorem.

Theorem 2.5. *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 2.6. 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 2.2). It is one of the well-known techniques that are used in the design of exponential algorithms.

Remark 2.7. Very often the timebound for an exponential algorithm follows from a recurrence relation like Formula (2.2). If you want to study exact algorithms it's a good idea to obtain some basic knowledge about solving recurrence relations.

Remark 2.8. The highly-praised Bron-Kerbosch algorithm generates all maximal independent sets in $O(3^{n/3})$ time. The Bron-Kerbosch algorithm is not output-sensitive.[3] Of course, for Theorem 2.1 this does not matter.

Remark 2.9. The best time-bound for an algorithm that solves the maximum independent set problem seems to be Robson's algorithm.[4] This algorithm uses exponential space. Robson claims that he improved his algorithm such that, nowadays, it runs in $O^*(1.1844^n)$. This algorithm uses a computer-generated case analysis with thousands of different cases. Jian designed an algorithm for maximum independent set which runs in $O(1.2346^n)$.[5] This algorithm uses only polynomial space.

2.2 Chromatic number

Definition 2.10. *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

[3] C. Bron and J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, *Communications of the ACM* **16** (1973), pp. 575–577.

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

[5] T. Jian, An $O(2^{0.304n})$ algorithm for solving maximum independent set problem, *IEEE Transactions on Computers* **35** (1986), pp. 847–851.

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.

$$\chi(G) = \min \{ 1 + \chi(G - T) \mid T \text{ is a maximal independent set in } G \}. \quad (2.3)$$

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

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 (2.3) 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 2.11. *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 2.12. Notice that this algorithm uses exponential space.

Remark 2.13. Recently, some improvements were obtained by Björklund, *et al*.[6] They show that the chromatic number problem can be solved in $O^*(2^n)$. We look at their method in Section 2.7.

2.2.1 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.[7]

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.

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

Remark 2.15. Until now, the best algorithm for solving the 3-coloring problem is an algorithm by Eppstein.[8] This algorithm runs in $O^*(1.3289^n)$.

---

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

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

[8] 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.

2.3 Domatic partition

Definition 2.16. *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 2.17. 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 2.6 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.[9]

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 2.18. *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 2.19. A set $\{D_1, \ldots, D_k\}$ is a partition of a set 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) $\cup_{i=1}^{k} D_i = V$.

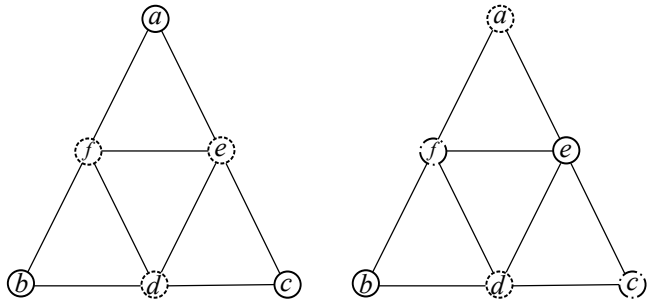


Fig. 2.2. This figure shows two domatic partitions; on the left $\{\{a, b, c\}, \{d, e, f\}\}$ and on the right $\{\{a, d\}, \{b, e\}, \{c, f\}\}$. The one on the right is a solution for the domatic partition problem.

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,

$$f(X) = \max \{ 1 + f(X \setminus Y) \mid Y \subseteq X \quad \text{and } Y \text{ is a dominating set in } G \}. \quad (2.4)$$

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

This simple observation gives us the following theorem.

Theorem 2.20. *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 2.21. Notice that the algorithm described above uses exponential space. The result can be improved to $O^*(2.8718^n)$ time and polynomial space.[10]

2.4 The traveling salesman problem

Suppose there are n cities, numbered $1, \ldots, n$. A traveling salesman has to visit the cities $1, \ldots, n$. 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 cities 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 $S \subseteq \{2, \ldots, n\}$ and for each $i \in S$ let $f(S, i)$ denote the length of a shortest tour 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

$$f(S, i) = \begin{cases} d(1, i) & \text{if } S = \{i\} \text{ and} \\ \min \{ f(S \setminus \{i\}, j) + d(j, i) \mid j \in S \setminus \{i\} \} & \text{otherwise.} \end{cases} \tag{2.5}$$

The following theorem was proved by Held and Karp, and, independently, by Bellman.[11] [12]

---

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

[11] R. Bellman, Dynamic programming treatment of the traveling salesman problem, *Journal of the Association for Computing Machinery* **9** (1962), pp. 61–63.

[12] M. Held and R. Karp, A dynamic programming approach to sequencing problems, *Journal of the Society for Industrial and Applied Mathematics* **10** (1962), pp. 196–210.

Theorem 2.22. *There exists an $O^*(2^n)$ algorithm which computes the minimum length of a traveling salesman tour.*

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 (2.5) in $O(n)$ time. Notice that the solution to the traveling salesman problem is given by the value of

$$\min \{ f(\{2, \ldots, n\}, j) + d(j, 1) \mid 2 \leqslant j \leqslant n \}. \tag{2.6}$$

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

2.5 Set cover

Definition 2.24. *Let U be a finite, nonempty set and let $n = |U|$. Let*

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

be a collection of nonempty 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 $f(U', i)$ for $i \geqslant 1$ be the minimal cardinality of a cover for U' with subsets from $\mathcal{S}_i$. Let $f(\varnothing, i) = 0$ and

$$\text{if} \quad U' \neq \varnothing \quad \text{and} \quad U' \not\subseteq \bigcup_{S \in \mathcal{S}_i} S \quad \text{then define} \quad f(U', i) = \infty.$$

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

$$f(U', 1) = \begin{cases} 1 & \text{if } U' \subseteq S_1 \\ \infty & \text{otherwise.} \end{cases} \tag{2.7}$$

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

$$f(U', i+1) = \begin{cases} \infty & \text{if } U' \not\subseteq \cup_{S \in \mathcal{S}_{i+1}} S \\ \min\{\, f(U', i), 1 + f(U' \setminus S_{i+1}, i) \,\} & \text{otherwise.} \end{cases} \tag{2.8}$$

Here is the proof that Formula (2.8) 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 elements from the set $\mathcal{S}_i$.

Theorem 2.25. *There exists an $O^*(m \cdot 2^n)$ 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 (2.7) and (2.8).

There are $O(m \cdot 2^n)$ pairs (U', i). First consider Formula (2.7). 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 (2.8). 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 $O(nm \cdot 2^n) = O^*(m \cdot 2^n)$.
This proves the theorem. □

2.6 Dominating set

Recall Definition 2.16. 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. Recall that we use $\gamma(G)$ to denote the cardinality of a minimum dominating set. Thus,

$$\gamma(G) = \min\{\, |D| \mid D \subseteq V(G) \quad \text{and} \quad \forall_{x \in V(G)}\ N[x] \cap D \neq \varnothing \,\}.$$

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

$$|D' \cup \mathcal{E}(D')| = |D'| + |\mathcal{E}(D')|. \tag{2.9}$$

Let $D' \subseteq W$ and define

$$I(D') = \{ s \in I \mid s \text{ has no neighbor in } D' \}. \tag{2.10}$$

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

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

$$U' = \{ x \in W \mid N[x] \cap (D' \cup I(D')) = \varnothing \}. \tag{2.11}$$

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

We now show how to compute minimum extensions.

Order the vertices of I, say

$$I = \{ s_1, \ldots, s_t \} \quad \text{and define, for } \ell \in \{1, \ldots, t\}, \quad I_\ell = \{ s_1, \ldots, s_\ell \}. \tag{2.12}$$

For each $U' \subseteq W$ and $\ell \in \{1, \ldots, t\}$, define

$$f(U', \ell) \subseteq I_\ell$$

as a subset of I_ℓ such that

(i) every vertex of U' has a neighbor in $f(U', \ell)$ and
(ii) $|f(U', \ell)|$ is minimal.

We obtain formulas similar to (2.7) and (2.8). When $U' = \varnothing$ then define $f(U', \ell) = \varnothing$ for all $\ell \in \{1, \ldots, t\}$.

Otherwise, when $U' \neq \varnothing$, first consider the case $\ell = 1$.

$$f(U', 1) = \begin{cases} \{x_1\} & \text{if } \varnothing \neq U' \subseteq N(s_1), \text{ and} \\ I & \text{otherwise.} \end{cases} \tag{2.13}$$

Assume that $U' \neq \varnothing$. Then we obtain the following recurrence relation for $f(U', \ell + 1)$.

$$f(U', \ell{+}1) = \begin{cases} f(U', \ell) & \text{if } |f(U', \ell)| < 1 + |f(U' \setminus N(s_{\ell+1}), \ell)| \\ f(U' \setminus N(s_{\ell+1}), \ell) \cup \{s_{\ell+1}\} & \text{otherwise.} \end{cases} \tag{2.14}$$

According to Theorem 2.25 the sets $f(U', \ell)$ for $U' \subseteq W$ and $\ell = 1, \ldots, t$ 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

$$|\mathcal{E}(D')| = |I(D') \cup f(U', t)| = |I(D')| + |f(U', t)|, \tag{2.15}$$

where $t = |I|$ and U' is defined by Formula (2.11).

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

$$2^{|W|} + \sum_{k=0}^{|W|} \sum_{\substack{D' \subseteq W \\ |D'|=k}} 1 = 2^{|W|} + \sum_{k=0}^{|W|} \binom{|W|}{k} = 2^{|W|+1}. \tag{2.16}$$

We now easily obtain the following theorem.

Theorem 2.26. *There exists an* $O^*(1.7088^n)$ *algorithm which solves the dominating set problem on graphs with* n *vertices.*

Proof. Let $\beta = 0.2271$. We consider two cases.

In the first case, assume that $|I| > \beta \cdot n$. Then, according to Formula (2.16), a minimum dominating set can be computed in

$$O^*(2^{(1-\beta)n}) = O^*(1.7088^n). \tag{2.17}$$

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

By Stirling's formula one can obtain that, since $0 < \beta < \frac{1}{2}$,

$$\sum_{i=0}^{\lfloor \beta n \rfloor} \binom{n}{i} \leqslant 2^{h(\beta)n} \quad \text{where} \quad h(\beta) = -\beta \log_2 \beta - (1-\beta)\log_2(1-\beta). \tag{2.18}$$

The function $h(\beta)$ is the binary entropy function. (See Figure 2.3.) Some calculations show that $2^{h(\beta)n} \leqslant 1.7088^n$.

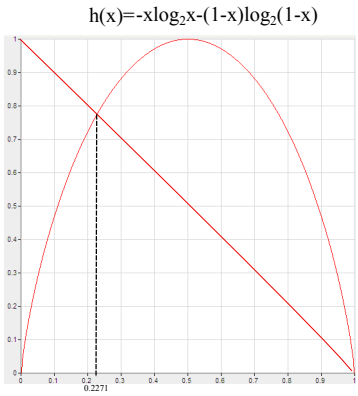


Fig. 2.3. This figure shows the binary entropy function; the value $\beta \approx 0.2270922$ is where $h(\beta) = 1 - \beta$.

This proves the theorem. □

Remark 2.27. Notice that we chose β such that $h(\beta) = 1 - \beta$ to obtain the best bound (see Figure 2.3). If the maximal independent set I is small then the domination number $\gamma(G) \leqslant |I|$ is small as well. In that case it is better to try all subsets of cardinality at most $|I|$. This gives us the bound in terms of the entropy function (2.18). When $|I|$ is large, then the 'universe' $W = V \setminus I$ is small. Then it is better to use the set cover method. By the choice of β we get the same timebound in both cases.

2.6.1 Dominating set in bipartite graphs

Theorem 2.26 produces a nice timebound for the dominating set problem on bipartite graphs.[13]

First recall the definition of a bipartite graph.

Definition 2.28. *A graph* G *is bipartite if* $\chi(G) \leqslant 2$.

Theorem 2.29. *There exists an* $O^*(2^{n/2})$ *algorithm that computes a minimum dominating set on bipartite graphs with* n *vertices.*

Proof. Let $G = (V, E)$ be a bipartite graphs with n vertices. We claim that $\alpha(G) \geqslant \frac{n}{2}$, where $\alpha(G)$ is the maximal cardinality of an independent set in G. To see this, consider a coloring with at most two colors. A color class is a subset of the vertices that have the same color. The coloring partitions the vertices into at most two color classes, and so at least one color class has at least $\frac{n}{2}$ vertices. In Exercise 2.4 we ask you to design a linear-time algorithm to recognize bipartite graphs. This recognition algorithm produces an independent set of size at least $\frac{n}{2}$ vertices in linear time. (Notice that it does *not* compute $\alpha(G)$ in linear time! The best way to compute $\alpha(G)$ in bipartite graphs is via Edmonds' matching algorithm.[14])

Our algorithm now runs the algorithm as described in the first case in the proof of Theorem 2.26. We can take $\beta = \frac{1}{2}$ in this case because the independent set has cardinality at least $\frac{n}{2}$. According to Formula (2.16) on Page 19, a minimum dominating set can be computed in

$$O^*(2^{(1-\beta)n}) = O^*(2^{n/2}) = O^*(1.4143^n).$$

This proves the theorem. □

2.7 Inclusion – exclusion

Good timebounds for many set covering problems can be obtained via the method of inclusion-exclusion. The method was developed by Björklund and Husfeldt and, independently, by Koivisto.[15] We illustrate their method by graph coloring.

We start with the inclusion-exclusion formula.

---

[13] M. Liedloff, Finding a dominating set on bipartite graphs, *Information Processing Letters* **107** (2008), pp. 154–157.

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

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

Lemma 2.30. *Let*

$$\mathcal{S} = \{ A_1,\ A_2,\ \ldots,\ A_t \} \tag{2.19}$$

be a collection of subsets of some finite set U. *Then*

$$|\bigcup_{i=1}^{t} A_i| = \sum_{k=1}^{t} (-1)^{k-1} \sum_{1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant t} |A_{i_1} \cap \cdots \cap A_{i_k}|. \tag{2.20}$$

Proof. Let $A = \cup_{i=1}^{t} A_i$.

Define the characteristic functions $f : U \to \{0,1\}$ and $f_i : U \to \{0,1\}$, for $i \in \{1,\ldots,t\}$, as follows.

$$f(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \in U \setminus A, \end{cases} \quad \text{and} \quad \forall_{i \in \{1,\ldots,t\}}\ f_i(x) = \begin{cases} 1 & \text{if } x \in A_i, \text{ and} \\ 0 & \text{if } x \in U \setminus A_i. \end{cases} \tag{2.21}$$

Then we have the following identity.

$$\forall_{x \in U}\ \prod_{i=1}^{t} (f(x) - f_i(x)) = 0. \tag{2.22}$$

To see this, first assume that $x \in A$. Then $x \in A_i$ for some $i \in \{1,\ldots,t\}$. Then the i^{th} term in the product $f(x) - f_i(x) = 1 - 1 = 0$ and so the whole product is zero.
Now assume that $x \notin A$. Then $x \notin A_i$ for any $i \in \{1,\ldots,t\}$, and so every term in the product $f(x) - f_i(x) = 0$.

Write this product as follows.

$$\sum_{k=0}^{t} \sum_{1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant t} (-1)^k \cdot f(x)^{t-k} \cdot f_{i_1}(x) \cdot f_{i_2}(x) \cdots f_{i_k}(x) = 0. \tag{2.23}$$

This implies

$$f(x)^t = \sum_{k=1}^{t} (-1)^{k-1} \sum_{1 \leqslant i_1 < \cdots < i_k \leqslant t} f_{i_1}(x) \cdots f_{i_k}(x). \tag{2.24}$$

Sum Equation (2.24) over all elements $x \in U$. This gives

$$\sum_{x \in U} f(x)^t = |A| = \sum_{k=1}^{t} \sum_{1 \leqslant i_1 < \cdots < i_k \leqslant t} (-1)^{k-1} |A_{i_1} \cap \cdots \cap A_{i_k}|. \tag{2.25}$$

This proves the lemma. □

A similar formula can be obtained for the number of elements that lie in none of the A_i. We give a different proof.

Lemma 2.31. *Let $\mathcal{S} = \{A_1, \ldots, A_t\}$ be a collection of subsets of a finite set U. Use the convention that*

$$\boxed{\cap_{i \in \varnothing} A_i = U.} \tag{2.26}$$

Then

$$|\bigcap_{i=1}^{t} \overline{A_i}| = \sum_{I \subseteq \{1,\ldots,t\}} (-1)^{|I|} \, |\cap_{i \in I} A_i|. \tag{2.27}$$

Proof. Give each element $x \in U$ a weight, say $w(x)$, and sum the weights over the sets on the left and right hand side of (2.27). Consider $x \in U$. If x lies in none of the A_i then it contributes $w(x)$ on the left hand side and it also contributes $w(x)$ on the right hand side, namely for $I = \varnothing$ (by the convention).

Now assume that x lies in A_i for $i \in I$, and assume that $I \neq \varnothing$. Then it is counted zero times on the left. On the right hand side it is counted for all $I' \subseteq I$. Its total contribution is zero because, by the binomial theorem,

$$\sum_{I' \subseteq I} (-1)^{|I'|} \, w(x) = w(x) \cdot \sum_{k=0}^{|I|} \binom{|I|}{k} (-1)^k = 0 \quad \text{because } I \neq \varnothing. \tag{2.28}$$

□

Definition 2.32. *Let $G = (V, E)$ be a graph and let $\mathcal{S}$ be some collection of subsets of V. A k-cover is a k-tuple over $\mathcal{S}$*

$$(S_1, \ldots, S_k) \quad \textit{such that} \quad \begin{cases} \textit{each } S_i \in \mathcal{S}, \textit{ and} \\ \cup_{i=1}^{k} S_i = V. \end{cases} \tag{2.29}$$

Notice that some of the sets in a k-tuple may be equal.

$$\boxed{\text{Denote the number of } k\text{-covers by } c_k.}$$

Observe the following. When $\mathcal{S}$ is the set of all independent sets in G then

$$c_k > 0 \quad \text{if and only if} \quad \chi(G) \leqslant k.$$

Definition 2.33. *For $X \subseteq V$, define $\alpha(X)$ as the number of sets in $\mathcal{S}$ that avoid X, that is,*

$$\boxed{\alpha(X) = |\{ S \in \mathcal{S} \mid S \cap X = \varnothing \}|.} \tag{2.30}$$

Lemma 2.34. *Let $G = (V, E)$ be a graph and let $\mathcal{S}$ be a collection of subsets of V. Let c_k be the number of k-covers. For $X \subseteq V$, let $\alpha(X)$ be the number of sets in $\mathcal{S}$ that avoid X. Then*

$$\boxed{c_k = \sum_{X \subseteq V} (-1)^{|X|}\, \alpha(X)^k.} \tag{2.31}$$

Proof. Let $V = \{1, \ldots, n\}$. We use Lemma 2.31.

For the universe U we take the set of all k-tuples,

$$U = \{ (S_1, \ldots, S_k) \mid S_i \in \mathcal{S} \}. \tag{2.32}$$

For the subset $A_i \subseteq U$, $i \in V$, we take the set of k-tuples that avoid $\{i\}$, that is,

$$A_i = \{ (S_1, \ldots, S_k) \mid i \notin \bigcup_{\ell=1}^{k} S_\ell \}. \tag{2.33}$$

The number of k-covers is exactly the number of k-tuples that lie in none of the A_i. Thus c_k is the left hand side of (2.27). For $X \subseteq V$, $\cap_{i \in X} A_i$ contains those k-tuples that avoid all elements of X. This number is of course $\alpha(X)^k$. Thus

$$|\bigcap_{i \in X} A_i| = \alpha(X)^k. \tag{2.34}$$

This proves the lemma. □

In the following theorem we illustrate the inclusion – exclusion method.

Theorem 2.35. *Let $G = (V, E)$ be a graph with n vertices. There exists an $O^*(2^n)$ algorithm that computes $\chi(G)$.*

Proof. Let $G = (V, E)$ be a graph with n vertices. Let $V = \{1, \ldots, n\}$. Let $\mathcal{S}$ be the collection of all nonempty, independent sets in G, not necessarily maximal.

By Formula (2.31) it is sufficient to compute $\alpha(X)$, for all subsets $X \subseteq V$.

Of course, we have that $\alpha(V) = 0$. Assume that $X \neq V$. Fix some $v \in V \setminus X$. We claim that

$$a(X) = a(X \cup \{v\}) + a(X \cup N[v]) + 1. \tag{2.35}$$

To prove (2.35) we partition the sets $S \in \mathcal{S}$ that avoid X into two types, the sets S that contain v and the sets S that do not contain v.

First consider sets S that avoid X and that do not contain v. Those sets S are counted in $a(X \cup \{v\})$.

Consider sets S that avoid X and that contain v. Since S is an independent set which contains v, S contains no neighbors of v. Thus

$$S \setminus \{v\} \cap (X \cup N[v]) = \varnothing. \tag{2.36}$$

Either $S = \{v\}$, which accounts for the $+1$-term, or $S \setminus \{v\} \neq \varnothing$. By (2.36), the number of sets S with $S \setminus \{v\} \neq \varnothing$ is exactly $a(X \cup N[v])$. This proves (2.35).

The recurrence relation (2.35) shows that the numbers $a(X)$ can be computed in $O(n2^n) = O^*(2^n)$ time. This proves the theorem. □

Remark 2.36. The algorithm of Theorem 2.35 uses exponential space. The paper of Björklund, *et al.*, shows that the chromatic number of a graph can be computed in $O^*(2.2461^n)$ time and polynomial space.

2.7.1 Triangle partition

As an example of the inclusion-exclusion method, let's have a look at the triangle partition problem. Let $G = (V, E)$ be a graph with n vertices. A triangle in G is a clique with three vertices. A triangle partition of G is a collection of vertex-disjoint triangles that partition V. Obviously, G can have a triangle partition only if

$$\frac{n}{3} \quad \text{is integer.}$$

The triangle partition problem is NP-complete (even for graphs with maximal degree four).[16]

Theorem 2.37. *There exists an $O^*(2^n)$ time and polynomial space algorithm that checks if G has a triangle partition.*

[16] J. van Rooij, M. van Kooten Niekerk and H. Bodlaender, Partitioning sparse graphs into triangles – relations to exact satisfiability and very fast exponential time algorithms. Technical Report UU-CS-2010-005, Utrecht University, 2010.

Proof. We use the inclusion-exclusion method.

Let G be a graph with n vertices. Let $\mathcal{S}$ be the set of all triangles in G. Then, obviously,

$$|\mathcal{S}| = O(n^3). \tag{2.37}$$

A k-cover is a k-tuple of $\mathcal{S}$:

$$(S_1, \ldots, S_k) \quad \text{such that} \quad \begin{cases} \text{each } S_i \text{ is a triangle in } G, \text{ and} \\ \cup_{i=1}^{k} S_i = V. \end{cases} \tag{2.38}$$

Let c_k be the number of k-covers.

We claim that G has a triangle partition if and only if

$$q = \frac{n}{3} \quad \text{is integer, and} \quad c_q > 0. \tag{2.39}$$

To see this, notice that G has a q-cover if and only if G has a triangle partition.

By Lemma 2.34 on page 24

$$c_k = \sum_{X \subseteq V} (-1)^{|X|} a(X)^k, \tag{2.40}$$

where $a(X)$ is the number of triangles that avoid X, that is, the number of triangles contained in $V \setminus X$.

Notice that we can easily count the number of triangles that are contained in $V \setminus X$ in polynomial time. Thus the numbers $a(X)$ can be computed in $O^*(2^n)$ time and polynomial space.

To compute $a(X)^q$ we need $O(\log q)$ multiplications and we can represent this number in $O(q \log n)$ bits. This shows that we can compute the Formula (2.40) in $O^*(2^n)$ time and polynomial space. □

Remark 2.38. There is an $O^*(1.7693^n)$ algorithm that solves the triangle partition problem.[17]

[17] M. Koivisto, Partitioning into sets of bounded cardinality, *Proceedings of the* 4th *International Workshop on Parameterized and Exact Computations, IWPEC 2009*, Springer-Verlag, LNCS 5917 (2009), pp. 258–263.

2.7.2 An improved algorithm for triangle partition

Consider the problem of partitioning the vertices of a graph $G = (V, E)$ into triangles. Koivisto's algorithm, mentioned in the previous section, reaches a timebound of $O^*(1.7693^n)$ for partitioning V into subsets of cardinality at most three. In this section we illustrate Koivisto's method for the triangle partition problem. The improvement follows from the idea to consider triangle partitions of which the elements are in lexicographic order.

Write $q = \frac{n}{3}$ and let $V = \{1, \ldots, n\}$. Order the triangles of G into lexicographic order. Then the first triangle in any partition must contain the first vertex, which is 1. In general, we have the following lemma.

Lemma 2.39. *Consider the lexicograph ordering of the triangles in a triangle partition $\mathcal{P}$ of G. Then, for $j \in \{1, \ldots, q\}$, the first j triangles in $\mathcal{P}$ must contain the vertices of $\{1, \ldots, j\}$.*

Proof. Since the triangles of $\mathcal{P}$ partition V each element of V is in some triangle of $\mathcal{P}$. Since the triangles are lexicographically ordered, the first j triangles of $\mathcal{P}$ must contain all vertices of $\{1, \ldots, j\}$. □

Let $\mathcal{F}$ be the set of all triangles in G. Define

$$\mathcal{R}_1 = \{\, T \mid T \in \mathcal{F} \quad \text{and} \quad 1 \in T \,\},$$

and, recursively for $k > 1$,

$$\mathcal{R}_k = \{\, Y \cup X \mid Y \in \mathcal{R}_{k-1},\ X \in \mathcal{F},\ Y \cap X = \varnothing \text{ and } \min\{\, i \mid i \in V \setminus Y \,\} \in X \,\}.$$

Lemma 2.40. *Let $\mathcal{P} = (T_1, \ldots, T_q)$ be a triangle partition of G, where the triangles of $\mathcal{P}$ are in lexicographic order. Then, for $j \in \{1, \ldots, q\}$,*

$$\bigcup_{k=1}^{j} T_k \in \mathcal{R}_j.$$

Proof. The first triangle of $\mathcal{P}$ must contain the vertex 1. Thus $T_1 \in \mathcal{R}_1$. We proceed by induction. Let

$$Y = \bigcup_{\ell=1}^{j-1} T_\ell.$$

Then the j^{th} triangle must contain the smallest element of $V \setminus Y$. Thus

$$\bigcup_{\ell=1}^{j} T_\ell \in \mathcal{R}_j.$$

□

Lemma 2.41.

$$\sum_{k=1}^{q} |\mathcal{R}_k| = O^*(1.7549^n).$$

Proof. By Lemma 2.39, the first j triangles must contain $\{1,\ldots,j\}$. The rest of the vertices of these j triangles are in $\{j+1,\ldots,n\}$. Thus,

$$\sum_{k=1}^{q} |\mathcal{R}_k| \leqslant \sum_{k=1}^{q} \binom{n-k}{2k}.$$

The binomial coefficients can be bounded as follows. Write

$$k = \alpha n \quad \text{and} \quad \beta = \frac{2\alpha}{1-\alpha}. \tag{2.41}$$

First consider the tails. When $\beta \leqslant \frac{1}{3}$ then $\alpha \leqslant \frac{1}{7}$ and we have

$$\binom{n-k}{2k} \leqslant 2^{2n/7} \leqslant 1.22^n.$$

When $\beta \geqslant \frac{2}{3}$ then $\alpha \geqslant \frac{1}{4}$ and we find

$$\binom{n-k}{2k} = \binom{n-k}{n-3k} \leqslant \binom{n}{n-3k} \leqslant 2^{n/4} \leqslant 1.2^n.$$

Consider the case where $\frac{1}{3} < \beta < \frac{2}{3}$, that is, $\frac{1}{7} < \alpha < \frac{1}{4}$. Write $\beta = \frac{1+\epsilon}{2}$. Then $-\frac{1}{3} < \epsilon < \frac{1}{3}$. With Stirling's formula we obtain (neglecting polynomial factors)

$$\binom{n-k}{2k} \sim \exp\left[\frac{2n}{5+\epsilon}\big(\, 2\ln(2) - (1-\epsilon)\ln(1-\epsilon) - (1+\epsilon)\ln(1+\epsilon)\,\big)\right]. \tag{2.42}$$

Now write $\gamma = \frac{1-\epsilon}{2}$. Then $\frac{1}{3} < \gamma < \frac{2}{3}$ and (2.42) becomes

$$\binom{n-k}{2k} \sim \exp\left[\frac{2n}{3-\gamma} f(\gamma)\right] \quad \text{where} \quad \gamma = \frac{1-3\alpha}{1-\alpha} \quad \text{and} \quad \alpha = \frac{k}{n} \tag{2.43}$$

and where $f(\gamma) = -\gamma\ln(\gamma) - (1-\gamma)\ln(1-\gamma)$ is the entropy function. (See Figure 2.4.)

Define

$$g(\gamma) = \frac{1}{3-\gamma} \cdot f(\gamma). \tag{2.44}$$

The function $g(\gamma)$ has its maximum at $\gamma \leqslant 0.56985$ and with this value (2.43) becomes

$$\binom{n-k}{2k} = O^*(1.7549^n). \tag{2.45}$$

□

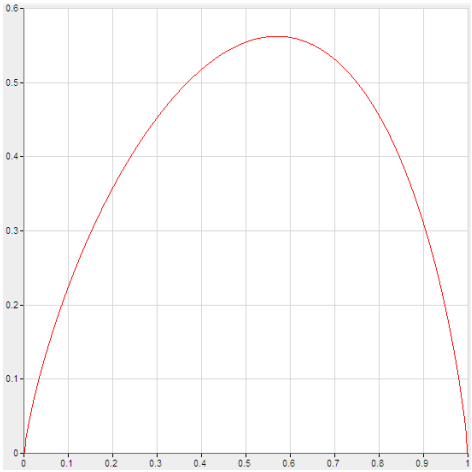


Fig. 2.4. This figure shows the function $\frac{2}{3-x}f(x)$ where $f(x)$ is the entropy function.

We show that there is a dynamic programming algorithm which runs in time proportional to

$$\sum_{k=1}^{q} |\mathcal{R}_k|.$$

Theorem 2.42. *There exists an* $O^*(1.7549^n)$ *algorithm that checks if a graph* G *with* n *vertices has a triangle partition.*

Proof. Define, for $S \subseteq V$,

$$f_1(S) = \begin{cases} 1 & \text{if } S \in \mathcal{R}_1, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

For $j > 1$, define

$$f_j(S) = \sum_{Y \subseteq S} f_{j-1}(Y) \cdot [\, S \setminus Y \in \mathcal{F} \,] \cdot [\, \min \{\, x \mid x \in V \setminus Y \,\} \in S \setminus Y \,]$$

where we write

$$[\, P \,] = \begin{cases} 1 & \text{if } P \text{ is true, and} \\ 0 & \text{otherwise.} \end{cases}$$

Thus $f_k(S)$ counts the number of ordered partitions of S into k triangles. Notice that

$$f_k(S) = 0 \quad \text{if} \quad S \notin \mathcal{R}_k.$$

By Lemma 2.41, the time to compute $f_k(S)$ is therefore proportional to

$$\sum_{k=1}^{q} |\mathcal{R}_k| \cdot n^3 = O^*(1.7549^n).$$

□

Remark 2.43. The $O^*(2^n)$ algorithm of section 2.7.1 on page 25 uses polynomial space. The algorithm of Theorem 2.42 uses exponential space.

Remark 2.44. For the partitioning into general (constant) subsets of size r this method gives

$$\binom{n-k}{(r-1)k} \sim \exp\left[\frac{n(r-1)}{r-x} \cdot f(x)\right] \quad \text{where } x = \frac{1-rk/n}{1-k/n} \tag{2.46}$$

and where $f(x) = -x\ln(x) - (1-x)\ln(1-x)$. To maximize one would need to solve

$$(1-x)^{r-1} = x^r.$$

When we write $x = \frac{1+\epsilon}{2}$ we get a first approximation

$$\epsilon \approx \frac{1}{3r-2} \quad \text{and} \quad x \approx \frac{3r-1}{2(3r-2)}.$$

For $r = 3$ this gives $x \approx 4/7 \approx 0.571$ while the actual value, which we found in the proof of Lemma 2.41, is $x \approx 0.5699$.

For large r, $x \approx 1/2$, $f(x) \approx \ln(2)$ and

$$\binom{n-k}{(r-1)k} \sim 2^{2n(r-1)/(2r-1)} = 2^{n(1-\frac{1}{2r})}(1 - \frac{n\ln(2)}{4r^2} + O(\frac{n^2}{r^3})). \tag{2.47}$$

2.8 Subset sum

Let's do an easy one to finish this chapter off.

Let $a_1, \ldots, a_n$ 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 $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = K$.

The subset sum problem is NP-complete.

Theorem 2.45. *There exists an* $O(K \cdot n)$ *algorithm which solves the subset sum problem.*

Proof. Define

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

For $i = 1, \ldots, n$, 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 the sums that can be made with numbers from $\{a_1, \ldots, a_i\}$ and which are at most K.

First consider $i = 1$. Then,

$$\Omega_1 = \begin{cases} \{\ 0,\ a_1\ \} & \text{if } a_1 \leqslant K, \text{ and} \\ \{\ 0\ \} & \text{if } a_1 > K. \end{cases} \tag{2.48}$$

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

$$\Omega_{i+1} = \Omega_i \cup \{\ 0 \leqslant t \leqslant K \mid \text{there exists a } t' \in \Omega_i \text{ such that } t' + a_{i+1} = t\ \}. \tag{2.49}$$

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

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

It is easy to see that Ω_n can be computed via Formulas (2.48) and (2.49) in $O(K \cdot n)$ time. The answer to the subset problem is YES if $K \in \Omega_n$, and NO otherwise. □

2.9 Problems

2.1. 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 no remaining vertex has any neighbor, that is, until the graph is an independent set. Show that this changes Formula (2.2) on Page 9 into

$$T(n) \leqslant T(n-1) + T(n-2) + O(n+m). \tag{2.50}$$

Show that the solution of Formula (2.50) is the n^{th} Fibonacci number times some polynomial. Use the exact formula for the Fibonacci numbers to show that this gives

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

Now assume that we branch until every vertex has degree at most one. This changes Formula (2.50) into

$$T(n) \leqslant T(n-1) + T(n-3) + O(n+m). \tag{2.51}$$

Show that Formula (2.51) gives a characteristic equation

$$\omega^3 = \omega^2 + 1. \tag{2.52}$$

This gives $T(n) = O^*(1.4656^n)$.

2.2. 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.

2.3. 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 $T(n-1) > T(n)$.

2.4. Design a linear-time algorithm which checks if a graph is bipartite.

2.5. Design an exact algorithm for 4-coloring.
`Hint:` The 4-coloring problem can be solved in $O^*(1.7504^n)$.[18]

[18] J. M. Byskov, Enumerating maximal independent sets with applications to graph colouring, *Operations Research Letters* **32** (2004), pp. 547–556.

2.6. Check Formula (2.4) on Page 14.

2.7. Check Formula (2.5) on Page 15.

2.8. 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

$$d(i,j) = \begin{cases} 1 & \text{if } \{i,j\} \in E, \text{ and} \\ \infty & \text{if } \{i,j\} \notin E. \end{cases} \tag{2.53}$$

(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.

2.9. Check the Formulas (2.13) and (2.14) and show that the sets $f(X, \ell)$, as defined by these formulas, can be computed in $O^*(2^{|W|})$.

2.10. Design an $O^*(2^n)$ algorithm that solves the domatic partition problem. `Hint:` Use the inclusion-exclusion method described in Section 2.7. You need to derive a recurrence relation similar to (2.35) on Page 25.

2.11. Design an $O^*(2^n)$ algorithm that solves the triangle packing problem. The triangle packing problem asks for the maximal number of vertex-disjoint triangles in a graph.
`Hint:` Use the inclusion-exclusion method as in Section 2.7.1. For the set S take the set of all triangles and all subsets that contain one vertex. Show that there is a triangle packing with k triangles if and only if $c_{n-2k} > 0$.

2.12. The subset sum problem described in Section 2.8 is NP-complete. Theorem 2.45 on Page 31 shows that it can be solved in $O(K \cdot n)$ time. Does this prove that $P = NP$?

3

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

$$\mathcal{G} = \{ \, G \mid G \text{ is a planar graph} \, \} \tag{3.1}$$

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,[1] 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.

3.1 Perfect graphs

Definition 3.1. *A graph* G *is perfect if for every induced subgraph* H *of* G

$$\boxed{\chi(H) = \omega(H),} \tag{3.2}$$

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

$$\chi(G) \geqslant \omega(G). \tag{3.3}$$

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 (3.3) does not hold for G.

[1] S. Williamson, Depth-first search and Kuratowski subgraphs, *Journal of the Association for Computing Machinery* **31** (1984), pp. 681–693.

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 maximum independent set for G' consists of an independent set in G plus one vertex in the clique that is added to G. Thus the independent set problem for G' is just as hard as it is for G. As we will see, we get a whole new ballgame when we require that

$$\boxed{\chi = \omega \quad \text{for every induced subgraph.}}$$

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.[2][3]

Theorem 3.2 (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

$$\boxed{\alpha(H) = \kappa(H),} \tag{3.4}$$

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 3.3 we ask you to prove that bipartite graphs are perfect. By Theorem 3.2 also the complements of bipartite graphs are perfect. (Can you prove this without using Theorem 3.2?)

Another important example of a class 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.

Coloring a linegraph $L(G)$ of a graph G is equivalent to coloring the edges of G such that any two edges that share an endpoint get different colors. By Vizing's theorem, for every graph G, $\chi(L(G))$ is either Δ or $\Delta + 1$, where Δ is the largest degree in G.[4] Kőnig proved that bipartite graphs fall into the first

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

[3] G. Gasparian, Minimal imperfect graphs: a simple approach, *Combinatorica* **16** (1996), pp. 209–216.

[4] V. Vizing, On an estimate of the chromatic class of a p-graph, *Diskret. Analiz* **3** (1964), pp. 25–30.

class, that is[5]

$$\text{when } G \text{ is bipartite:} \quad \chi(L(G)) = \Delta(G) = \omega(L(G)).$$

In the following lemma we prove (3.4) for linegraphs of bipartite graphs.

Lemma 3.3. *Let* G *be bipartite. Then* $L(G)$ *is perfect.*

Proof. First of all, it is sufficient to prove Equation (3.4) 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:[6]

$$\boxed{\text{If } G \text{ is bipartite then } \nu(G) = \tau(G).}$$

Here, $\tau(G)$ 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 $C = \{x_1, \ldots, x_s\}$ 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 $\{L_1, \ldots, L_s\}$ 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 (3.3), $\alpha(H) \leqslant \kappa(H)$ for any graph H.

Thus $\alpha(L(G)) = \kappa(L(G))$ and by Theorem 3.2 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 3.2, when G is perfect it cannot have the complement of an odd cycle of length at least five as an induced subgraph. Claude Berge conjectured in 1961 that this characterizes perfect

[5] D. Kőnig, Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre, *Mathematische Annalen* **77** (1916), pp. 453–465.

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

graphs.[7] The proof of the conjecture sent a shock wave through the graph theory community.[8]

Definition 3.4. *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.*

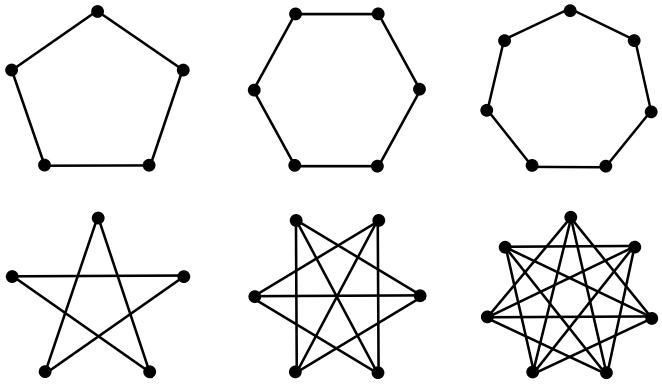


Fig. 3.1. The figure shows some holes and antiholes. Notice that the complement of C_5 is C_5. The complement of C_6 is planar but the complement of C_7 is not (and also not the complements of longer cycles).

Theorem 3.5 (The strong perfect graph theorem). *A graph is perfect if and only if it has no odd hole and no odd antihole.*

Theorem 3.5 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.

[7] 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.
[8] 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.[9][10]

Theorem 3.6. *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.[11]

Theorem 3.7. *There exists a polynomial-time algorithm that computes* $\omega(G)$ *and* $\chi(G)$ *for graphs* G *which 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

$$\omega(\bar{G}) = \alpha(G) \leqslant \Theta(G) \leqslant \vartheta(G) \leqslant \kappa(G) = \chi(\bar{G}). \tag{3.5}$$

The theorem above was proved by showing that the Lovász number $\vartheta(G)$ of a graph G is computable in polynomial time.[12][13]

3.2 Comparability graphs

Perhaps the best known class of perfect graphs is the class of comparability graphs.

Definition 3.8. *Let* $G = (V, E)$ *be a graph. An orientation of* G *gives each edge* $\{x, y\}$ *a direction, either from* x *to* y *or from* y *to* x.

Thus, in an orientation of G each edge $\{x, y\}$ is directed in exactly one way, either as $\overrightarrow{xy}$ or as $\overleftarrow{xy}$.

---

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

[10] P. Seymour, How the proof of the strong perfect graph conjecture was found, *Gazette des Mathématiciens* **109** (2006), pp. 69–83.

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

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

[13] M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* **1** (1981), pp. 169–197.

Definition 3.9. *An orientation of a graph* G *is transitive if for every three vertices* x, y *and* z, *if there are directed edges* $\overrightarrow{xy}$ *and* $\overrightarrow{yz}$ *then the pair* $\{x, z\}$ *is an edge in* G *and in the orientation it is directed as* $\overrightarrow{xz}$.

Remark 3.10. A graph $G = (V, E)$ equipped with a transitive orientation is closely related to a partially ordered set, or a poset. A partially ordered set is a pair $(V, \preceq)$ where V is a set and $\preceq$ is a binary relation. The binary relation $\preceq$ is reflexive, antisymmetric and transitive. That means

(i) $x \preceq x$ for any element $x \in V$. This is reflexivity.
(ii) When $x \preceq y$ and $y \preceq x$ then $x = y$. This is antisymmetry.
(iii) When $x \preceq y$ and $y \preceq z$ then $x \preceq z$. This is transitivity.

(So, the only 'difference' with the directed graph is the reflexivity.) When every pair of elements of V is ordered by $\preceq$ then $(V, \preceq)$ is a total order (or linear order). An example of a total order is the set of natural numbers, or the real numbers, ordered by $\leqslant$. In the case where $(V, \preceq)$ is a total order and $|V|$ is finite there is exactly one minimal element and one maximal element in V. When $(V, \preceq)$ is a total order and if $|V|$ is finite, then the corresponding comparability graph is a clique (every pair of elements is comparable; some people like to add a 'loop' at every vertex to have also the reflexivity).

Of course, when $G = (V, E)$ is equipped with a transitive orientation, then the directed graph is acyclic. Vertices with zero indegree are sometimes called 'sources' and vertices with zero outdegree, 'sinks.'

Obviously, not every graph can be oriented transitively. For example, consider an odd cycle C of length at least five. In any orientation of C there will be two edges $\{x, y\}$ and $\{y, z\}$ that are oriented in the same direction, that is, either $\overrightarrow{xy}$ and $\overrightarrow{yz}$ or $\overleftarrow{xy}$ and $\overleftarrow{yz}$. If the orientation were transitive then in both cases we would have an edge $\{x, z\}$, but since C has length at least 5, $\{x, z\}$ is not an edge of C.

Definition 3.11. *A graph is a comparability graph if it has a transitive orientation.*

Remark 3.12. Computing a transitive orientation can be done in linear time.[14] However, in order to check that the orientation, which is obtained by this algorithm, is indeed transitive one needs to do a matrix multiplication. Thus the total time for comparability graph recognition takes $O(n^{2.3727\cdots})$, using

[14] M. Tedder, D. Corneil, M. Habib and C. Paul, Simple, linear-time modular decomposition. ArXiv: 0710.3901v2, 2008.

Williams' algorithm for fast matrix multiplication.[15] Alternatively, Golumbic designed an easy $O(n^3)$ algorithm.[16]

Let G be a graph and consider a transitive orientation of G.

Definition 3.13. *A chain is a directed path* $[x_1, \ldots, x_k]$ *such that*

$$\overrightarrow{x_i x_{i+1}} \quad \textit{is a directed edge for all } i \in \{1, \ldots, k-1\}.$$

Notice that, when $[x_1, \ldots, x_k]$ is a chain in a transitive orientation of a graph G, then $\{x_1, \ldots, x_k\}$ is a clique in G. For example, $\overrightarrow{x_1x_2}$ and $\overrightarrow{x_2x_3}$ imply that $\{x_1, x_3\}$ is an edge in G. By induction it easily follows that x_1 is adjacent to all other vertices in the chain, and the same holds for all other vertices.

The converse holds as well.

Lemma 3.14. *Let* G *be a graph and assume that* G *has a transitive orientation. Then every clique in* G *corresponds with a chain in any transitive orientation of the graph* G.

Proof. Let $\Omega = \{x_1, \ldots, x_k\}$ be a clique in G. Every edge $\{x_i, x_j\}$ has a direction, either $\overrightarrow{x_ix_j}$ or $\overleftarrow{x_ix_j}$. By transitivity there can be no directed cycles, not even triangles.

Consider a vertex x_i in Ω with a minimal number of incoming edges. Assume that it has an incoming edge $\overrightarrow{x_jx_i}$. Then we obtain a contradiction as follows. Each incoming edge $\overrightarrow{x_kx_j}$ yields an incoming edge $\overrightarrow{x_kx_i}$ and so the number of incoming edges into x_i is at least one more than the number of incoming edges into x_j.

So there exists a vertex in Ω, say x_1, without any incoming edge. Notice that x_1 is unique, since any other vertex $x_j \in \Omega$ now has at least one incoming edge, namely $\overrightarrow{x_1x_j}$.

By induction $\Omega \setminus \{x_1\}$ forms a chain in the orientation of G; say $[x_2, \ldots, x_k]$. Then $[x_1, \ldots, x_k]$ is also a chain in that orientation of G. □

Thus, if G is a graph with a transitive orientation then $\omega(G)$ is the number of vertices in the longest chain. We want to prove that any comparability graph is perfect. In order to do that, we need the concept of an antichain.

Definition 3.15. *Let* G *be a graph with a transitive orientation. An antichain is a set of vertices which is an independent set in* G.

[15] V. Williams, Breaking the Coppersmith-Winograd Barrier. Manuscript 2011.

[16] M. Golumbic, The complexity of comparability graph recognition and coloring, *Computing* **18** (1977), pp. 199–208.

Theorem 3.16. *Comparability graphs are perfect.*

Proof. When G is a comparability graph then every induced subgraph of G is also a comparability graph. Therefore, it is sufficient to prove that

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

Consider a transitive orientation of G. For each vertex x let $\Delta(x)$ be the maximal number of vertices in a chain that ends in x. Now define

$$N_i = \{\, x \mid \Delta(x) = i \,\}.$$

Notice that $N_1 \neq \varnothing$, since there is a vertex without any incoming edge. This follows, in the same manner as in the proof of Lemma 3.14, from the fact that there are no directed cycles.

Also notice that, for any $i \geqslant 1$,

$$N_{i+1} \neq \varnothing \quad \Rightarrow \quad N_i \neq \varnothing.$$

Let $\omega \in \mathbb{N}$ be the maximal number such that $N_\omega \neq \varnothing$. Then, by Lemma 3.14, $\omega = \omega(G)$. The sets $N_1, \ldots, N_\omega$ form a partition of G into independent sets. Thus $\chi(G) \leqslant \omega(G)$.

This proves the theorem because, of course, $\chi(G) \geqslant \omega(G)$ for any graph (this is (3.3) on Page 36). □

As a corollary of Theorem 3.16 we obtain Dilworth's theorem.[17] This theorem is usually formulated for posets. The theorem can be proved in many ways. For example, it can be seen that it is equivalent to the König-Egerváry theorem. We obtain an easy proof via the perfect graph theorem.

Theorem 3.17 (Dilworth's theorem). *Let $G = (V, E)$ be a graph with a transitive orientation. There exists an antichain A and a partition $\mathcal{C}$ of V into κ chains*

$$\mathcal{C} = \{\, C_1, \ldots, C_\kappa \,\} \quad \textit{such that } \kappa = |A|.$$

Proof. By the perfect graph theorem, the complement of G is perfect, and so

$$\alpha(G) = \kappa(G).$$

A clique cover corresponds with a partition of V into a collection of chains. □

[17] R. Dilworth, A decomposition theorem for partially ordered sets, *Annals of Mathematics* **51** (1950), pp. 161–166.

Comparability graphs can be characterized by a set of forbidden induced subgraphs. That is, there exists a collection of graphs $\mathcal{F}$ such that a graph G is a comparability graph if and only if no element of $\mathcal{F}$ is an induced subgraph of G. The set $\mathcal{F}$ is not finite, for example it contains all the odd cycles of length at least five.[18]

In the following figures, Figure 3.2 (Table I) and Figure 3.3 on the facing page (Table II), we show Gallai's list of forbidden induced subgraphs for comparability graphs. Some care is needed, since the *complements* of the graphs in Figure 3.3 are forbidden. By the way, the circled vertices in this figure form asteroidal triples. We'll talk about those in Section 3.6.

Theorem 3.18 (Gallai). *A graph is a comparability graph if and only if it has none of the graphs in Figure 3.2 and none of the complements of the graphs in Figure 3.3 as an induced subgraph.*

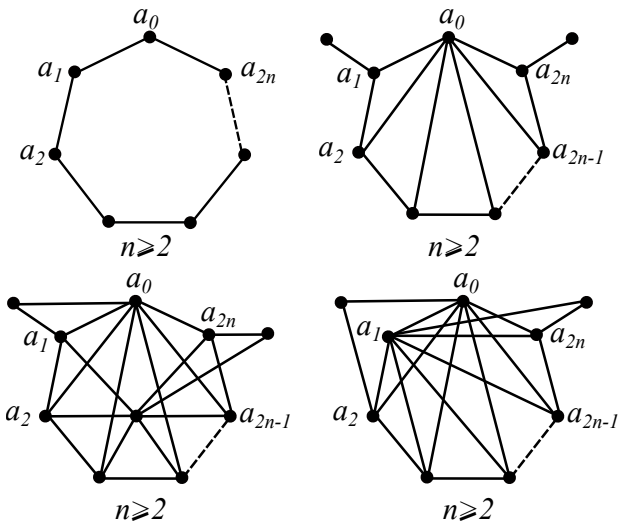


Fig. 3.2. Table I.

By the perfect graph theorem, the complements of comparability graphs are perfect. Let's have a closer look at those.

Definition 3.19. *A graph* G *is a cocomparability graph if* $\bar{G}$ *is a comparability graph.*

[18] T. Gallai, Transitiv orientierbare Graphen, *Acta Mathematica Academiae Scientiarum Hungaricae* **18** (1967), pp. 25–66.

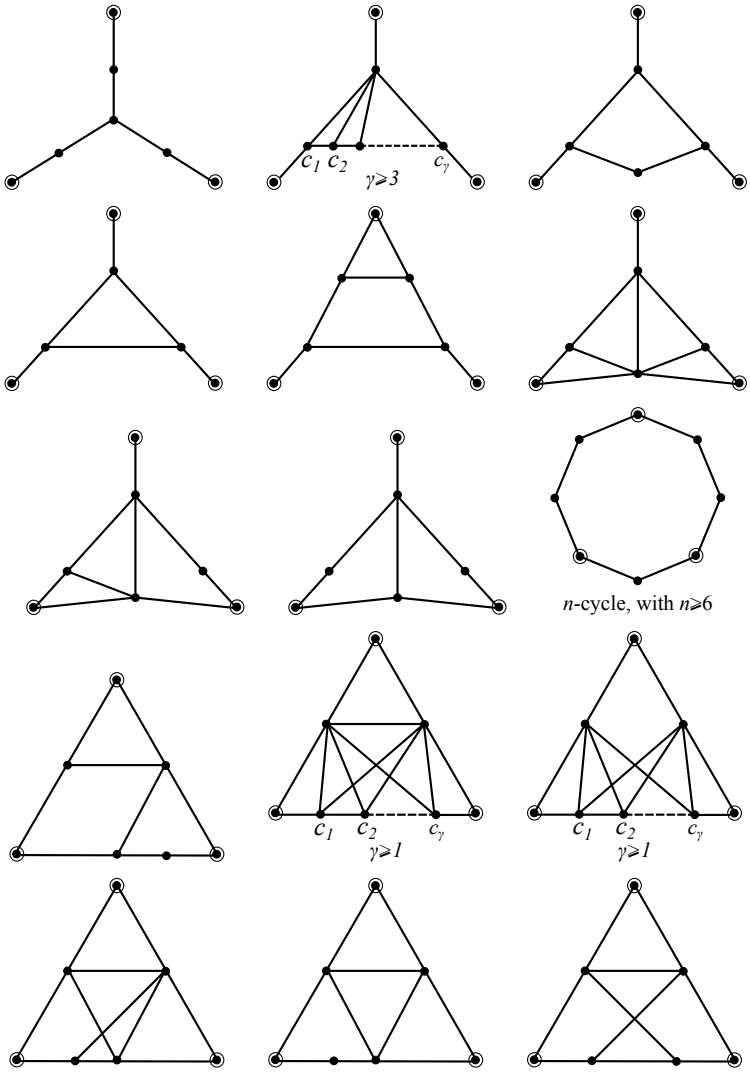


Fig. 3.3. Table II.

Cocomparability graphs have a nice intersection model.[19]

Theorem 3.20. *A graph* G *is a cocomparability graph if and only if for every vertex* $x \in V$ *there exists a continuous function*

$$f_x : [0, 1] \rightarrow \mathbb{R}$$

[19] M. Golumbic, D. Rotem and J. Urrutia, Comparability graphs and intersection graphs, *Discrete Mathematics* **43** (1983), pp. 37–46.

such that two vertices x and y are adjacent if and only if the functions f_x and f_y intersect each other. By that we mean that there exists a point $0 \leqslant r \leqslant 1$ such that

$$f_x(r) = f_y(r).$$

Proof. Let $P = (V, \preceq)$ be a poset. A realizer for P is a collection

$$\{ \leqslant_1, \ldots, \leqslant_k \}$$

of total orders of V such that $\preceq$ is the intersection of $\leqslant_i$, $i \in \{1, \ldots, k\}$. By that we mean that, for any two elements x and y in V,

$$x \preceq y \quad \text{if and only if} \quad x \leqslant_1 y \quad \text{and} \quad \ldots \quad \text{and} \quad x \leqslant_k y.$$

Dushnik and Miller show that every partial order has a realizer. A finite poset has a finite realizer. The dimension of the poset is the minimal cardinality of a realizer.[20]

Consider a realizer $\{\leqslant_1, \ldots, \leqslant_k\}$ of a poset $P = (V, \preceq)$. Let $V = \{1, \ldots, n\}$. Draw k vertical lines, $L_1, \ldots, L_k$, next to each other. On each line L_i label n points $\{1, \ldots, n\}$ in the order of $\leqslant_i$. Now connect any point on line L_i by a straight line segment with the point that has the same label on line L_{i+1}, for $i \in \{1, \ldots, k-1\}$. See Figure 3.4 on the next page.

Now we have that, for any two elements x and y in V, that $x \preceq y$ if and only if $x \leqslant_i y$ for all $i \in \{1, \ldots, k\}$. That is, $x \preceq y$ if and only if the functions f_x and f_y, defined as the concatenation of the straight line segments, do not intersect.

The converse is easy. If a graph G has an intersection model by continuous functions, as above, then the ordering (x, y), defined by "f_x lies below f_y," is a transitive orientation of $\bar{G}$.

This proves the theorem. □

Remark 3.21. In Section 3.7 we have a look at the posets of dimension two. The corresponding comparability graphs are called permutation graphs.

Remark 3.22. It is NP-complete to determine whether the dimension of a poset is at most k for any $k \geqslant 3$.[21]

[20] B. Dushnik and E. Miller, Partially ordered sets, *American Journal of Mathematics* **63** (1941), pp. 600–610.

[21] M. Yannakakis, The complexity of the partial order dimension problem, *SIAM Journal on Algebraic and Discrete Methods* **3** (1982), pp. 351–358.

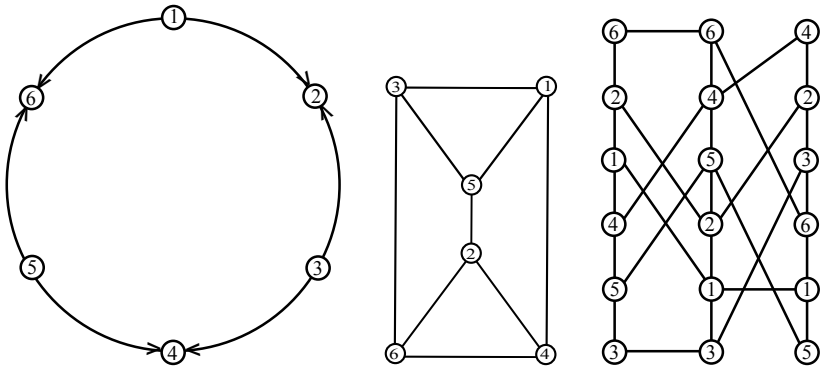


Fig. 3.4. This figure shows the complement of C_6. Notice that C_6 is a comparability graph. The figure shows the intersection model. The dimension of the poset is three. The picture on the right shows the three total orders of the realizer.

3.2.1 Recognition of comparability graphs

In this section we explain Golumbic's recognition algorithm for comparability graphs.[22][23] This algorithm can be implemented such that it runs in $O(n^3)$ time.

Let $G = (V, E)$ be a graph. For a set $\mathcal{E} \subseteq E$ write

$$\overrightarrow{\mathcal{E}} = \{ \overrightarrow{xy},\ \overrightarrow{yx} \mid \{x, y\} \in \mathcal{E} \}. \tag{3.6}$$

We call the elements of $\overrightarrow{E}$ the directed edges of G.

Let F be a set of directed edges of a graph G. Define the sets F^{-1}, F^2 and $\hat{F}$ as follows.

1.
$$F^{-1} = \{ \overleftarrow{xy} \mid \overrightarrow{xy} \in F \} \tag{3.7}$$

2.
$$F^2 = \{ \overrightarrow{xz} \mid \exists_{y \in V}\ \overrightarrow{xy} \in F \quad \text{and} \quad \overrightarrow{yz} \in F \} \tag{3.8}$$

3.
$$\hat{F} = F \cup F^{-1}. \tag{3.9}$$

---

[22] M. Golumbic, Comparability graphs and a new matroid, *Journal of Combinatorial Theory, Series B* **22**, (1977) pp. 68–90.

[23] M. Golumbic, The complexity of comparability graph recognition and coloring, *Computing* **18** (1977), pp. 199–208.

Definition 3.23. *Let* $G = (V, E)$ *be a graph and let* $\mathcal{E} \subseteq E$. *A set* $F \subseteq \overrightarrow{\mathcal{E}}$ *of directed edges is an orientation of* $\mathcal{E}$ *if*

$$F \cup F^{-1} = \overrightarrow{\mathcal{E}} \quad \textit{and} \quad F \cap F^{-1} = \varnothing. \tag{3.10}$$

If $\mathcal{E} = E$ *then* F *is an orientation of* G.

Definition 3.24. *Let* F *be an orientation of* G. *Then* F *is transitive if* $F^2 \subseteq F$. *A graph* G *is a comparability graph if it has a transitive orientation.*

Definition 3.25. *Define a binary relation* Γ *on* $\overrightarrow{E}$ *of a graph* $G = (V, E)$ *as follows.*

$$\overrightarrow{ab}\ \Gamma\ \overrightarrow{a'b'} \quad \Leftrightarrow \quad \begin{cases} a = a' & \textit{and} \quad \{b, b'\} \notin E, \quad \textit{or} \\ b = b' & \textit{and} \quad \{a, a'\} \notin E. \end{cases} \tag{3.11}$$

Notice that the relation Γ is reflexive and symmetric. Its transitive closure Γ^* is an equivalence relation on $\overrightarrow{E}$. We call the equivalence classes of Γ^* the implication classes of $\overrightarrow{E}$.

The Γ relation on $\overrightarrow{E}$ captures the fact that, for any transitive orientation F of G,

$$\boxed{\overrightarrow{ab}\ \Gamma\ \overrightarrow{a'b'} \quad \text{and} \quad \overrightarrow{ab} \in F \quad \Rightarrow \quad \overrightarrow{a'b'} \in F.} \tag{3.12}$$

Golumbic gave a simple algorithm to test whether an undirected graph G is a comparability graph and to give it a transitive orientation if it is. The central part of Golumbic's algorithm is to compute a G-decomposition of $\overrightarrow{E}$ which is defined as follows.

Definition 3.26. *Let* $G = (V, E)$ *be a graph. A partition of* $\overrightarrow{E}$

$$\{\ \hat{B}_1,\ \hat{B}_2,\ \cdots,\ \hat{B}_k\ \} \tag{3.13}$$

is a G*-decomposition if, for* $i \in \{1, \ldots, k\}$, B_i *is an implication class of*

$$\bigcup_{\ell=i}^{k} \hat{B}_\ell. \tag{3.14}$$

Golumbic's algorithm follows directly from the following theorem.

Theorem 3.27. *Let* $G = (V, E)$ *be a graph with a* G*-decomposition*

$$\overrightarrow{E} = \bigcup_{i=1}^{k} \hat{B}_i. \tag{3.15}$$

The following statements are equivalent.

(i) G *is a comparability graph.*
(ii) $A \cap A^{-1} = \varnothing$ *for all implication classes* A *of* G.
(iii) $B_i \cap B_i^{-1} = \varnothing$ *for* $i \in \{1, \ldots, k\}$.

Furthermore, when these conditions hold, then

$$\bigcup_{\ell=1}^{k} B_\ell \tag{3.16}$$

is a transitive orientation of E.

By Theorem 3.27, we can test whether a graph G is a comparability graph, and give G a transitive orientation if it is a comparability graph, by computing a G-decomposition of $\overrightarrow{E}$. Golumbic describes an $O(n^3)$ algorithm to compute a G-decomposition.

To prove Theorem 3.27 we need a lemma.

Lemma 3.28 (The Triangle Lemma). *Let* X*,* Y *and* Z *be three implication classes of a graph* $G = (V, E)$*. Assume that*

$$X \neq Y \quad \textit{and} \quad X \neq Z^{-1}. \tag{3.17}$$

Let

$$\overrightarrow{xy} \in Z \quad \textit{and} \quad \overrightarrow{xz} \in Y \quad \textit{and} \quad \overrightarrow{yz} \in X. \tag{3.18}$$

Then

$$\overrightarrow{y'z'} \in X \quad \Rightarrow \quad \overrightarrow{xy'} \in Z \quad \textit{and} \quad \overrightarrow{xz'} \in Y. \tag{3.19}$$

Proof. By definition, there exists a chain

$$\overrightarrow{yz} = \overrightarrow{y_0z_0} \,\Gamma\, \overrightarrow{y_1z_0} \,\Gamma\, \overrightarrow{y_1z_1} \,\Gamma\, \ldots \,\Gamma\, \overrightarrow{y_kz_k} \quad \text{with } y_k = y' \text{ and } z_k = z'. \tag{3.20}$$

We use induction on i. Assume that $\overrightarrow{xz_i} \in Y$ and $\overrightarrow{xy_i} \in Z$. Then we have that (since $X \neq Y$ and $X \neq Z^{-1}$)

$$\begin{aligned}
Y \ni \overrightarrow{xz_i} \;\not\Gamma\; \overrightarrow{y_{i+1}z_i} \in X \quad &\Rightarrow \quad \{x, y_{i+1}\} \in E \\
\{y_{i+1}, y_i\} \notin E \quad &\Rightarrow \quad \overrightarrow{xy_{i+1}} \,\Gamma\, \overrightarrow{xy_i} \in Z \\
Z^{-1} \ni \overrightarrow{y_{i+1}x} \;\not\Gamma\; \overrightarrow{y_{i+1}z_{i+1}} \in X \quad &\Rightarrow \quad \{x, z_{i+1}\} \in E \\
\{z_i, z_{i+1}\} \notin E \quad &\Rightarrow \quad \overrightarrow{xz_{i+1}} \,\Gamma\, \overrightarrow{xz_i} \in Y.
\end{aligned}$$

Therefore, $\overrightarrow{xy_k} \in Z$ and $\overrightarrow{xz_k} \in Y$. □

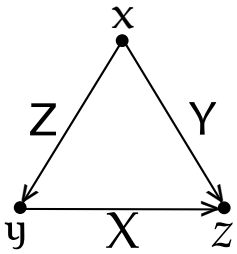


Fig. 3.5. The figure illustrates the Triangle Lemma.

We are now ready to prove Theorem 3.27.

Theorem 3.29. *Let* $G = (V, E)$ *be a graph with a* G*-decomposition*

$$E = \{ \hat{B}_1, \cdots, \hat{B}_k \}. \tag{3.21}$$

The following statements are equivalent.

(i) G *is a comparability graph.*
(ii) $A \cap A^{-1} = \varnothing$ *for all implication classes* A *of* $\overrightarrow{E}$.
(iii) $B_i \cap B_i^{-1} = \varnothing$ *for* $i \in \{1, \ldots, k\}$.

Furthermore, when these conditions hold, then

$$\bigcup_{\ell=1}^{k} B_\ell \tag{3.22}$$

is a transitive orientation of E.

Proof. Assume that G is a comparability graph. We show that (i) implies (ii). Let A be an implication class. We show that $A \cap A^{-1} = \varnothing$.

Let F be a transitive orientation. By applying (3.12) inductively, either

$$A \cap F = \varnothing \quad \text{or} \quad A \subseteq F.$$

First assume that $A \cap F = \varnothing$. Since $A \subseteq F \cup F^{-1}$ we have that $A \subseteq F^{-1}$. Since $F \cap F^{-1} = \varnothing$ we have that $A \cap A^{-1} = \varnothing$. Now assume that $A \subseteq F$. Since F^{-1} is also a transitive orientation of G, it follows similarly that $A \cap A^{-1} = \varnothing$.

We now show that (ii) implies (iii). We use induction on the number of implication classes in the G-decomposition. By assumption, B_1 is an implication class of G. When $k = 1$ we are done. Assume that the claim holds true for all graphs G' with less than k implication classes in some G'-decomposition.

Consider the graph induced by $E \setminus \hat{B}_1$. Let D be an implication class of $E \setminus \hat{B}_1$. We claim that either D is an implication class of E or $D = C_1 \cup C_2$, where C_1 and C_2 are two implication class of E.

The removal of $\hat{B}_1$ causes some implication classes to merge into one. Assume that D is the union of t implication classes and assume that $t \geqslant 2$. Then there exists a triangle $\{x, y, z\}$ with $\overrightarrow{yz} \in \hat{B}_1$ and either

$$(\overrightarrow{xz} \in C_1 \quad \text{and} \quad \overrightarrow{xy} \in C_2) \quad \text{or} \quad (\overleftarrow{xz} \in C_1 \quad \text{and} \quad \overleftarrow{xy} \in C_2). \tag{3.23}$$

Without loss of generality we may assume that $\overrightarrow{xz} \in C_1$ and $\overrightarrow{xy} \in C_2$ since the alternative follows by symmetry by considering D^{-1}.

Assume that $C_1 = C_2^{-1}$. Then $\overrightarrow{yx} \in C_1$ and $\overrightarrow{xz} \in C_1$. But $\overrightarrow{yz} \notin C_1$ and so $C_1 = \hat{C}_1 = C_1^{-1}$ which implies $C_1 = C_2$. This is a contradiction. Therefore, $\hat{C}_1 \cap \hat{C}_2 = \varnothing$.

Any Γ-chain in $E \setminus \hat{B}_1$ with edges in $\hat{C}_1 \cup \hat{C}_2$ cannot contain edges from any other implication class since any triangle in E with one edge in $\hat{B}_1$ and the second in $\hat{C}_1$ must have its third edge in $\hat{C}_2$. Thus $t = 2$ and $D = C_1 \cup C_2$.

If D is an implication class of $E \setminus \hat{B}_1$ then $D \cap D^{-1} = \varnothing$. If $D = C_1 \cup C_2$, for two implication classes C_1 and C_2 with $\hat{C}_1 \cap \hat{C}_2 = \varnothing$, then

$$\begin{aligned} D \cap D^{-1} &= (C_1 \cup C_2) \cap (C_1^{-1} \cup C_2^{-1}) \\ &= (C_1 \cap C_1^{-1}) \cup (C_2 \cap C_2^{-1}) = \varnothing. \end{aligned}$$

Therefore, by induction, $B_i \cap B_i^{-1} = \varnothing$ for $i \in \{2, \ldots, k\}$.

We prove that (iii) implies (i). We first show that B_1 is transitive. Let $\overrightarrow{xy} \in B_1$ and $\overrightarrow{yz} \in B_1$. If $\{x, z\} \notin E$ then

$$\overrightarrow{xy} \; \Gamma \; \overrightarrow{zy} \quad \Rightarrow \quad \overrightarrow{zy} \in B_1 \quad \Rightarrow \quad \overrightarrow{yz} \in B_1^{-1} \tag{3.24}$$

which is a contradiction since $B_1 \cap B_1^{-1} = \varnothing$. Thus $\{x, z\} \in E$. Let W be the implication class containing $\overrightarrow{xz}$ and assume that $W \neq B_1$. Since $B_1 \neq B_1^{-1}$ and $\overrightarrow{xy} \in B_1$, by the triangle lemma, $\overrightarrow{xy} \in W$ which is a contradiction. Thus $\overrightarrow{xz} \in B_1$ and this proves that B_1 is transitive.

When $k = 1$ we are done. Assume that the implication holds for all graphs G' that have a G'-decomposition with less than k implication classes. Thus

$$F = \bigcup_{\ell=2}^{k} B_i \tag{3.25}$$

is transitive.

Let $\overrightarrow{xy}$ and $\overrightarrow{yz}$ be two elements of $B_1 \cup F$. If both are in B_1 or if both are in F then we are done. Assume that $\overrightarrow{xy} \in B_1$ and $\overrightarrow{yz} \in F$. Then

$$\overrightarrow{xy} \; \not\Gamma^* \; \overrightarrow{zy} \quad \Rightarrow \quad \{x, z\} \in E. \tag{3.26}$$

Assume that $\overrightarrow{xz} \notin B_1 \cup F$. Then $\overrightarrow{zx} \in B_1 \cup F$. Now

(a) $\overrightarrow{zx} \in B_1$ and $\overrightarrow{xy} \in B_1$ imply $\overrightarrow{zy} \in B_1$ which is a contradiction, and
(b) $\overrightarrow{zx} \in F$ and $\overrightarrow{yz} \in F$ imply $\overrightarrow{yx} \in F$, which is also a contradiction.

Thus $\overrightarrow{xz} \in B_1 \cup F$. Similarly, $\overrightarrow{xy} \in F$ and $\overrightarrow{yz} \in B_1$ imply $\overrightarrow{xz} \in B_1 \cup F$.

This proves that $\cup_{i=1}^{k} B_i$ is a transitive orientation of G.

This proves the theorem. □

A G-decomposition for a graph $G = (V, E)$ can be found as follows.

1. Let $i = 1$ and let $E_1 = E$.
2. Choose $e_i \in E_i$.
3. Enumerate the implication class B_i of E_i which contains e_i.
4. Define $E_{i+1} = E_i \setminus \hat{B}_i$.
5. If $E_{i+1} = \varnothing$ then $k = i$ and STOP. Otherwise, increase i by one and continue with step 2.

Theorem 3.30. *There exists an $O(n^3)$ algorithm to recognize comparability graphs.*

Proof. An edge is put into an implication class at most one time. Whenever an edge e is put in an implication class, all the neighbors of the two endpoints are examined. For all the neighbors that are adjacent to exactly one endpoint of e, the edges are put in the same implication class as e.

A flag on each edge is used to detect when an arc $\overrightarrow{xy}$ is detected for which $\overleftarrow{xy}$ is in the same implication class. When this happens, the graph is not a comparability graph and otherwise, by Theorem 3.27 it is.

Each edge is put into an implication class only once. When an edge e is put in an implication class $O(n)$ vertices are examined to see which edges are Γ-related to e. This shows that the algorithm can be implemented to run in $O(nm)$ time. □

Remark 3.31. By Theorem 3.16 on page 43, when a transitive orientation is a part of the input, then a maximum clique and an optimal coloring in a comparability graph can be computed in $O(n^2)$ time by dynamic programming. Therefore, the clique and the chromatic number problem can be solved in $O(n^3)$ time on comparability graphs.[24]

[24] M. Golumbic, The complexity of comparability graph recognition and coloring, *Computing* **18** (1977), pp. 199–208.

3.3 Cographs

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

Definition 3.32. *A graph* G *is a cograph if it has no induced* P_4*, which is a path with four vertices.*

Fig. 3.6. 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.

Lemma 3.33. *If* G *is a cograph then* G *is perfect.*

Proof. Let G be a cograph. We show that neither 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 3.5. □

Theorem 3.34. *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 the complement of $G[C'_1]$ is connected, we may assume that a' and b' are not adjacent 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. □

Remark 3.35. There are many alternative characterizations. For example, a graph is a cograph if and only if in every induced subgraph every maximal clique intersects every maximal independent set (in one vertex).

Remark 3.36. A very interesting subclass of cographs is the class of trivially perfect graphs. These are the graphs without induced P_4 and C_4.

An intersection model for the trivially perfect graphs can be obtained as follows. If $G = (V, E)$ is a connected trivially perfect graph then there exists a rooted tree with vertex set V such that any two vertices x and y are adjacent if and only if one of them lies on the path to the root of the other one. It follows that G is trivially perfect if and only if every connected, induced subgraph has a universal vertex, that is a vertex which is adjacent to all other vertices in that subgraph.

The reason why this class is called trivially perfect is the following. A graph is trivially perfect if and only if in every induced subgraph H, the number of maximal cliques is equal to the independence number $\alpha(H)$. (Notice that the leaves of the tree described above form a maximum independent set and that the paths from the leaves to the root form all maximal cliques.)

This property implies (trivially) that the graphs are perfect, since

$$\alpha(H) \leqslant \kappa(H)$$

for any graph H and $\kappa(H)$ is of course at most the number of maximal cliques, which is $\alpha(H)$ by the property.[25] [26]

3.3.1 Cotrees

Let G be a cograph. Theorem 3.34 shows us how we can build a decomposition tree for the graph G. For cographs, this decomposition tree is called a cotree.[27]

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 a $\oplus$-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 3.34, 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 (binary) cotree. One obtains a somewhat different cotree by recursively creating a child for *every* component of the subgraph or its complement (see Figure 3.7). In that case, the labels on every path from a leaf to the root alternate between $\oplus$ and $\otimes$.

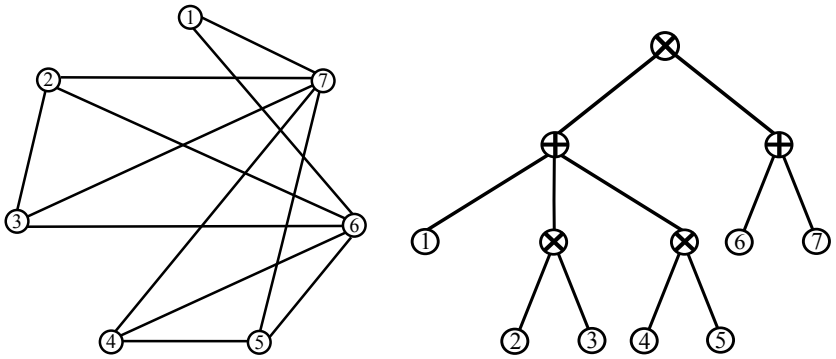
Notice that a graph G is a cograph if and only if it has a cotree (see Exercise 3.9). Corneil, Perl and Stewart proved the following theorem.[28]

[25] E. Wolk, A note on the comparability graph of a tree, *Proceedings of the American Mathematical Society* **16** (1965), pp. 17–20.

[26] M. Golumbic, Trivially perfect graphs, *Discrete Mathematics* **24** (1978), pp. 105–107.

[27] D. Corneil, H. Lerchs and L. Stewart-Burlingham, Complement reducible graphs, *Discrete Applied Mathematics* **3** (1981), pp. 163–174.

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



(a) a cograph (b) a cotree

Fig. 3.7. The figure shows a cograph and a cotree. Notice the difference with the binary cotree described above. When a cotree is built by creating a child for each component of the subgraph or its complement, the labels alternate on every path from a leaf to the root. For algorithmic purposes one usually prefers the binary cotree.

Theorem 3.37. *There exists a linear-time algorithm that recognizes cographs. When* G *is a cograph then this algorithm builds a cotree for* G.

3.3.2 Finding cliques in cographs

One appreciates cotrees when one looks at some examples. Let's use it for the computation of the clique number.

Theorem 3.38. *There exists a linear-time algorithm that computes the clique number* $\omega(G)$ *of a cograph* G.

Proof. Let $G = (V, E)$ be a cograph. First, the algorithm builds a cotree (T, f) for G. By Theorem 3.37 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

$$\omega(G_p) = \omega(G_1) + \omega(G_2). \tag{3.27}$$

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

$$\omega(G_p) = \max\{\, \omega(G_1),\ \omega(G_2)\,\} \tag{3.28}$$

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 (3.27) and (3.28).

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. □

Remark 3.39. Obviously, there exists also a linear-time algorithm to compute a maximum independent set via formulas similar to (3.27) and (3.28).

Recall Definition 2.16 on page 13. A dominating set in a graph $G = (V, E)$ is a set D of vertices such that every vertex $x \in V \setminus D$ has at least one neighbor in D. We denote the minimal cardinality of a dominating set in G by $\gamma(G)$. The formulas to compute $\alpha(G)$ and $\gamma(G)$ for each of the nodes in a cotree are almost the same, except that the max-operator in Equation (3.28) for the $\otimes$-nodes is replaced by

$$\min\{\, \gamma(G_1),\ \gamma(G_2),\ 2\,\}.$$

It follows that also $\gamma(G)$ can be computed in linear time for cographs.

3.4 Distance-hereditary graphs

Edward Howorka introduced distance-hereditary graphs (abbr. DH-graphs).[29]

Definition 3.40. *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.)

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

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. See Figure 3.8.

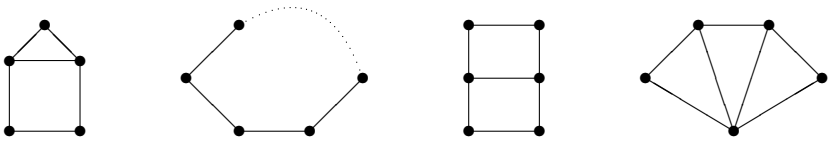


Fig. 3.8. A graph is distance hereditary if it has no induced house, hole, domino or gem.

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.

Definition 3.41. *A twin is a pair of vertices* x *and* y *such that either*

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

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 3.14 we ask you to check that G' is also distance hereditary.

Theorem 3.42. *Let* $G = (V, E)$ *be a graph. The following statements are equivalent.*

(1) *The graph* G *is distance hereditary.*
(2) *Every induced subgraph of* G *has an isolated vertex, or a pendant vertex or a twin.*
(3) *The graph* G *has no induced house, hole, domino or gem.*

Proof. We prove

$$(1) \quad \Rightarrow \quad (3) \quad \Rightarrow \quad (2) \quad \Rightarrow \quad (1).$$

It is easy to check that, when G is distance hereditary, then it has no induced house, hole, domino or gem. We show that this implies that every induced subgraph H of G has an isolated vertex, or a pendant vertex or a twin.

First, since the class of distance-hereditary graphs is hereditary, it is sufficient to prove that G has an isolated vertex, or a pendant vertex or a twin. We may assume that G is connected, since a graph is distance hereditary if and only if every component of G is distance hereditary.

Let x be a vertex for which the largest component C of $G - N[x]$ has a largest number of vertices. We may assume that $C \neq \varnothing$, since otherwise G is a clique and then we are done. Let

$$\Delta = N(C) = \{\, y \mid y \in V \setminus C \quad \text{and} \quad N(y) \cap C \neq \varnothing \,\}.$$

Thus $\Delta \subseteq N(x)$ and every vertex of Δ has a neighbor in C. Notice that $\Delta \neq \varnothing$ since we assume that G is connected.

Notice that, by the choice of x,

every vertex of $V \setminus (C \cup \Delta)$ is adjacent to every vertex of Δ.

(If this were not true, then there would be another vertex x' with a larger component than C in $G - N[x']$, namely, one component of $G - N[x']$ would include C and some vertex of Δ.)

Now, notice that all vertices of Δ have the same neighbors in C, since otherwise there would be a house, hole, domino or gem. (This takes some effort to figure out, but it is not difficult and we leave it as an exercise.)

Let

$$\Omega = V \setminus (C \cup \Delta).$$

Thus $x \in \Omega$ and every vertex of Ω is adjacent to every vertex of Δ. Notice that Ω has no induced P_4 since otherwise, with some vertex $\delta \in \Delta$ there would be a gem and a gem is not distance hereditary.

Thus $G[\Omega]$ is a cograph, and so, by Exercise 3.10, either it has only one vertex or it has a twin. Any twin in $G[\Omega]$ is a twin in G, since all vertices of Ω have the same neighbors in $G - \Omega$, namely the vertices of Δ.

Thus, now we may assume that $\Omega = \{x\}$, since otherwise there is a twin. Notice that $G[\Delta]$ has no induced P_4, since otherwise there would be an induced gem in G (with the vertex x), and a gem is clearly not distance hereditary. Thus, if $G[\Delta]$ has at least two vertices then it contains a twin, since $G[\Delta]$ is a cograph. (Here we use Exercise 3.10 again.)

A twin in $G[\Delta]$ is a twin in G, since all vertices of Δ have the same neighbors in $V \setminus \Delta$. Thus, finally, we may assume that $|\Delta| = 1$. But then x is a pendant vertex.

Thus, if G is distance hereditary, then every induced subgraph has an isolated vertex, or a pendant vertex or a twin. We leave it as an easy exercise (Exercise 3.14) to prove the converse.

This proves the theorem. □

Theorem 3.43. *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 vertex 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 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. □

3.4.1 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.[30] [31]

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

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

[31] H. Bandelt and H. Mulder, Distance-hereditary graphs, *Journal of Combinatorial Theory, Series B* **41** (1986), pp. 182–208.

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 3.44. 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

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

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. See Figure 3.9 for another example.

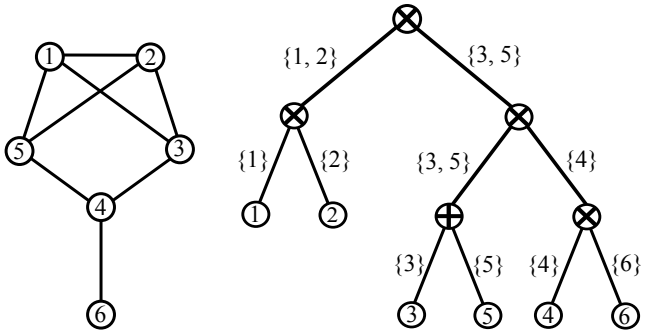


Fig. 3.9. An example of a distance-hereditary graph and a decomposition tree for it. The labels at the edges are the twinsets. The $\oplus$- and $\otimes$-labels at the nodes represent the union- and join-operation on the twinsets of the two children.

Theorem 3.45. *Let* G *be a graph. Then* G *is distance hereditary if and only if it has a decomposition tree as described above.*

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.

Assume that a graph G has a decomposition tree as described above. You can use the technique of Exercise 3.10 to show that every induced subgraph has an isolated vertex, a pendant vertex or a twin. We leave it as an exercise. □

When G is distance hereditary then a tree-decomposition for G with the three properties described above can be obtained in linear time.[32]

3.4.2 Feedback vertex set in DH-graphs

To appreciate decomposition trees for DH-graphs you need to look at some examples. In this section we show how to use it for the computation of a feedback vertex set.

Definition 3.46. *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.*

[32] E. Gioan and C. Paul, Split decomposition and graph labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. Manuscript on arXiv: 0810.1823v2, 2012.

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.

Recall (from Section 3.3.2) that a graph G is a join of two graphs G_1 and G_2, if G is obtained from 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.

When G is disconnected then a feedback vertex set for G is of course very easy to obtain from minimum feedback vertex sets of its components. In the following two lemmas we summarize two basic observations for the case where G is the join of two graphs.

Lemma 3.47. *Assume that G is the join of two graphs G_1 and G_2. For $i \in \{1,2\}$, let V_i be the vertex set of G_i. Let F be a feedback vertex set of G. Then*

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

Proof. Otherwise $G - F$ contains a 4-cycle, that is, an induced 4-cycle or a triangle. □

Lemma 3.48. *Let G be a cograph which is the join of cographs G_1 and G_2. Let V_1 and V_2 be the vertex sets of G_1 and G_2. Let F be a feedback vertex set of G. Assume that $|V_1 \setminus F| = 1$. Then $G_2 - F$ is an independent set.*

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

We are ready to prove our theorem.

Theorem 3.49. *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 no two vertices of $Q - F$ are in one component of $G[W] - F$;
(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 3.18).

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.
This completes the proof. □

One of the oldest classes of graphs that have been studied in great detail is the class of chordal graphs.[33] We'll give you a crash course in chordal graphs in the next section. Brace yourselves!

3.5 Chordal graphs

Definition 3.50. *A graph is chordal if it has no induced cycle of length more than three.*

[33] J. Blair, B. Peyton, An introduction to chordal graphs and clique trees. In: (A. George, J. Gilbert, J. Liu eds.) *Graph theory and sparse matrix computation*, Springer-Verlag, 1993, pp. 1–29. Notice that Theorem 2.1 of this paper (the 'proof' is M. Golumbic's erroneous proof) is not correct. Dirac gives a counterexample in his paper.

Notice that the class of chordal graphs contains, for example, all trees. It may be helpful to keep this in mind; chordal graphs look a lot like ordinary trees.

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

Theorem 3.51. *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.[34]

Definition 3.52. *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 3.53. *An x, y-separator S is a minimal x, y-separator if no proper subset of S is an x, y-separator.*

Definition 3.54. *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 3.55. 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. So, both $\{a, c\}$ and $\{c\}$ are minimal separators (separating different pairs of vertices) and

$$\{c\} \subset \{a, c\}.$$

(See Figure 3.10.) By the way, $\{b, d\}$ is another minimal a, e-separator (disjoint from $\{c\}$, that is, $\{c\} \cap \{b, d\} = \varnothing$). Minimal separators for the same pair do not have to be disjoint, but one cannot properly contain another one.

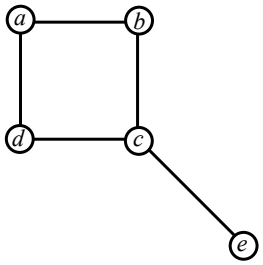


Fig. 3.10. This graph has a minimal separator contained in another one.

Theorem 3.56. *A graph is chordal if and only if each 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 3.20 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*.[35] To get the idea, you may think of simplicial vertices as the leaves of a tree.

Definition 3.57. *Let $G = (V, E)$ be a graph. A vertex x in G is simplicial if $N(x)$ induces a clique in G.*

[34] Theorem 1 in: G. Dirac, On rigid circuit graphs, *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* **25** (1961), pp. 71–76.

[35] D. Rose, R. Tarjan and G. Lueker, Algorithmic aspects of vertex elimination of graphs, *SIAM Journal on Computing* **5** (1976), pp. 266–283.

Theorem 3.58. *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 $N(z)$ induces a clique in G, 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 a simplicial vertex in G. □

Definition 3.59. *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 3.60. *A graph is chordal if and only if it has a perfect elimination ordering.*

Proof. This is an immediate consequence of Theorem 3.58. □

In Exercise 3.24 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.[36] 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 3.61. *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. □

3.5.1 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 3.62. *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 points of T. For a point i in T, let $S_i \in \mathcal{S}$ be the subset that is assigned to the point 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*

[36] 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.

(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 3.63. *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 $C_1, \ldots, C_t$ 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

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

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 $i = 2, \ldots, t$ 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. □

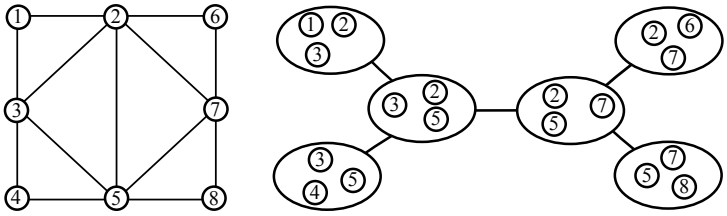


Fig. 3.11. This figure shows a chordal graph and a clique tree.

The other way to describe clique trees is as follows.

Theorem 3.64. *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 3.60, G is chordal.
This proves the theorem. □

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

Theorem 3.65. *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 3.66. *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 3.60 on Page 68, G has a perfect elimination ordering

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

Let

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

Then

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

This proves the theorem. □

Let $G = (V, E)$ be a chordal graph. Then G is perfect, and $\chi(G) = \omega(G)$. By Theorem 3.66 there exists a linear-time algorithm that computes $\chi(G)$. An actual coloring is also very easy to obtain, as we show next.

Theorem 3.67. *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

$$N(x_i) \cap \{\, x_{i+1},\ \ldots,\ x_n\,\}. \tag{3.29}$$

To see that this is possible, notice that the set in Formula (3.29) contains at most $\omega(G) - 1$ vertices. Thus there is a color in Ω available to color x_i.

The claim follows by induction. □

3.6 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.

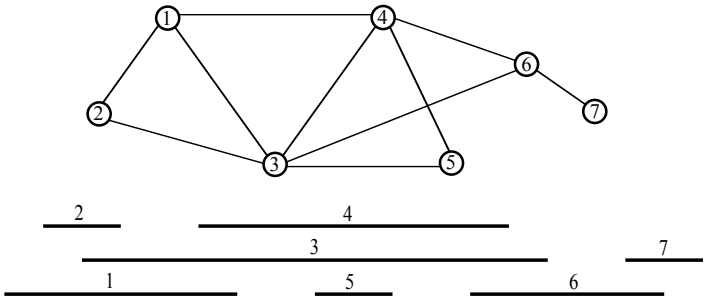


Fig. 3.12. The figure shows an interval graph.

Definition 3.68. *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* $I_x \cap I_y \neq \emptyset$.

Remark 3.69. 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, that is, they have finite length.

Theorem 3.70. *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 $[x_1, \ldots, x_k]$. Suppose it has an interval representation. By the previous observation we may assume that

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

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 $\{x_1, \ldots, x_{k-1}\}$. Thus it is a chordless cycle (that is an induced cycle) only for $k = 3$. □

Corollary 3.71. *Interval graphs are perfect.*

By Theorem 3.70 every interval graph is chordal and by Theorem 3.63 on Page 69 every chordal graph has a clique tree. We show next that interval graphs have a clique tree which is a path.[37]

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

The clique tree for interval graphs is usually called

a consecutive clique arrangement.

Theorem 3.72. *An interval graph* G *has a consecutive clique arrangement, that is, there is a linear ordering*

$$[M_1, \ldots, M_t]$$

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 3.28).

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. □

Definition 3.73. *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 3.74. 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.

Remark 3.75. 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. More examples can be found in Figure 3.3 on page 45.

Actually, Gallai found those graphs that contain asteroidal triples (and that are minimal with respect to the induced subgraph relation) and which do not contain C_5. Ekkehard Köhler completed the list.[38] The remaining graphs are depicted in Figure 3.13.

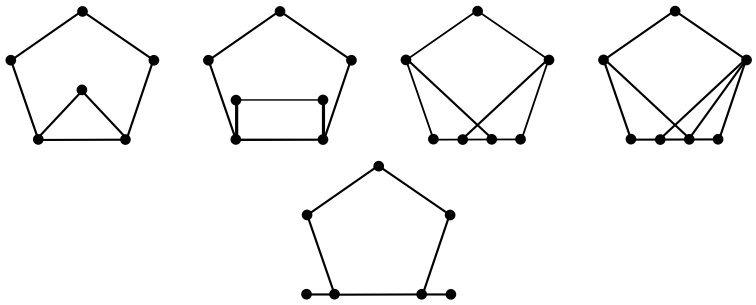


Fig. 3.13. This figure shows the remaining minimal graphs that contain asteroidal triples. These remaining ones are those that contain a C_5. The set of all minimal graphs that contain asteroidal triples consists of the graphs in Figure 3.3 on page 45 plus the ones in this figure.

Lemma 3.76. *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 $P = [x = x_1, \ldots, x_k = z]$ 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. □

[38] Ekkehard Köhler, *Graphs without asteroidal triples*, PhD Thesis, TU-Berlin, 1999.

Lekkerkerker and Boland characterize interval graphs as those chordal graph that have no asteroidal triple.[39] Halin gave an elegant proof of the theorem. We present Halin's proof. [40]

Theorem 3.77 (Lekkerkerker and Boland, Halin). *A graph* G *is an interval graph if and only if* G *is chordal and has no asteroidal triple.*

Proof. Assume that G is chordal and that it has no asteroidal triple. We prove that G has a consecutive clique arrangement.

By Theorem 3.63 on page 69, G has a clique tree. We prove that this tree can be turned into a path. We use induction on the number of maximal cliques.

When G is a clique then we are done. Assume that G has $k+2$ maximal cliques and assume that for all AT-free chordal graphs with less maximal cliques the claim is true.

Consider a clique tree for G. Let C be a maximal clique which is mapped to a leaf. Remove that leaf from the clique tree. The remaining clique tree is the clique tree of a chordal graph G'. The vertices of G' are the vertices of G minus the simplicial vertices that appear only in C. Repeating the process, each time removing a leaf from the clique tree, gives a sequence of cliques, in reversed order:

$$C_0,\ C_1,\ \ldots,\ C_k,\ C_{k+1} = C. \tag{3.30}$$

Let H be the graph induced by

$$\bigcup_{i=0}^{k} C_i. \tag{3.31}$$

Then H is chordal and it contains no asteroidal triple. By induction, H is an interval graph and we may assume that the cliques can be arranged consecutively, say

$$[\ C_0,\ \ldots,\ C_k\ ]. \tag{3.32}$$

Define

$$S_i = (C_0 \cup \cdots \cup C_{i-1})\ \cap\ C_i, \quad \text{for } i \in \{\ 1,\ \ldots,\ k\ \}. \tag{3.33}$$

We write

$$S = (C_0 \cup \ldots, C_k)\ \cap\ C.$$

If $S \subset C_0$ or $S \subset C_k$ then we are done. In that case we can place C before C_0 or after C_k. This gives a consecutive clique arrangement for G.

---

[39] C. Lekkerkerker and J. Boland, Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae* **51** (1962), pp. 45–64.

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

Notice that S is contained in at least one clique of H. Assume that $S \subset C_m$ for $0 < m < k$. We now consider two cases.

In the first case, assume that $S_i \subseteq S$ for at least one $i \in \{1, \ldots, k\}$. Define $S_0 = S_{k+1} = \varnothing$ and

$$p = \max \{\, i \mid S_i \subseteq S \quad \text{and} \quad 0 \leqslant i \leqslant m \,\}$$
$$q = \min \{\, i \mid S_i \subseteq S \quad \text{and} \quad m+1 \leqslant i \leqslant k+1 \,\}.$$

Since we assume that $S \not\subseteq C_0$ and $S \not\subseteq C_k$ we have that

$$p \geqslant 1 \quad \text{or} \quad q \leqslant k. \tag{3.34}$$

Let

$$Z = C_p \cup \ldots \cup C_m \cup \ldots \cup C_{q-1}. \tag{3.35}$$

Then $Z \setminus S$ is the component of $H - S$ which contains $C_m \setminus S$. Notice that,

$$S_p \subseteq C_j \quad \text{for } j \in \{p, \ldots, m\}, \text{ and} \quad S_q \subseteq C_j \quad \text{for } j \in \{m, \ldots, q-1\}. \tag{3.36}$$

This follows because $S_p \subseteq S \subseteq C_m$ and, by definition also $S_p \subseteq C_p$. Similarly, $S_q \subseteq S \subseteq S_m$ and $S_q \subseteq C_q$, which yields the second series of relations.

Notice that the maximal cliques of $Z \cup C$ are $C_p, \ldots, C_{q-1}$ and C, and so $Z \cup C$ has less maximal cliques than G. Thus we may assume that $Z \cup C$ has a consecutive clique arrangement. Let this consecutive clique arrangement be

$$[\, Z_0, \ \ldots, \ Z_t \,]. \tag{3.37}$$

Since S does not separate $Z \cup C$, we have that

$$C = Z_0 \quad \text{or} \quad C = Z_t. \tag{3.38}$$

Without loss of generality, assume that $C = Z_0$.

Assume that $Z_t = C_\sigma$ for some $\sigma \in \{p, \ldots, q-1\}$. If $\sigma \leqslant m$ then $S_p \subseteq C_\sigma = Z_t$ and we have a consecutive clique arrangement

$$[\, C_0, \ \ldots, \ C_{p-1}, \ Z_t, \ \ldots, \ Z_1, \ C, \ C_q, \ \ldots, \ C_k \,]. \tag{3.39}$$

When $\sigma \geqslant m+1$ then $S_q \subseteq C_\sigma = Z_t$, and we have a consecutive clique arrangement

$$[\, C_0, \ \ldots, \ C_{p-1}, \ C, \ Z_1, \ \ldots, \ Z_t, \ C_q, \ \ldots, \ C_k \,]. \tag{3.40}$$

In the second case, assume that

$$S_i \not\subseteq S \quad \text{for } i \in \{1, \ldots, k\}. \tag{3.41}$$

Assume that

(i) $S \not\subseteq S_k$ and $S_i \not\subseteq S_k$ for $i \in \{1, \ldots, k-1\}$, and
(ii) $S \not\subseteq S_1$ and $S_i \not\subseteq S_1$ for $i \in \{2, \ldots, k\}$.

Then there exist vertices

$$x \in C_0 \setminus S_1 \quad y \in C_k \setminus S_k \quad z \in C \setminus S. \tag{3.42}$$

Then $\{x, y, z\}$ is an asteroidal triple.

If Item (i) does not hold then the first case applies, with C_k and S_k instead of C and S. If Item (ii) does not hold then the first case applies with C_0 and S_1 instead of C and S.

This proves the theorem. □

Lekkerkerker and Boland obtained a list of forbidden induced subgraphs for the class of interval graphs.[41]

Theorem 3.78 (Lekkerkerker and Boland). *A graph is an interval graph if and only if it does not contain any graph of Figure 3.14 as an induced subgraph.*

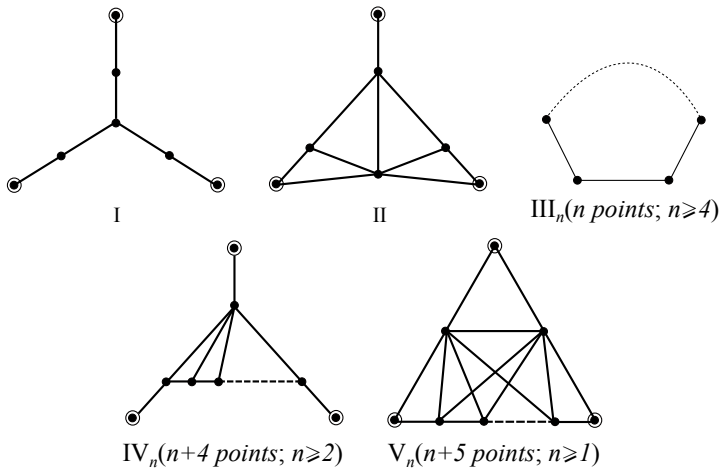


Fig. 3.14. The forbidden induced subgraphs for interval graphs. Notice that all of them, except C_4 and C_5 are contained in Gallai's table Figure 3.3 on page 45. The C_5 appears in Figure 3.2 on page 44. The C_4 is a comparability graph and so it does not appear in Gallai's list.

[41] C. Lekkerkerker and J. Boland, Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae* **51** (1962), pp. 45–64.

Let $G = (V, E)$ be a graph. Recall Section 3.2. 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 3.79. *A graph is a comparability graph if there exists a transitive orientation of its edges.*

Lemma 3.80. *If G is an interval graph 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 of 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, then there is an edge $\{x, z\}$ in $\bar{G}$ and it is oriented from x to z.

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

Actually, Lemma 3.80 'almost' characterizes interval graphs. Gilmore and Hoffman proved the following characterization.[42]

Theorem 3.81 (Gilmore and Hoffman). *A graph G is an interval graph if and only if G has no induced 4-cycle and $\bar{G}$ is a comparability graph.*

Proof. By Lemma 3.80 we only need to prove one direction. Assume that G is a graph without induced C_4 and assume that $\bar{G}$ has a transitive orientation. We first prove that G is chordal.

Assume that G has a hole H, that is, H is an induced cycle in G of length at least five. First notice that a transitive orientation of $\bar{G}$ induces a transitive orientation of H.

Number the vertices of H consecutively $[x_1, \ldots, x_t]$. First consider the neighbors of x_1 in $\bar{G}$, that is

$$\{x_3, \ldots, x_{t-1}\}.$$

Without loss of generality, assume that the edge $\{x_1, x_3\}$ is directed from x_1 to x_3. Then, by transitivity, the edge $\{x_1, x_4\}$ is directed from x_1 to x_4, otherwise

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

there would have to be an edge $\{x_3, x_4\}$ in $\bar{G}$. Repeating the argument we find that all edge $\{x_1, x_i\}$ for $i \in \{3, \ldots, t-1\}$ are directed from x_1 to x_i.

Consider the neighbors of x_2 in $\bar{G}$, that is the set $\{x_4, \ldots, x_t\}$. Since $\{x_1, x_{t-1}\}$ is directed from x_1 to x_{t-1} and since $\{x_1, x_2\}$ is not an edge in G, by transitivity the edge $\{x_2, x_{t-1}\}$ in $\bar{G}$ is directed from x_2 to x_{t-1}. (Notice that here we use the fact that $t > 4$.) As above, we find that all edges $\{x_2, x_i\}$ for $i \in \{4, \ldots, t\}$ are directed from x_2 to x_i.

We can repeat the argument, and we find that for all vertices x_i of H, all the edges $\{x_i, x_j\}$ of $\bar{G}$ are directed from x_i to x_j, which is of course a contradiction.

We conclude that G is chordal. From Theorem 3.20 on page 45 it follows easily that G has no asteroidal triple. (See Exercise 3.36.) By Theorem 3.77 we conclude that G is an interval graph. □

We summarize some results of Halin, Lekkerkerker and Boland, and Gilmore and Hoffman as follows.

Theorem 3.82. *Let* G *be a graph. The following statements are equivalent.*

(a) G *is an interval graph.*
(b) G *is AT-free and chordal.*
(c) G *has no induced* C_4 *and* $\bar{G}$ *is a comparability graph.*
(d) *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* $G - M_1$.

3.7 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 $1, 2, \ldots, n$. Connect each point on the topline by a straight linesegment to 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.[43] 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.

[43] 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.

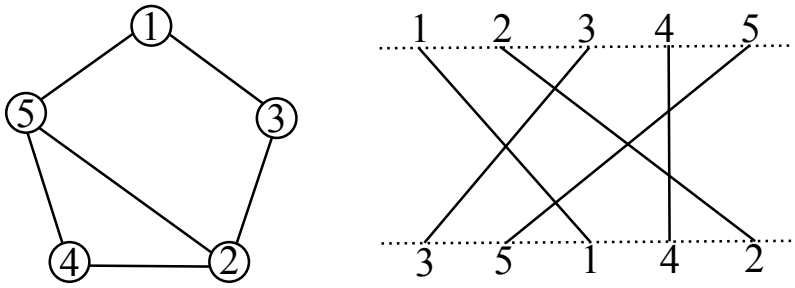


Fig. 3.15. A permutation graph and its permutation diagram

Lemma 3.83. *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 3.79. 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.

In the proof of Theorem 3.20 on page 45 we introduced the dimension of a poset, and in Remark 3.21 we promised that we would have a closer look at posets of dimension two. By presenting the next theorem we keep that promise.[44]

Theorem 3.84. *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 to the left of a linesegment y, and the linesegment of y is to the left of a linesegment z, then the linesegment of x is to the 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.

[44] K. Baker, P. Fishburn and F. Roberts, Partial orders of dimension 2, *Networks* **2** (1971), pp. 11–28.

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

3.7.1 Interval containment graphs

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.

Dushnik and Miller proved that classes of interval containment graphs and permutation graphs are the same.[45]

Theorem 3.85. *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.[46]

[45] B. Dushnik and E. Miller, Partially ordered sets, *American Journal of Mathematics* **63** (1941), pp. 600–610.

[46] 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.

Theorem 3.86. *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.*

Remark 3.87. Consider a collection $\mathcal{S}$ of subsets of some universal set U. One can construct a graph such that each element of $\mathcal{S}$ is represented by a vertex and in which two vertices are adjacent if one of the two subsets contains the other one. It is easy to see that the containment graphs are exactly the comparability graphs.[47] [48]

3.7.2 Cliques and independent sets in permutation graphs

To end this chapter smoothly, we'll have a quick and easy look at the clique and independence number of permutation graphs. Since permutation graphs are perfect, we get the chromatic number and the clique cover number for free.

Let G be a permutation graph. Consider a permutation diagram for G. Let the vertices on the topline be numbered from left to right as $1, \ldots, n$. Let

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

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.[49]

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

Theorem 3.88. *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.*

---

[47] W. Trotter, Jr., *Combinatorics and partially ordered sets – Dimension theory*, The Johns Hopkins University Press, Studies in the Mathematical Sciences, Baltimore, Maryland, 1992.

[48] J. Urrutia, Partial orders and Euclidean geometry. In: (I. Rival, ed.) *Algorithms and order*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1987, pp. 327–436.

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

3.8 Problems

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

3.2. Let $\mathcal{B}$ be the class of bipartite graphs. Is $\mathcal{B}$ hereditary?

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

3.4. 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?

3.5. 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

$$\alpha(G) + \tau(G) = n \quad \text{where } n = |V|. \tag{3.43}$$

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

3.7. Let G be a comparability graph. Design a linear-time algorithm to compute $\omega(G)$.
`Hint:` When G is a comparability graph then a transitive orientation can be obtained in linear time. You may assume that the graph has a transitive orientation.[50]

3.8. Is a cograph a comparability graph?

3.9. Show that a graph G is a cograph if and only if it has a cotree.

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

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

(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.

3.11. Show that a cograph with n vertices can have $3^{n/3}$ maximal cliques.
`Hint:` Show that the complement of the union of $\frac{n}{3}$ disjoint triangles is a cograph.

---

[50] M. Golumbic, *Algorithmic graph theory and perfect graphs*, Elsevier, Annals of Discrete Mathematics **57**, Amsterdam, 2004.

3.12. In Section 3.3.2 on Page 56 we showed how to compute the clique number in cographs in linear time. Design a 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.

3.13. Check that the house, hole, domino and gem depicted in Figure 3.8 on Page 58 are not distance hereditary.

3.14. 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.

3.15. Design an algorithm that constructs a DH-decomposition tree for a tree.

3.16. Is every distance-hereditary graph a comparability graphs?

3.17. Let G be distance hereditary. Prove that, for every vertex x and every integer k, the set of vertices at distance k from x induces a cograph.

3.18. Finish the proof of Theorem 3.49 on Page 63 by showing how the four values, that are mentioned in the proof, are computed for an edge e in the decomposition tree T.

3.19. Which of the graphs in Figure 3.8 on Page 58 is chordal?

3.20. Finish the proof of Theorem 3.56 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.

3.21. Let G be a chordal graph and let S be a minimal x, y-separator for non-adjacent 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

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

3.22. 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,

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

3.23. 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.

3.24. 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.

3.25. The proof of Theorem 3.66 is perhaps a bit sketchy. Rub the sleep out of your eyes and make sure that you agree with all the details.

3.26. Show that the coloring algorithm of Theorem 3.67 on Page 72 can be implemented to run in linear time.

3.27. Design a polynomial-time algorithm which checks 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?

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

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

3.29. 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

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

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

3.30. 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?

3.31. Let $G = (V, E)$ be a chordal graph. By Theorem 3.64 there exist a tree T and a collection of subtrees

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

such that any 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.

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

3.33. Check that the circled vertices in Figure 3.3 on page 45 form asteroidal triples.

3.34. Point out an asteroidal triple in each graph in Figure 3.13 on page 75.

3.35. Are permutation graphs AT-free?

3.36. Are cocomparability graphs AT-free?

3.37. 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. (See Figure 3.16 on the next page.)

(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.

(iii) Prove that circle graphs are closed under local complementations. By that we mean the following. Let G be a circle graph and let x be a vertex of G. Consider the graph G' obtained from G by replacing the graph induced by $N(x)$ by its complement.
`Hint:` Consider reversing the order of the endpoints of chords on *one* of the two parts of the circle bounded by the endpoints of the chord that represents x.

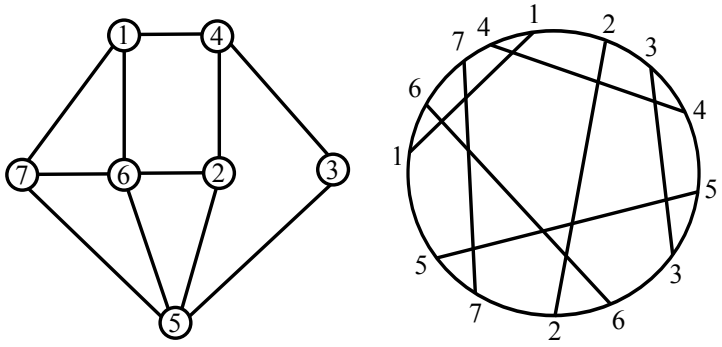


Fig. 3.16. A circle graph and its circle diagram.

4

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 one.

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 $\Omega = \{x_1, \ldots, x_k\}$ 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, unbounded 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 an unbounded, 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 4.1. *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. If a graph has n vertices and if $n \geqslant k$ then the graph has a coloring with at least k colors.

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 some other parameterized problem (Q, k'), which is hard for fixed-parameter tractability, to (P, k).

Suppose it is known that some parameterized problem (Q, k') is hard for fixed-parameter tractability. To reduce (Q, k') to (P, k) one reduces Q to P in polynomial time. The *extra condition* that is put on this reduction is, that there has to be 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 Q to P. 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.

4.1 Vertex cover

Definition 4.2. *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 4.3. *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 4.3 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 $(\omega(G), k')$ to $(\alpha(G), k)$. 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 4.3,

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

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

Here comes the surprise!

Theorem 4.4. *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 4.5. 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 $(\alpha(G), k)$ to $(\zeta(G), k')$. 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 4.6. The technique used in Theorem 4.4 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.

4.2 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 4.7. *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

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

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

Lemma 4.8. *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 4.7 and 4.8 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. (You could say that the kernel is 'the heart of the problem.')

Theorem 4.9. *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 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 4.8 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 4.4 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 4.10. *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 4.11. *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 4.12. *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 4.13. *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 4.12 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 (Page 39)

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

Via 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. Notice that, if $U \neq \varnothing$, then U separates I' and $V \setminus (U \cup I')$.

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.

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

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

$$U \cup I'$$

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. □

Remark 4.14. For every constant c there exists a polynomial-time algorithm that reduces an instance $(\zeta(G), k)$ to a kernel with at most $2k - c \log k$ vertices.[3] This seems pretty close to optimal. Unless $P = NP$ there is no kernelization which reduces the parameterized vertex cover problem to a kernel of size $(2 - \epsilon)k$ which is a subgraph of the original graph.[4]

4.3 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 4.15. *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. □

[3] M. Lampis, A kernel of order $2k - c \log k$ for vertex cover, *Information Processing Letters* **111** (2011), pp. 1089–1091.

[4] J. Chen, H. Fernau, I. Kanj and G. Xia, Parametric duality and kernelization: Lower-bounds and upperbounds on kernel size, *SIAM Journal on Computing* **37** (2008), pp. 1077–1106.

Lemma 4.16. *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 4.17. *There exists an $O(1.39^k \cdot n^2)$ 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 neighbor 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, then 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 neighbors, 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 then 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 the three vertices x, y and z 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'. In this case all vertices of

$$N[y] \cup N[z]$$

are removed from G'. By the assumption that y or z has at least one neighbor which is not a neighbor of x, this removes at least four vertices. The correctness of this step follows from Lemma 4.16.

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

Let $T(n)$ 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

$$T(n) \leqslant T(n-1) + T(n-4) + O(n^2). \tag{4.1}$$

In the fourth case, we have, apart from two greedy cases,

$$T(n) \leqslant T(n-3) + T(n-4) + O(n^2). \tag{4.2}$$

This is so because in one branch x, y and z are removed, and in the other branch the vertices of $N[y] \cup N[z]$ are removed (and this set contains at least four vertices).

The search tree has depth at most k. The solution for the Recurrence (4.1) is $T(k) = O(1.39^k \cdot n^2)$. The solution for the Recurrence (4.2) is $T(k) = O(1.23^k \cdot n^2)$. (In Exercise 4.5 we ask you to check that.) Clearly, the worst case occurs when the algorithm branches all the time as in the third case. □

Remark 4.18. The best algorithm that we know of for the parameterized vertex cover problem $(\zeta(G), k)$ runs in $O^*(1.28^k)$ and polynomial space. Here we suppressed the polynomial in n.[5] This algorithm does a more extensive case analysis. No better timebound is known, even when one allows exponential space.[6]

Remark 4.19. The Exponential Time Hypothesis assumes that k-SAT cannot be solved in $O(2^{o(k)n})$ time. It can be shown that, if $(\zeta(G), k)$ can be solved in $O(2^{o(k)} \cdot n^{O(1)})$ time then the Exponential Time Hypothesis fails.[7] [8]

[5] J. Chen, I. Kanj, and G. Xia, Improved upperbounds for vertex cover, Theoretical Computer Science **411** (2010), pp. 3736–3756.

[6] L. Chandran, F. Grandoni, Refined memorisation for vertex cover, *Information Processing Letters* **93** (2005), pp. 125–131.

[7] R. Impagliazzo and R. Paturi, On the complexity of k-SAT, *Journal of Computer and System Sciences* **62** (2001), pp. 367–375.

[8] D. Marx, *Fixed parameter algorithms.* Open lectures for PhD students in computer science, 2010, Warsaw, Poland.

4.4 Minimum fill-in

Recall Definition 3.50 on Page 64 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.

In the minimum fill-in problem one wants to add as few edges as possible to make it chordal.

Definition 4.20. *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 of a graph G is minimal if the deletion of any edge that is added to G, makes H non-chordal. If G is chordal then the only minimal embedding of G is G itself.

Definition 4.21. *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. (See Figure 4.1 on the following page.)

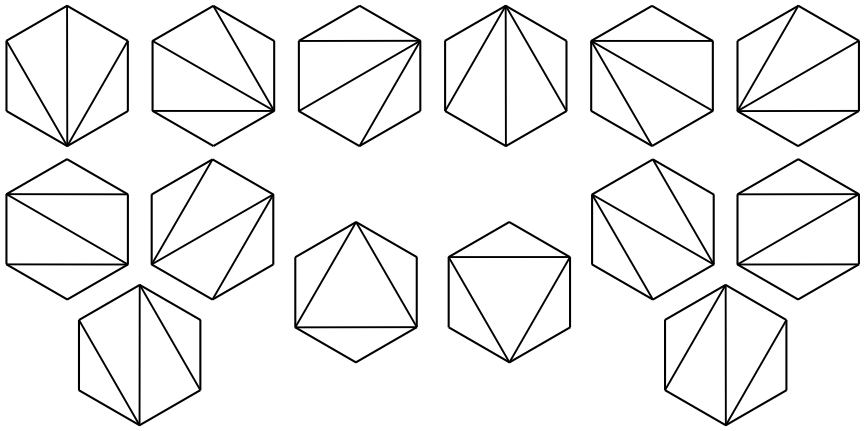


Fig. 4.1. The figure shows the different triangulations of C_6. The number of these is the Catalan number $C_4 = 14$.

Lemma 4.22. *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, but he didn't call them 'Catalan numbers!')

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 4.23. *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 path from y to z in

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

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 4.22 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 no suitable chordal embedding of G since we cannot add any edge.

By Lemma 4.22 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

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

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 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 $O((2 \cdot 4^k + 1) \cdot n^5) = O(4^k \cdot n^5)$. This proves the theorem. □

4.5 Odd cycle transversals

Many problems can be solved in polynomial time for bipartite graphs. Therefore, it is of some interest to remove a small number of vertices from the graph such that the remaining graph is bipartite.

The problem to remove a minimal number of vertices from the graph such that the remaining graph is bipartite is of course equivalent to finding a set of vertices of minimal cardinality such that each odd cycle in the graph has at least one vertex in this set. The problem is called the odd cycle transversal problem.

Definition 4.24. *Let* $G = (V, E)$ *be a graph. An odd cycle transversal in* G *is a set* $F \subseteq V$ *of vertices such that* $G - F$ *is bipartite.*

The odd cycle transversal problem asks to find a minimum set F of vertices in G such that $G - F$ is bipartite. The odd cycle transversal problem is NP-complete. In this section we show that it is fixed-parameter tractable.[9]

Given an integer k we show that there exists an $O(4^k k \cdot mn)$ algorithm that either finds an odd cycle transversal with at most k vertices or decides that there is no odd cycle transversal with at most k vertices. Here, m and n are the numbers of edges and vertices in the graph.

[9] B. Reed, K. Smith and A. Vetta, Finding odd cycle transversals, *Operations Research Letters* **32** (2004), pp. 299–301.

The algorithm is recursive. It chooses an arbitrary vertex x and recursively calls the procedure to find an odd cycle transversal with at most k vertices in $G - x$. If $G - x$ has no odd cycle transversal with at most k vertices, then neither has G. In that case we are done.

Assume that $G - x$ has an odd cycle transversal F with at most k vertices. Let

$$F' = F \cup \{x\}.$$

Lemma 4.25. *Let G be a graph and let x be a vertex of G. Let F be an odd cycle transversal in $G - x$. Then $F' = F \cup \{x\}$ is an odd cycle transversal in G.*

Proof. This is obvious. Any odd cycle in G which contains no vertex of F must contain x, otherwise it would be an odd cycle in $(G - \{x\}) - F$. Thus F' is an odd cycle transversal in G. □

If $|F'| \leqslant k$, and we are done. Otherwise

$$|F'| = k + 1.$$

In that case, the algorithm proceeds as follows. It calls a subroutine which, given a graph G and an odd cycle transversal T with $k+1$ vertices, determines if G has an odd cycle transversal with at most k vertices.

Assume that T is an odd cycle transversal of $G = (V, E)$. Let

$$\{B_1, B_2\}$$

be a partition of the vertices of $G - T$ into two color classes. Thus B_1 and B_2 induce two independent sets in $G - T$.

Define a bipartite graph

$$B = (V', E') \tag{4.3}$$

as follows. For each vertex $t \in T$ introduce two vertices t_1 and t_2 and put one in each color class of B. That is,

$$V' = (V \setminus T) \cup \{ t_1, t_2 \mid t \in T \}. \tag{4.4}$$

For each edge e in G we put the following edges in B, depending on the endpoints of e.

(i) If e is an edge in $G - T$ then e is an edge of B with the same endpoints as e.
(ii) If e connects a vertex $s \in B_i$ with a vertex $f \in T$ then the edge e connects s in B_i with the vertex f_{3-i} in B_{3-i} in B. Here $i \in \{1, 2\}$.

(iii) Assume $e = \{f, g\}$ with $\{f, g\} \subseteq T$. The graph B contains *one* edge for each edge e of this type. Either, arbitrarily, $\{f_1, g_2\}$ or $\{f_2, g_1\}$.

This completes the description of the bipartite graph B. Thus the vertices of B are partitioned into two color classes

$$B'_1 = B_1 \cup \{\, t_1 \mid t \in T \,\} \quad \text{and} \quad B'_2 = B_2 \cup \{\, t_2 \mid t \in T \,\}. \tag{4.5}$$

Definition 4.26. *Let* $Y \subseteq T$. *Write*

$$Y' = \{\, y_1, y_2 \mid y \in Y \,\}, \tag{4.6}$$

where y_i *is the copy of* y *in* B'_i, $i \in \{1, 2\}$. *A partition* $\{Y_\alpha, Y_\beta\}$ *of* Y' *is valid if*

$$|\{\, y_1,\ y_2 \,\} \cap Y_\alpha| = 1 \quad \textit{for all } y \in Y. \tag{4.7}$$

The following theorem shows when there is an odd cycle transversal in G of cardinality smaller than $|T|$.

Theorem 4.27. *An odd cycle transversal* T *is minimum if and only if for every valid partition*

$$\{\, Y_\alpha,\ Y_\beta \,\} \tag{4.8}$$

of Y', *for any* $Y \subseteq T$, *there are* $|Y|$ *vertex-disjoint paths from* Y_α *to* Y_β *in*

$$B' = B[Y_\alpha \cup Y_\beta \cup B_1 \cup B_2] \tag{4.9}$$

Proof. First assume that T is minimum. Let $Y \subseteq T$, let Y' be defined as in (4.6), let $\{Y_\alpha, Y_\beta\}$ be a valid partition of Y', and assume that there are less than $|Y|$ vertex-disjoint paths from Y_α to Y_β in

$$B' = B[Y_\alpha \cup Y_\beta \cup B_1 \cup B_2].$$

By the max flow – min cut theorem, or Menger's theorem, there exists a separator S in B' which separates Y_α from Y_β with cardinality less than $|Y|$.

Define a set of vertices $\Omega \subseteq V$ as follows. A vertex $w \in V$ is in Ω if, either

$$w \in S \quad \text{or} \quad w \in T \text{ and at least one of } w_1 \text{ and } w_2 \text{ is in } S. \tag{4.10}$$

Then we obtain a smaller odd cycle transversal in G, namely,

$$T^* = \Omega \ \cup\ (T \setminus Y). \tag{4.11}$$

To see that this is an odd cycle transversal, assume that $G - T^*$ has an odd cycle O. Then O must have a vertex $y \in Y$. The set Ω separates Y_α from Y_β in B', where B' is defined as in (4.9). One copy of y is in Y_α and the other copy

is in Y_β. Since S separates y_1 and y_2 in B', the y_1, y_2-path in B' induced by O must have a vertex in S. Then either this is a vertex of G or it is a vertex in T. In both cases O contains a vertex in Ω, by definition.

For the converse, assume that G has an odd cycle transversal U which is smaller than T. That is, we assume $|U| < |T|$. Let $\{ \tilde{B}_1, \tilde{B}_2 \}$ be a partition of the vertices of $G - U$ into two color classes.

Let $Y = T \setminus U$, and let the valid partition $\{Y_\alpha, Y_\beta\}$ be defined by

$$Y_\alpha = \{ y_1 \mid y \in Y \cap \tilde{B}_1\} \cup \{y_2 \mid y_2 \in Y \cap \tilde{B}_2 \} \tag{4.12}$$
$$Y_\beta = \{ y_2 \mid y \in Y \cap \tilde{B}_1 \} \cup \{ y_1 \mid y \in Y \cap \tilde{B}_2 \}. \tag{4.13}$$

We claim that there are not $|Y|$ vertex-disjoint paths from Y_α to Y_β in

$$B[Y_\alpha \cup Y_\beta \cup B_1 \cup B_2]. \tag{4.14}$$

To show that, we prove that $U \setminus T$ is a separator separating Y_α from Y_β. Since

$$|U \setminus T| < |T \setminus U| = |Y| \tag{4.15}$$

this will prove the claim.

Consider a chordless path P from Y_α to Y_β with vertices in

$$Y' = (Y_\alpha \cup Y_\beta \cup B_1 \cup B_2) \ \setminus \ (U \setminus T). \tag{4.16}$$

Let u and v be the endpoints of P. We may assume that P contains no other vertices of Y'.

By symmetry, we may assume that either both u and v are in $Y \cap \tilde{B}_1$ or that $u \in Y \cap \tilde{B}_1$ and $v \in Y \cap \tilde{B}_2$.

Consider the first case. Then, by (4.12) and (4.13) $u = y_1$ for some $y \in Y$ and $v = y_2'$ for some $y' \in Y$. Thus P has odd length. By definition, P contains no copy of a vertex in U. Thus P must have even length, since y_1 and y_2' are both in $\tilde{B}_1$.

Consider the second case, that is $u \in Y \cap \tilde{B}_1$ and $v \in Y \cap \tilde{B}_2$. Then $u = y_1$ for some $y \in Y$ and $v = y_1'$ for some $y' \in Y$. Then P has even length. However, $u \in \tilde{B}_1$ and $v \in \tilde{B}_2$, and so P must have odd length.

This proves the theorem. □

We now easily obtain the result.

Theorem 4.28. *For any $k \in \mathbb{N}$ there exists an $O(4^k k \cdot mn)$ algorithm which determines if a graph G can be made bipartite by deleting at most k vertices.*

Proof. For a given odd cycle transversal F' of size $k+1$ we can check if there exists a smaller one by solving a maximum flow problem for each choice of $Y \subseteq F'$ and each valid partition $\{Y_\alpha, Y_\beta\}$ of Y' as defined in (4.6). There are $O(2^k)$ choices for Y and $O(2^k)$ choices for a valid partition.

The Ford-Fulkerson algorithm takes time $O(km)$, since the value of the flow is $O(k)$. Since the subroutine is called for at most n vertices x, this proves the theorem. □

Remark 4.29. The method illustrated in this section is sometimes called iterative compression. Given a solution of small size $k+1$ one tries to 'compress' it to a solution of size k or decide that no such solution exists.

4.5.1 Feedback vertex set

As another example of the iterative compression technique we discuss the feedback vertex set problem.

Recall Definition 3.46 on page 62. Let G be a graph. A set $F \subseteq V$ is a feedback vertex set if $G - F$ is a forest. Denote by $f(G)$ the cardinality of a minimum feedback vertex set in G.

The minimum feedback vertex set problem is NP-complete. In this section we design a fixed-parameter algorithm for feedback vertex set.[10]

Let $V = \{v_1, \ldots, v_n\}$ and define, for $i \in \{1, \ldots, n\}$,

$$V_i = \{ v_1, \ \ldots, \ v_i \} \quad \text{and} \quad G_i = G[V_i]. \tag{4.17}$$

The iterative compression technique tries to find a feedback vertex set F_i in G_i of cardinality at most k as follows.

1. Let $i = 1$ and $F_1 = \varnothing$.
2. For $i \in \{2, \ldots, n\}$, let
$$F_i = F_{i-1} \cup \{v_i\}. \tag{4.18}$$
3. Assume that $|F_{i-1}| \leqslant k$. If $|F_i| \leqslant k$, then G_i has a feedback vertex set with at most k vertices. Otherwise, $|F_i| = k+1$. In that case, using iterative compression, find a new minimum feedback vertex set F_i in G_i and check if $|F_i| \leqslant k$.

[10] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier and S. Wernicke, Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, *Journal of Computer and System Sciences* **72** (2006), pp. 1386–1396.

We concentrate on the third item. For convenience we write G instead of G_i, V instead of V_i and F instead of F_i. We assume that $|F| \leqslant k+1$.

First observe that it is sufficient to describe an algorithm which solves the following problem.

Problem 1.
Instance: A graph $G = (V, E)$ and a feedback vertex set F in G of cardinality at most $k+1$.
Task: Find a minimum feedback vertex set F' such that $F' \cap F = \varnothing$.

To see that we may restrict ourselves to solving this problem, let $\hat{F}$ be a minimum feedback vertex set in G. Let

$$X = \hat{F} \cap F \quad \text{and} \quad Y = \hat{F} \setminus F. \tag{4.19}$$

Then $F \setminus X$ is a feedback vertex set in $G - X$ and Y is a minimum feedback vertex set in $G - X$ which is disjoint from $F \setminus X$.

In our algorithm we try all subsets $X \subseteq F$. Remove X from the graph and find a minimum feedback vertex set Y in $G - X$ disjoint from $F \setminus X$. Since there are 2^{k+1} possible subsets $X \subseteq F$, it is sufficient to solve the problem stated above.

We first get rid of some vertices that are unimportant for solving the problem. We present four reduction rules which reduce an instance of Problem 1 to an instance in which all vertices of $V \setminus F$ have degree at least three in G.

Lemma 4.30. *Let $\{G, F\}$ be a problem instance of Problem 1 and let x be a vertex in G of degree at most one. Then a solution to Problem 1 with instance*

$$\{G - x, F \setminus \{x\}\}$$

is a solution to Problem 1 with instance $\{G, F\}$.

Proof. If x has degree at most one then it is not contained in any cycle. Thus any feedback vertex set in $G - x$ is a feedback vertex set in G. Thus, a solution of Problem 1 for instance $\{G - x, F \setminus \{x\}\}$ is a solution to Problem 1 with instance $\{G, F\}$. □

Lemma 4.31. *Let $\{G, F\}$ be a problem instance of Problem 1. Let x be a vertex of degree two and let y and z be the two neighbors of x. Assume that $x \in F$ and assume that y and z are not adjacent. Let G' be the graph obtained from G by removing x and adding the edge $\{y, z\}$. Let Y be a solution to Problem 1 with instance*

$$\{G', F \setminus \{x\}\}.$$

Then Y is a solution to Problem 1 with instance $\{G, F\}$. If there is no solution to Problem 1 with instance $\{G', F \setminus \{x\}\}$ then there is no solution to Problem 1 with instance $\{G, F\}$.

Proof. Let Y be a solution to Problem 1 with instance $\{G', F \setminus \{x\}\}$. We first show that Y is a feedback vertex set in G, disjoint from F. Assume that $G - Y$ has a cycle C. If $x \in C$ then C contains y and z since x has two neighbors in C and y and z are the only two neighbors of x in G. Since y and z are not adjacent C must contain at least one vertex of $V \setminus \{x, y, z\}$. But then $C \setminus \{x\}$ is a cycle in $G' - Y$, which is a contradiction.

Let $\tilde{Y}$ be a feedback vertex set which solves Problem 1 with instance $\{G, F\}$. Since $x \in F$ and $\tilde{Y} \cap F = \varnothing$, $x \notin \tilde{Y}$. We show that $\tilde{Y}$ is a feedback vertex set in G'. If C were a cycle in $G' - \tilde{Y}$ then C had to contain the edge $\{y, z\}$, since otherwise C would also be a cycle in $G - \tilde{Y}$. But then $C \cup \{x\}$ would be a cycle in $G - \tilde{Y}$, which is a contradiction.

Now assume that there is no solution to problem 1 with instance $\{G', F \setminus \{x\}\}$. Then there is a cycle C in G' with all vertices in F. If C is a cycle in G then $\{G, F\}$ has no solution. Otherwise, C contains the edge $\{y, z\}$ in G'. Then $C \cup \{x\}$ is a cycle in G with all vertices in F. Thus $\{G, F\}$ has no solution. □

Lemma 4.32. *Let $\{G, F\}$ be a problem instance of Problem 1. Assume x has a vertex of degree two and let y and z be its two neighbors. Assume that y and z are adjacent. Assume that $y \notin F$. Let $\tilde{Y}$ be a solution to Problem 1 with instance*

$$\{G - x, F \setminus \{x\}\}. \tag{4.20}$$

(1) *If $y \in \tilde{Y}$ or $z \in \tilde{Y}$ then $\tilde{Y}$ is a solution to Problem 1 with instance $\{G, F\}$.*
(2) *If $y \notin \tilde{Y}$ and $z \notin \tilde{Y}$ then $Y = \tilde{Y} \cup \{y\}$ is a solution to Problem 1 with instance $\{G, F\}$.*
(3) *If there is no solution to Problem 1 with instance 4.20 then there is no solution to problem 1 with instance $\{G, F\}$.*

Proof. Let $\tilde{Y}$ be a minimum feedback vertex set in $G - x$ disjoint from $F \setminus \{x\}$. Assume that $y \in \tilde{Y}$ or $z \in \tilde{Y}$. We first show that $\tilde{Y}$ is a feedback vertex set in G, disjoint from F. Any cycle C in G which is not a cycle in $G - x$ must contain x. Furthermore, C contains y and z since these are the only two neighbors of x in G. Since $\{y, z\} \cap \tilde{Y} \neq \varnothing$, $\tilde{Y}$ has a vertex of C.

Let Y be a solution to Problem 1 with instance $\{G, F\}$. If $x \in Y$ then we may replace x in Y with y and obtain an alternative solution which does contain y and which does not contain x. Then Y is a solution to Problem 1 with instance $\{G - x, F\}$ with $y \in Y$ and then we are done. Thus we may assume that $x \notin Y$.

Since $\{x, y, z\}$ is a cycle in G, Y must contain at least one of y and z. Then Y is a solution to Problem 1 with instance $\{G - x, F \setminus \{x\}\}$, with $\{y, z\} \cap Y \neq \varnothing$. So we are done.

Now assume that $y \notin \tilde{Y}$ and $z \notin \tilde{Y}$. We claim that $\tilde{Y} \cup \{y\}$ is a solution to Problem 1 with instance $\{G, F\}$. First observe that $\tilde{Y} \cup \{y\}$ is a feedback vertex set in G disjoint from F. This is so because any cycle in G which is not a cycle in $G - x$ contains x, y and z and so it has a vertex in $\tilde{Y} \cup \{y\}$.

Let Y^* be a minimum feedback vertex set in G, disjoint from F. If $y \in Y^*$ or $z \in Y^*$ then we are done, since $Y^* \setminus \{x\}$ is a minimum feedback vertex set in $G - x$ and it contains y or z. Assume that $y \notin Y^*$ and that $z \notin Y^*$. Then $x \in Y^*$ since $\{x, y, z\}$ is a cycle in G. Now $(Y^* \setminus \{x\}) \cup \{y\}$ is also a solution to Problem 1 with instance $\{G, F\}$ and it contains y.

It is easy to check that, when (4.20) has no solution then $\{G, F\}$ has no solution either. This proves the lemma. □

Lemma 4.33. *Let $\{G, F\}$ be a problem instance of Problem 1. Assume G has a vertex x with degree two and let y and z be its neighbors. Assume that y and z are adjacent. Assume that $y \in F$ and $z \in F$. If $x \in F$ then there is no solution to Problem 1. If $x \notin F$ then let $\tilde{Y}$ be a solution to Problem 1 with instance $\{G - x, F\}$. Then $\tilde{Y} \cup \{x\}$ is a solution to Problem 1 with instance $\{G, F\}$.*

Proof. Since $y \in F$ and $z \in F$ there can only be a solution when $x \notin F$. Furthermore, any solution contains x. □

Henceforth, we may assume that all vertices in $V \setminus F$ have degree at least three in $G = (V, E)$. We now show that if G has a feedback vertex set F' disjoint from F and strictly smaller than F then the number of vertices in G is at most $15|F|$.

Lemma 4.34. *Let $G = (V, E)$ be a graph and let F be a feedback vertex set in G. Assume that every vertex of $V \setminus F$ has degree at least three in G. If G has a feedback vertex set F' with*

$$F' \cap F = \varnothing \quad \textit{and} \quad |F'| < |F| \tag{4.21}$$

then G has at most $15 \cdot |F|$ vertices.

Proof. Let $V' = V \setminus F$. Partition V' into the following three sets.

$$\begin{aligned} A &= \{x \mid x \in V' \quad \text{and} \quad |N(x) \cap F| \geqslant 2\} \\ B &= \{x \mid x \in V' \setminus A \quad \text{and} \quad |N(x) \cap V'| \geqslant 3\} \\ C &= V' \setminus (A \cup B). \end{aligned}$$

We bound the cardinality of A, B and C as follows.

We first show that $|A| < 2|F|$. Consider the bipartite graph G_A with color classes F and A and edges

$$E_A = \{\, \{a, f\} \mid a \in A \quad \text{and} \quad f \in F \quad \text{and} \quad \{a, s\} \in E \,\}. \tag{4.22}$$

If G_A is a forest then $|E_A| \leqslant |F| + |A| - 1$. Each vertex of A has at least two neighbors in F, thus

$$|E_A| \geqslant 2|A|. \tag{4.23}$$

Hence, if G_A is a forest then $|A| \leqslant |F| - 1$. If $|A| \geqslant 2|F|$ then it is clearly impossible to remove at most $|F|$ vertices from A such that G_A is acyclic.

We now show that $|B| \leqslant 2|F|$. Since F is a feedback vertex set in G, $G[V']$ is a forest. All vertices of V' that have degree at most one in $G[V']$ are in A, since every vertex of $G - F$ has degree at least three. All vertices of B have degree at least three in $G[V']$ and since $G[V']$ is a forest

$$|B| \leqslant |A| < 2|F|. \tag{4.24}$$

Each vertex of C has degree two in $G[V']$. Thus it has at least one neighbor in F. Since $C \cap A = \varnothing$, every vertex of C has exactly one neighbor in F. The graph $G[C]$ consists of paths and isolated vertices. We first bound the number of isolated vertices. Each isolated vertex connects two components of $G[A \cup B]$ and no two components of $G[A \cup B]$ are connected by more than one isolated vertex in C. Since $G[V']$ is a forest, the number of isolated vertices in C is at most

$$|A \cup B| - 1 < 4|S|. \tag{4.25}$$

We now bound the number of vertices in C that are in paths of length at least one. We claim that the number of vertices in $G[C]$ that are not isolated is at most $6|F|$. First notice that each edge of $G[C]$ creates a path between two vertices in F. Thus, if there are at least $|F|$ edges in $G[C]$ then there is a cycle in $G[C \cup S]$.

Assume that $G[C]$ has more than $3|F|$ edges. Each vertex of $G[C]$ is incident with at most two edges of $G[C]$. If we remove at most $|F|$ vertices from $G[C]$ then at least $|F|$ edges remain and then there is a cycle by the argument above. Thus $G[C]$ can have at most $3|F|$ edges, that is, at most $6|F|$ vertices of $G[C]$ are in paths of length at least one.

In total, we find that $|V'|$ is bounded by

$$|V'| = |A| + |B| + |C| < 2|F| + 2|F| + (4 + 6)|F| = 14|F|. \tag{4.26}$$

This proves the lemma. □

Lemma 4.35. *Let G be a graph and let F be a feedback vertex set in G of cardinality $k+1$. There exists an $O(c^k \cdot n^2)$ algorithm, for some constant c, which decides if there exists a feedback vertex set F' with cardinality at most k.*

Proof. The algorithm tries all 2^{k+1} subsets $X \subseteq F$ and tries to find a feedback vertex set in $G - X$ which is disjoint from $F \setminus X$.

Let $G^* = G - X$ and reduce G^* by repeated application of the reduction rules implied by Lemmas 4.30, 4.31, 4.32 and 4.33. This takes at most linear time. By Lemma 4.34, the remaining graph G' has at most $14 \cdot |F|$ vertices in $V' \setminus F$, where V' is the set of vertices of G' and where F is a feedback vertex set in G' with at most $k+1$ vertices. Hence, the algorithm needs to try at most

$$\sum_{i=0}^{k} \binom{14(k+1)}{i} < 2^{14\cdot(k+1)} \tag{4.27}$$

subsets of $V' \setminus F$.

In total, the time complexity is bounded by $O(2^{15\cdot k} \cdot n^2)$. □

By lemma 4.35 the compression step takes $O(c^k \cdot n^2)$ time. Since this step is performed at most n times, namely for the graphs G_i where $i = 1, \ldots, n$, the overall time complexity is bounded by $O(c^k \cdot n^3)$ time. This proves the following theorem.

Theorem 4.36. *The parameterized feedback vertex set problem $(f(G), k)$ can be solved in $O(c^k \cdot n^3)$ time, for some constant c.*

Remark 4.37. By a careful analysis the constant c can be lowered to $c \approx 10.6$.[11]

Remark 4.38. Thomassé proved that $(f(G), k)$ can be reduced to a kernel with at most $4k^2$ vertices.[12]

[11] F. Dehne, M. Fellows, M. Langston, F. Rosamund and K. Stevens, An $O(2^{O(k)} \cdot n^3)$ FPT algorithm for the undirected feedback vertex set problem, *Theory of Computing Systems* **41** (2007), pp. 479–492.

[12] S. Thomassé, A $4k^2$ kernel for feedback vertex set, *ACM Transactions on Algorithms* **6** (2010), pp. 32:1–8.

4.6 Homogeneous coloring of perfect graphs

Definition 4.39. *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 3.7 on Page 40 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 k and ℓ there exists a finite set of graphs $\mathcal{F}(k, \ell)$ such that a perfect graph has a (k, ℓ)-coloring if and only if it has no element of $\mathcal{F}(k, \ell)$ as an induced subgraph.[13] It follows that there is a polynomial-time algorithm to check if a perfect graph can be colored with k cliques and ℓ independent sets. Although this algorithm 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.

Recently, it was shown that classes of graphs that do not contain all complete multipartite graphs nor their complements have a (k, k)-coloring for some fixed k. Chudnovsky and Seymour proved the following result.[14]

Theorem 4.40. *Let H be a complete multipartite graph and let J be a disjoint union of cliques. There exists a k such that any graph G that does not contain H nor J as an induced subgraph has a (k, k)-coloring.*

---

[13] 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.
A. Kézdy, H. Snevily and C. Wang, Partitioning permutations into increasing and decreasing subsequences, *Journal of Combinatorial Theory, Series A* **73** (1996), pp. 353–359.

[14] M. Chudnovsky and P. Seymour, Extending the Gyárfás-Sumner conjecture. Manuscript 2012.

In the following theorem we show that the (k, ℓ)-coloring problem is fixed-parameter tractable on perfect graphs.[15]

Theorem 4.41. *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

$$H(L_P, L_Q) \leqslant k \cdot \ell' + k' \cdot \ell. \tag{4.28}$$

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

---

[15] P. Heggernes, D. Kratsch, D. Lokshtanov, V. Raman and S. Saurabh, Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing, *Proceedings SWAT'10* , Springer, LNCS 6139 (2010), pp. 334–345.

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 at least $k+1$. 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 and reduce this, if necessary, to $k+1$ vertices.

If there is a partition $\mathcal{Q}$ of G_i with k cliques and ℓ independent sets, then, by Equation (4.28),

$$H(L, L_Q) \leqslant \mu = 2k \cdot \ell + \ell. \tag{4.29}$$

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 at least $\ell + 1$ 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

$$k + 1 \leqslant k + \ell \quad \text{and} \quad \ell + 1 \leqslant k + \ell \tag{4.30}$$

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

$$(k + \ell)^{2k \cdot \ell + \ell}. \tag{4.31}$$

The graph searches for a partition $\mathcal{P}$ in G_i for $i = 1, \ldots, n$. 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. □

4.7 Color coding

Alon, *et al.*, introduced the color coding technique to obtain good randomized algorithms for various problems where one searches for specific subgraphs of

small cardinality.[16] The technique has become a popular technique for obtaining good fixed-parameter algorithms.

We illustrate the color-coding technique for finding a path of length k in a graph G between two given vertices s and t. Obviously, this problem is NP-complete, since by putting $k = n - 1$ the problem reduces to the Hamiltonian Path problem (by trying all pairs s and t).

The trick to solve this problem is very easy. First, randomly color the vertices of the graph with k colors. Say the colors are $\{1, \ldots, k\}$. Now search for a path

$$P = [p_1, \ldots, p_k] \tag{4.32}$$

where $s = p_1$ and $t = p_k$ such that p_i has color i for all $i \in \{1, \ldots, k\}$.

When G is equipped with a k-coloring then the search algorithm for a path P as above is very easy. Namely, first delete all edges from G except those that connect vertices with colors i and $i + 1$, for $i \in \{1, \ldots, k - 1\}$. Direct the remaining edges, $\{x, y\}$ as $\overrightarrow{xy}$ if x has color i and y has color $i + 1$. This makes G into a directed, acyclic graph. Next, the algorithm searches for an s, t-path in this acyclic digraph. This takes linear time using dynamic programming; simply maintain the subset of vertices with color i that can be reached from s by a colored path $[p_1, \ldots, p_i]$.

Lemma 4.42. *Assume that G has an s, t-path. Consider the algorithm which tries all permutations of the k colors and which runs the algorithm described above for each permutation. This algorithm finds an s, t-path with probability at least e^{-k}.*

Proof. Consider an s, t-path P' in G. The probability that all vertices of P' are colored with different colors is

$$\frac{k!}{k^k} > \frac{(k/e)^k}{k^k} = e^{-k}. \tag{4.33}$$

This proves the lemma. □

Theorem 4.43. *Assume that G has an s, t-path. When the algorithm described in Lemma 4.42 is ran at least $T \cdot e^k$ times, then the probability that it does not find an s, t-path is at most e^{-T}.*

[16] N. Alon, R. Yuster and U. Zwick, Color-coding, *Journal of the ACM* **42** (1992), pp. 844–856.

Proof. The probability that the algorithm of Lemma 4.42 does not find an s,t-path is at most

$$1 - e^{-k}. \tag{4.34}$$

Thus, after $T \cdot e^k$ trials, the probability that the algorithm fails to find the s,t-path is at most

$$(1-p)^{T/p} \leqslant e^{-T} \quad \text{where } p = e^{-k}. \tag{4.35}$$

□

Remark 4.44. using derandomization techniques like hashing schemes this algorithm can be made deterministic.[17]

4.8 Problems

4.1. 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:

the inverse Ackermann function $\alpha(k)$ or $\log(\log(\cdots \log(k))\cdots)$.

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

4.3. Let S be a set and let

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

be a collection of n subsets of S such that $|S_i| = 3$ for all $i \in \{1, \ldots, n\}$. 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.

[17] M. Fredman, J. Komlós and E. Szemerédi, Storing a sparse table with $O(1)$ worst case access time, *Journal of the ACM* **31** (1984), pp. 538–544.

4.4. Find a kernel for the 3-hitting set that we defined in Exercise 4.3.
Hint: Consider an element $x \in S$ that appears in at least $k^2 + 1$ subsets S_i. Design a proof, similar to the proof of Lemma 4.8, which shows that x is in any 3-hitting set with at most k elements.

4.5. Solve the recurrences in (4.1) and (4.2).

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

4.7. Prove Lemma 4.22 on Page 102.
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

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

4.8. 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?

4.9. 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?

4.10. 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.

4.11. Check the details in the proof of Theorem 4.41. Especially

(a) check Formula 4.28,
(b) check Formula 4.29, and
(c) check Formula 4.31.

4.12. Design a fixed-parameter algorithm that checks if a graph $G = (V, E)$ has a set F of at most k edges such that $H = (V, E \setminus F)$ is bipartite.
Hint: The following paper describes an $O(2^k \cdot m^2)$ algorithm.
J. Guo, J. Gramm, F. Hüffner, R. Niedermeier and S. Wernicke, Improved fixed-parameter algorithms for two feedback set problems, *Proceedings WADS 2005*, Springer, LNCS 3608 (2005), pp. 158–168.

4.13. Given a graph $G = (V, E)$. The k-maximum cut problem asks if there is a partition $\{A, B\}$ of V such that at least k edges have one endpoint in A and the other endpoint in B. Design a fixed-parameter algorithm for the k-maximum cut problem.
Hint: This is a difficult exercise. The following paper describes a reduction to

a kernel with $2k$ edges.
E. Prieto, The method of extremal structure on the k-maximum Cut problem, *Proceedings CATS 05*, In (M. Atkinson and E. Dehne, eds.) *Conferences in Research and Practice in Information Technology* **41**, Australian Computer Society, 2005.
Another method is described in this paper:
M. Mahajan and V. Raman, Parameterizing above guaranteed values: MaxSat and MaxCut, *Journal of Algorithms* **31** (1999), pp. 335–354.

5

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 3.5.1.
2. When G is a cograph then it has a cotree. We explained that in Section 3.3.1.
3. In Section 3.4.1 we explained the notion of a decomposition tree for distance-hereditary graphs.
4. Interval graphs are chordal graphs. Interval graphs have a special clique tree, which is a path. We explained that in Theorem 3.72 on Page 74.

In Section 3.1 we hinted at a decomposition tree for perfect graphs that decomposes the graph into four basic classes of graphs. Furthermore, in Chapter 3 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.

5.1 Graph minors

Let's start with a basic lemma on natural numbers.

Lemma 5.1. *Consider a sequence of natural numbers*

$$n_1,\ n_2,\ n_3,\ \ldots \tag{5.1}$$

There exists an infinite subsequence which is nondecreasing.

Proof. Define

$$I = \{\, i \in \mathbb{N} \mid \forall_{j>i}\ n_j < n_i \,\}. \tag{5.2}$$

The subsequence of (5.1) of the numbers n_i with $i \in I$ is strictly decreasing. Since it is bounded from below by 1, it is finite.

Now let

$$k = \begin{cases} 0 & \text{if } I = \varnothing, \text{ and} \\ \max\{\, i \mid i \in I \,\} & \text{if } I \neq \varnothing. \end{cases} \tag{5.3}$$

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 nondecreasing subsequence. □

Consider an infinite sequence of graphs

$$G_1,\ G_2,\ \ldots \tag{5.4}$$

Let n_i be the number of vertices of the graph G_i, for all i. By lemma 5.1, there exists an infinite subsequence of (5.4) 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 exist some integers i and j with $i < j$ such that $G_i \preceq G_j$?

Notice that, by Lemma 5.1 this is true for a sequence (5.1) 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 with 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 5.2. *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 (5.4) 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 (5.4) 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 5.3. *A graph G is a minor 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.*

Alternatively, we can define graph minors as follows. Let $G = (V, E)$ be a graph and let

$$V = \{ x_1, \ldots, x_n \}.$$

Then $G = (V, E)$ is a minor of a graph $H = (V', E')$ if there exist n subsets of vertices

$$X_i \subseteq V', \quad \text{for } i = 1, \ldots, n$$

such that

(a) $X_i \cap X_j = \varnothing$, when $i \neq j$, 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 $\{x_i, x_j\} \in E$.

See Figure 5.1 on the facing page for an illustration.

Theorem 5.4 (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.

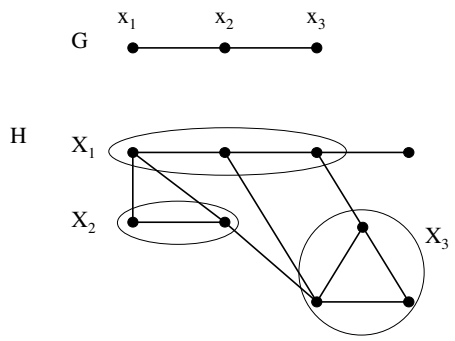


Fig. 5.1. The graph G is a minor of the graph H. Each induced subgraph $H[X_i]$ contracts to the vertex x_i. If we only contract the subsets X_i, without removing other edges, then we end up with a 'paw,' that is a triangle $\{x_1, x_2, x_3\}$ with one pendant vertex adjacent to x_1. The graph G is a subgraph of the paw.

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.

Theorem 5.5. *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 5.4, 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 $G_i \notin \Omega$. □

[1] J. Kruskal, Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, *Transactions of the American Mathematical Society* **95** (1960), pp. 210–225.

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 5.5 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.[2]

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 5.6 (The Minor Test Theorem). *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 5.6 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, because the obstruction set is finite (it has only two elements), the total time to test if a graph is planar takes $2 \cdot O(n^3) = O(n^3)$.

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 35) for graph classes that are closed under taking minors can be solved in polynomial time.

[2] K. Wagner, Über eine Erweiterung eines Satzes von Kuratowski, *Deutsche Mathematik* **2** (1937), pp. 280–285.

Theorem 5.7. *Let $\mathcal{G}$ be a class of graphs which is closed under taking minors. There exists an $O(n^3)$ algorithm to test if a graph $G \in \mathcal{G}$.*

Proof. By Theorem 5.5 the class $\mathcal{G}$ has a finite obstruction set. By Theorem 5.6 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. □

Remark 5.8. Kawarabayashi and Wollan are working on a shorter proof of Theorem 5.5. Recently, they published a shorter proof of Theorem 5.6. They claim that it shortens the proof of the graph minor algorithm by at least 200 pages.[3]

Remark 5.9. Kawarabayashi, Kobayashi and Reed recently announced an $O(n^2)$ algorithm for the minor test, that is, Theorem 5.6.[4]

5.2 Parameterized feedback vertex set

Recall Definition 3.46 on Page 62. 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. We already proved that in Section 4.5.1 on page 108 but in this section we give a much easier argument.

Lemma 5.10. *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 $G \in \mathcal{G}(k)$. We prove that every minor of G is in $\mathcal{G}(k)$.

Let F be a feedback 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.

---

[3] K. Kawarabayashi and P. Wollan, A shorter proof of the graph minor algorithm – The unique linkage theorem, *proceedings STOC'10*, ACM (2010), pp. 687–694.

[4] K. Kawarabayashi, Y. Kobayashi and B. Reed, The disjoint paths problem in quadratic time, *Journal of Combinatorial Theory, Series B* **102** (2012), pp. 424–435.

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.

This proves the lemma, since $|F'| \leqslant k$ in all cases. □

We're done! The following theorem states the result.

Theorem 5.11. *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 5.10 the class $\mathcal{G}(k)$ is minor closed. By Theorem 5.5 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 5.6 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. □

5.3 Treewidth

Recall Definition 4.20 on Page 101. Let $G = (V, E)$ be a graph. A chordal embedding of G is a chordal graph $H = (V, E')$ with $E \subseteq E'$.

Definition 5.12. *Let* $G = (V, E)$ *be a graph. The treewidth of* G *is*

$$tw(G) = \min \{\ \omega(H) - 1 \mid H \textit{ is a chordal embedding of } G\ \}. \tag{5.5}$$

Robertson and Seymour came up with the graph parameter treewidth during their research on graph minors.[5] It turns out that, if $\mathcal{G}$ is a class of graphs 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 $\mathcal{G}(\ell)$ be the class of graphs that have a feedback vertex set with at most ℓ vertices. By Lemma 5.10 this class is closed under taking minors. Furthermore, $\mathcal{G}(\ell)$ 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 increasing number of vertices in a minimum feedback vertex set. The result of Robertson and Seymour mentioned above now says that there exists a k (which is a function of ℓ) such that all graphs in $\mathcal{G}(\ell)$ have treewidth at most k.

In this section we take a close look at this important graph parameter.

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,

$$tw(T) = \begin{cases} 0 & \text{if } T \text{ has only one vertex} \\ 1 & \text{if } T \text{ has at least two vertices.} \end{cases} \tag{5.6}$$

Thus any nontrivial tree has treewidth one. This explains why we subtract one from $\omega(H)$ in Definition 5.12. 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 3.60 on page 68. Since H is chordal it has a perfect elimination ordering, say

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

By Theorem 3.66 on page 71, every maximal clique in H is one of the sets

$$\Omega_i = \{\, x_j \mid j \geqslant i \quad \text{and} \quad x_j \in N[x_i] \,\}.$$

Since $\omega(H) = 2$, we have that

[5] N. Robertson and P. Seymour, Graph minors. II. Algorithmic aspects of tree-width, *Journal of Algorithms* **7** (1986), pp. 309–322.

$$|\Omega_i| \leqslant 2 \quad \text{for all } i \in \{ 1, \ldots, n \}.$$

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. But 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 $tw(C) = 2$.

Thus the treewidth of a graph G is one if and only if G is a forest.

In Section 5.3.1 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 5.13. *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 5.13 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.

By the way, of course the class of chordal graphs is *not* closed under minors, since it is not even closed under taking subgraphs. Every graph is a subgraph (and thus also a minor) of a large enough clique!

Lemma 5.14. *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 3.64 on Page 70. Since G is chordal there exist a tree T and a collection of subtrees of T

$$\{ T_x \mid x \in V \} \tag{5.7}$$

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

$$T_{xy} = T_x \cup T_y. \tag{5.8}$$

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 5.15. *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*

$$\omega(G') \leqslant \omega(G). \tag{5.9}$$

Proof. Let Ω' be a maximal clique in G'. If $xy \notin \Omega'$ then Ω' is a clique in G and so

$$|\Omega'| \leqslant \omega(G). \tag{5.10}$$

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 (5.10) holds.
Of course, by symmetry, (5.10) 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

$$x' \in N(x) \setminus N(y) \quad \text{and} \quad y' \in N(y) \setminus N(x). \tag{5.11}$$

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 5.16. *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 5.13.

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 5.14 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 5.15 $\omega(H') \leqslant \omega(H)$.
This proves the theorem. □

Via the theory on graph minors we immediately obtain the following theorem.

Theorem 5.17. *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 5.16 the class $\mathcal{T}(k)$ of graphs with treewidth at most k is minor closed. By Theorem 5.5 there exists a finite obstruction set $\Omega(k)$ and, by Theorem 5.6 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 5.18. 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.[6] [7] In Section 5.3.3 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.

[6] T. Kloks, *Treewidth - Computations and Approximations*, Springer-Verlag, Lecture Notes in Computer Science **842**, 1994.

[7] L. Perković and B. Reed, An improved algorithm for finding tree decompositions of small width, *International Journal of Foundations of Computer Science* **11** (2000), pp. 365–371.

Remark 5.19. The obstruction set for the class of graphs with treewidth one has only one element, namely the triangle. This is so because a graph is a forest if and only if it has no cycle. Therefore, a graph is a forest if and only if it has no triangle as a minor. The obstruction set for graphs of treewidth two also has only one element, namely K_4. The following Figure 5.2 shows the obstruction set for the class of graphs with treewidth three.[8] [9] [10]

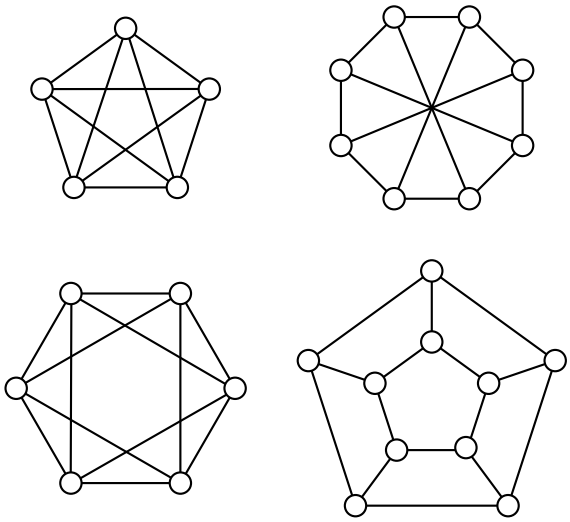


Fig. 5.2. The figure shows the obstruction set for treewidth three.

5.3.1 An algorithm for treewidth two

To get in the mood, 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 5.21.

[8] H. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, *Theoretical Computer Science* **209** (1998), pp. 1–45.

[9] S. Arnborg, A. Proskurowski and D. Corneil, Forbidden minors characterization of partial 3-trees, *Discrete Mathematics* **80** (1990), pp. 1–19.

[10] A. Satyanarayana and L. Tung, A characterization of partial 3-trees, *Networks* **20** (1990), pp. 299–322.

Let $G = (V, E)$ be a graph. The algorithm to check if the treewidth of G is at most two runs as follows.

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 5.18 we ask you to prove a similar lemma for the graphs with treewidth at most k.

Lemma 5.20. *Let $G = (V, E)$ be a graph and assume that $tw(G) \leqslant 2$. Then every subgraph of G has a vertex with at most two neighbors.*

Proof. By Lemma 5.13 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 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 chordal graph H.

By Theorem 3.60 on Page 68 there exists a perfect elimination ordering for H, say

$$\sigma = [x_1, \ldots, x_n].$$

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 5.21. *There exists an* $O(n)$ *algorithm which checks if the treewidth of a graph* G *is at most two.*

Proof. By Lemma 5.20, 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 5.13, 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 5.16, 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. □

5.3.2 k-Trees

When a graph G has treewidth k then it has an embedding into a chordal graph H with clique number $k + 1$. In this section we show that there is a special chordal graph embedding for G.

Definition 5.22. *Let* $k \in \mathbb{N} \cup \{0\}$. *A* k*-tree is a chordal graph in which*

(i) *every maximal clique has cardinality* $k+1$*, and*
(ii) *every minimal separator is a* k*-clique.*

Theorem 5.23. *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 3.60 there exists a perfect elimination ordering for H, say

$$\sigma = [x_1, \ldots, x_n].$$

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

$$\{\, x_{i+1},\ \ldots,\ x_n \,\}.$$

This step makes a $(k+1)$-clique of $\{x_{n-k}, \ldots, x_n\}$.

For $i = n-k-1$ down to 1 consider

$$N_i = N_H(x_i) \cap \{\, x_{i+1},\ \ldots,\ x_n\}.$$

Then $|N_i| \leqslant k$ since it is a clique in H of cardinality at most k.

By induction, every maximal clique in

$$H'_i = H'[\{\, x_{i+1},\ \ldots,\ x_n \,\}]$$

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. Every maximal clique in H' has cardinality $k+1$ and every minimal separator is a k-clique.

This proves the theorem. □

5.3.3 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.[11] In this section we explain their algorithm.

Recall Theorem 3.56 on Page 66 which says that a graph is chordal if and only if every minimal separator is a clique.

Lemma 5.24. *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, assume that there exists an a, b-path P 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, respectively. 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 $C_a \subseteq C$.

Assume that $S' \subseteq S$. Then $C_a = C$, $b \in S$ and $|S'| < |S|$. In that case we can remove vertices of S that have no neighbors in C. By induction there exists a vertex in C_a which is simplicial in G.

Otherwise, when S' has at least one vertex of C, then $|C_a| < |C|$. 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 5.24 there exists a perfect elimination ordering for G which eliminates all vertices of C first.

The algorithm of Arnborg, *et al.*, 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 $C_1, \ldots, C_t$ be the components of $G - S$.

[11] 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 5.25. *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.*

Proof. The proof is easy.

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 5.26. *Let* G *be a graph and let* $k \in \mathbb{N} \cup \{0\}$*. There exists an* $O(n^{k+2})$ *algorithm which checks 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 5.23 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 5.24, 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 5.18). 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. □

5.3.4 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 (see Definition 3.62 on page 68). This clique tree of the chordal embedding of the graph G is called the tree decomposition for G.

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 5.27. *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

$$\omega(G) = \max\{\, |W| \mid \exists_{M \in \mathcal{M}}\, W \subseteq M \quad \text{and} \quad G[W] \text{ is a clique} \,\}. \tag{5.12}$$

By Lemma 3.61 on Page 68, H has at most n maximal cliques, where n is the number of vertices of H.

Let

$$\sigma = [x_1, \ldots, x_n] \tag{5.13}$$

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

$$N_i = \{\, x_j \mid j \geqslant i \quad \text{and} \quad x_j \in N[x_i] \,\}. \tag{5.14}$$

Then Formula (5.12) becomes

$$\omega(G) = \max\{\, |W| \mid W \subseteq N_i, \quad i \in \{1, \ldots, n\} \text{ and } G[W] \text{ is a clique} \,\}. \tag{5.15}$$

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 data structure 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. □

5.3.5 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 5.28. *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

$$V_p = \{\, x \in V \mid \exists_i\ i \text{ is a vertex in } T_p \text{ and } x \in S_i \,\}. \tag{5.16}$$

Consider all vertex colorings of S_p. A coloring of S_p with ℓ colors is a partition

$$\mathcal{P} = \{\, Q_1,\ \ldots,\ Q_\ell \,\}$$

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 $c_1, \ldots, c_t$ be the children of p in T. For each $i \in \{1, \ldots, t\}$ 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 3.62 on Page 68 of the clique tree.)

Let $\mathcal{P} = \{Q_1, \ldots, Q_\ell\}$ 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 $\mathcal{P}' = \{Q'_1, \ldots, Q'_\ell\}$ 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. □

5.3.6 Disjoint cycles

As an example of what's in the magic box, openened by the Finite Basis Theorem 5.5 and the Minor Test Theorem 5.6, we look at the problem to find k vertex-disjoint cycles.

The problem to find a maximum collection of vertex-disjoint cycles in a graph is NP-complete. Even for chordal graphs, where the problem reduces to finding a maximum collection of vertex-disjoint triangles, the problem is NP-complete. Also for planar graphs and for bipartite graphs the disjoint cycles problem is NP-complete.

In the parameterized version of the disjoint cycles problem we want to find k vertex-disjoint cycles. Let $cp(G)$ be the maximal number of vertex-disjoint cycles in a graph G ('cp' stands for 'cycle packing').

Theorem 5.29. *The parameterized problem*

$$(cp(G), k)$$

is fixed-parameter tractable. For every natural number k *there exists an* $O(n^3)$ *algorithm to test if a graph* G *has* k *vertex-disjoint cycles.*

Proof. Let H be the graph which is the union of k triangles. We claim that a graph G has k vertex-disjoint cycles if and only if H is a minor of G.

First assume that G has k vertex-disjoint cycles, $C_1, \ldots, C_k$. Remove all vertices and edges from G except those vertices that appear in the cycles and except those edges that connect consecutive vertices in one of the cycles.

Now contract edges in the cycles such that each cycle becomes a triangle. This proves that H is a minor of G.

Now assume that H is a minor of G. Let $\{h_1, \ldots, h_{3k}\}$ be the set of vertices of H. By the alternative definition of a minor, (see Figure 5.1 on page 125) there exist a collection of disjoint sets of vertices $V_1, \ldots, V_{3k}$, one for each vertex in H, such that each $G[V_i]$ is connected, and such that there is an edge in G between some vertex in V_i and some vertex in V_j whenever h_i is adjacent to h_j in H.

Consider a triangle $\{h_1, h_2, h_3\}$ in H. Let $x_1 \in V_1$ and $x_2 \in V_1$ be two vertices in V_1 which are adjacent to some vertex in V_2 and V_3, respectively. Since $G[V_1]$ is connected, there is a path P_1 in $G[V_1]$ that connects x_1 and x_2. In the same manner we find paths P_2 and P_3 in $G[V_2]$ and $G[V_3]$ such that the union $P_1 \cup P_2 \cup P_3$ contains a cycle. Since all sets V_i are vertex-disjoint, this gives a collection of k vertex-disjoint cycles in G.

Thus, an algorithm to check if G has k vertex-disjoint cycles only needs to test if H is a minor of G. By Theorem 5.6 on page 126 this can be done on $O(n^3)$ time. □

In the rest of this section we show that the time complexity for solving the parameterized problem $(\mathrm{cp}(G), k)$ can be reduced to linear time.

A brick is a 6-cycle. An elementary h-wall, for $h \in \mathbb{N}$ and $h \geqslant 2$, consists of h levels and each level contains h bricks. An elementary wall of height 8 is depicted in Figure 5.3 on the next page. An elementary h-wall, for $h \geqslant 2$ is defined similarly. A wall is obtained from an elementary wall by subdividing some of the edges, that is, one may replace each edge in the elementary wall by an induced path of length at least one.

Theorem 5.30 (Robertson and Seymour[12]). *There exists a function*

$$f : \mathbb{N} \to \mathbb{N}$$

such that if a graph G *has treewidth at least* $f(k)$ *then* G *contains a wall of height* k *as a subgraph.*

Remark 5.31. The best-known upperbound for $f(k)$ in Theorem 5.30 is 20^{2k^5}.

Notice that a wall of height $2k$ has at least k^2 disjoint cycles.

[12] N. Robertson and P. Seymour, Graph minors. V. Excluding a planar graph, *Journal of Combinatorial Theory, Series B* **41** (1986), pp. 92–114.

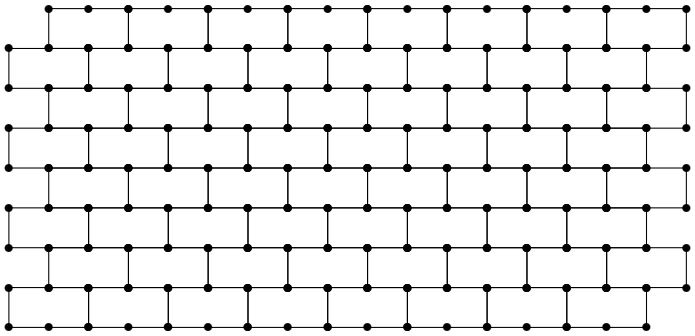


Fig. 5.3. This figure shows an elementary 8-wall.

Theorem 5.32. *For each constant* k *there exists a linear-time algorithm which checks if a graph has* k *vertex-disjoint cycles.*

Proof (Sketch). Let $k \in \mathbb{N}$. We can check in linear time whether the treewidth of G is at most $f(2k)$ (or, whether it is at most the known upperbound for $f(2k)$). If the treewidth is more than $f(2k)$ then, by Theorem 5.30, G contains a wall of height $2k$. In that case G has at least k^2 disjoint cycles. (See Remark 5.33 below to see how these cycles can be found in linear time.)

Otherwise, when the treewidth of G is at most $f(2k)$ then one can find the maximum number of vertex-disjoint cycles in G by standard dynamic programming techniques on the tree decomposition of G (that is, the clique tree of the chordal embedding of G). □

Remark 5.33. Assume that the treewidth of a graph G is more than k for some $k \in \mathbb{N}$. Perković and Reed show that there is a linear-time algorithm which computes a subgraph G' with treewidth more than k but at most $2k$.[13]

When the treewidth of G is more than $f(2k)$, one can construct the tree decomposition of a subgraph G' of G with treewidth more than $f(2k)$ but at most $2 \cdot f(2k)$. Then G' has a wall of height $2k$. Since the treewidth of G' is bounded by the constant $2 \cdot f(2k)$ one can construct this wall in linear time by dynamic programming on the tree decomposition of G'. This observation shows that the collection of k vertex disjoint cycles in G can be found in linear time (if it exists).

[13] L. Perković and B. Reed, An improved algorithm for finding tree decompositions of small width, *International Journal of Foundations of Computer Science* **11** (2000), pp. 365–371.

5.4 Pathwidth

Pathwidth is a graph parameter which is closely related to treewidth. One can see that right away from the definition.

An interval embedding of a graph $G = (V, E)$ is a graph $H = (V, E')$ which is an interval graph with $E \subseteq E'$.

Definition 5.34. *Let* $G = (V, E)$ *be a graph. The pathwidth of* G *is*

$$pw(G) = \min \{ \omega(H) - 1 \mid H \text{ is an interval embedding of } G \}. \tag{5.17}$$

Lemma 5.35. *For any graph* G*,*

$$tw(G) \geqslant pw(G). \tag{5.18}$$

Proof. Any interval graph embedding of G is a chordal embedding. □

Let's start our investigations with the easy case; trees.

Definition 5.36. *A caterpillar is a tree* T *which has a path* P *such that every other vertex of* T *is adjacent to a vertex in* P.

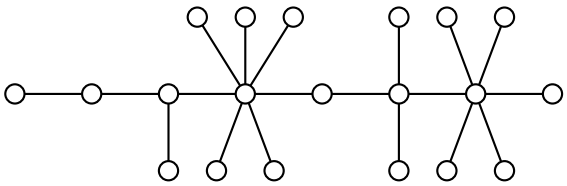


Fig. 5.4. The figure shows a caterpillar.

Lemma 5.37. *Let* T *be a caterpillar. Then*

$$pw(T) = \begin{cases} 0 & \text{if } T \text{ has one vertex} \\ 1 & \text{if } T \text{ has at least two vertices.} \end{cases} \tag{5.19}$$

Proof. If T has only one vertex then, by definition, $pw(T) = 0$ since $\omega(T) = 1$.

Assume that T is a caterpillar with at least two vertices. Then $pw(T) \geqslant 1$ since T is a chordal graph and

$$pw(T) \geqslant tw(T) = 1. \tag{5.20}$$

Notice that a caterpillar has no asteroidal triple. Therefore, by Theorem 3.77 on page 76 T is an interval graph. This proves that $pw(T) = 1$. □

Theorem 5.38. *A graph* G *has pathwidth one if and only if every component of* G *is a caterpillar and at least one component has at least two vertices.*

Proof. When G has pathwidth one then its treewidth is also one. Therefore, G is a tree with at least two vertices. If G is not a caterpillar then it has an asteroidal triple. We claim that any interval embedding of G has clique number at least three.

We may assume that G is connected and that G has at least two vertices. Let H be an interval embedding of G and assume that $\omega(H) = 2$. Then $H = G$, since adding any edge to G creates a cycle and the treewidth of a cycle is two. Since H is an interval graph it has no asteroidal triple. This implies that H is a caterpillar (see Exercise 5.20). □

We know that every cycle C has treewidth two. However, not every chordal embedding of a cycle is an interval graph. For example, a C_6 has a chordal embedding which has an asteroidal triple. However, consider the following chordal embedding of C. Take any vertex of C and make it adjacent to all other vertices. Then this is a chordal embedding without any asteroidal triple, and so it is an interval embedding with clique number three. This shows that the pathwidth of cycles is two (see Figure 5.5).

Fig. 5.5. This figure shows a 3-sun, which is a triangulation of C_6 which has an asteroidal triple. The graph on the right is a triangulation of C_6 which is an interval graph.

We want to prove that, for any integer k, the class of graphs with pathwidth at most k is closed under taking minors. For $k = 1$ this is true, since any minor of a caterpillar is a caterpillar. To prove the general result, we follow the same trajectory as for treewidth. Let's start with the following easy lemma.

Lemma 5.39. *Let G' be a subgraph of a graph G. Then*

$$pw(G') \leqslant pw(G). \tag{5.21}$$

Proof. Write $G = (V, E)$ and $G' = (V', E')$. By assumption

$$V' \subseteq V \quad \text{and} \quad E' \subseteq E. \tag{5.22}$$

Let H be an interval graph embedding of G and assume that

$$pw(G) = \omega(H) - 1.$$

Then $H[V']$ is an interval graph, since the class of interval graphs is closed under taking induced subgraph. We claim that $H[V']$ is an interval embedding of G'. Let $e = \{x, y\}$ be an edge of G'. Then x and y are both vertices of V' and $\{x, y\} \in E$. Therefore, $\{x, y\}$ is an edge of $H[V']$.

Obviously,

$$\omega(H[V']) \leqslant \omega(H) = pw(G) + 1. \tag{5.23}$$

This proves the lemma. □

Lemma 5.40. *The class of interval graphs is closed under edge contractions.*

Proof. Let G be an interval graph. Consider a collection of intervals on the real line such that G is the intersection graph of these intervals.

Let $e = \{x, y\}$ be an edge of G. Let I_x and I_y be the two intervals that represent x and y. Then

$$I_x \cap I_y \neq \varnothing \tag{5.24}$$

since x and y are adjacent. Replace the two intervals I_x and I_y by one interval I_{xy}, which is their union. The new collection of intervals represent an interval graph, say G'. We claim that G' is the graph obtained from G by contracting the edge $\{x, y\}$.

Every vertex $z \notin \{x, y\}$ which is adjacent to x or y is represented by an interval I_z which intersects I_x or I_y. Then I_z intersects I_{xy} in the new interval model. Furthermore, if z is not adjacent to x or y then I_z does not intersect I_x nor I_y. Then it has to be to the left or to the right of both I_x and I_y, since I_x and I_y intersect. Therefore, I_z does not intersect the interval I_{xy} in the model of G'.

This proves the claim and the lemma. □

Following the same path as we did in Section 5.3 we have one more step to go.

Lemma 5.41. *Let* G *be an interval graph. Let* $e = \{x, y\}$ *be an edge in* G *and let* G' *be the interval graph obtained from* G *by contracting the edge* $\{x, y\}$*. Then*

$$\omega(G') \leqslant \omega(G). \tag{5.25}$$

Proof. The proof of this lemma is the same as the proof of Lemma 5.15 on page 131. In that lemma we showed that the clique number of a chordal graph does not increase under edge contractions. Since interval graphs are chordal, we are done. □

In the following theorem we show that the class of graphs with pathwidth at most k is closed under taking minors.

Theorem 5.42. *Let* $k \in \mathbb{N} \cup \{0\}$*. Let*

$$\mathcal{P}(k) = \{\, G \mid G \textit{ is a graph and } pw(G) \leqslant k \,\}. \tag{5.26}$$

Then $\mathcal{P}(k)$ *is minor closed.*

Proof. Let G be a graph with $pw(G) \leqslant k$. Let G' be a minor of G. When G' is a subgraph of G then the claim follows from Lemma 5.39.

Assume that G' is obtained from G by contracting an edge $\{x, y\}$ in G to a single vertex xy. Let H be an interval embedding of G with $\omega(H) \leqslant k + 1$. By lemma 5.40 the graph H' obtained from H by contracting the edge $\{x, y\}$ is an interval graph. By Lemma 5.41 $\omega(H') \leqslant \omega(H)$.

We claim that the graph H' is an interval embedding of G'. This follows from the observation that any edge $\{a, b\}$ in G' is also an edge in H'.

This proves the theorem. □

Remark 5.43. By Theorem 5.5 on page 125, for any $k \in \mathbb{N} \cup \{0\}$, the class of graphs with pathwidth at most k is characterized by a finite collection of forbidden minors. For $k = 1$ this set consists of two graphs, the triangle and the smallest tree which has an asteroidal triple (see Figure 5.6). The obstruction set for graphs with pathwidth two is also known. It contains 110 graphs.[14]

[14] N. Kinnersley and M. Langston, Obstruction set isolation for the gate matrix layout problem, *Discrete Applied Mathematics* **54** (1994), pp. 169–213.

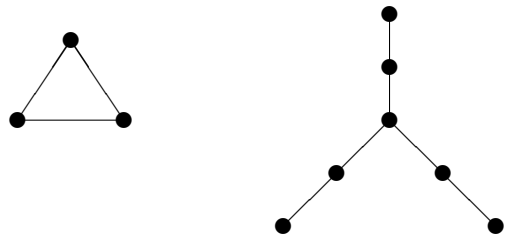


Fig. 5.6. The obstruction set for pathwidth one.

Remark 5.44. By Theorem 5.6 on page 126 the parameterized problem

$$(pw(G), k)$$

which asks if the pathwidth of a graph G is 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 pathwidth of a graph G is at most k. This algorithm is not constructive, since the graph minor theory does not provide the obstruction set. However, for each $k \in \mathbb{N} \cup \{0\}$ there is an explicit linear-time algorithm which checks if a graph G has pathwidth at most k.[15]

Remark 5.45. The pathwidth problem asks for an interval embedding of a graph G with minimal clique number. The pathwidth of a tree with n vertices can be as large as $\log n$. Scheffler showed that there is a linear-time algorithm which computes the pathwidth of a tree.[16] The pathwidth problem is also polynomial for outerplanar graphs and permutation graphs (which includes cographs). The pathwidth problem remains NP-complete for graph classes such as chordal graphs, planar graphs, bipartite distance-hereditary graphs and cocomparability graphs.

We end this section on pathwidth with an interesting observation. Recall that a chordal embedding of a graph G adds some edges to G such that the result is a chordal graph. When the deletion of any new edge creates a chordless cycle of length at least four, then the embedding is minimal.

[15] T. Kloks, *Treewidth - Computations and Approximations*, Springer-Verlag, Lecture Notes in Computer Science **842**, 1994.

[16] P. Scheffler, A linear algorithm for the pathwidth of trees. In (R. Bodendieck and R. Henn eds.) *Topics in Combinatorics and Graph Theory*, Springer Physica-Verlag (1990), pp. 613–620.

Definition 5.46. *A chordal embedding* $H = (V, E')$ *of a graph* $G = (V, E)$ *is minimal if for for any edge* $e \in E' \setminus E$ *the graph* $H' = (V, E' \setminus \{e\})$ *is not chordal.*

Theorem 5.47 (Kloks, Möhring). *Let* G *be an AT-free graph. Then every minimal chordal embedding of* G *is an interval graph. Consequently,*

$$pw(G) = tw(G). \tag{5.27}$$

Remark 5.48. There exists an elegant $O(n^3)$ algorithm which computes the treewidth of circle graphs.[17] When the graph is a permutation graph it is AT-free and then this algorithm computes the pathwidth. However, for permutation graphs there is also a linear-time algorithm which computes the pathwidth.

5.5 Rankwidth

Recall the definition of a decomposition tree (T, f) of a distance-hereditary graph $G = (V, E)$ as described in Section 3.4.1 on page 60. Thus T is a rooted binary tree and f is a bijection 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 $V \setminus W_e$.

[17] T. Kloks, Treewidth of circle graphs, *International Journal of Foundations of Computer Science* **7** (1996), pp. 111–120.

Since all vertices of Q_e have the same neighbors in $V \setminus W_e$, the cutmatrix C_e has a shape

$$C_e = \begin{pmatrix} J & 0 \\ 0 & 0 \end{pmatrix} \tag{5.28}$$

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 (5.28).

The binary field GF[2] has two elements, 0 and 1. In this field we can add elements and multiply elements. The addition in the field GF[2] is defined by the following rules.

$$0 + 0 = 0 \quad \text{and} \quad 0 + 1 = 1 + 0 = 1 \quad \text{and} \quad 1 + 1 = 0.$$

The multiplication in GF[2] is defined by

$$0 \cdot 0 = 0 \quad \text{and} \quad 0 \cdot 1 = 1 \cdot 0 = 0 \quad \text{and} \quad 1 \cdot 1 = 1.$$

Column vectors with entries in GF[2] 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].

A set of column vectors $\mathbf{a_1}, \ldots, \mathbf{a_t}$ is linearly independent if

$$\alpha_1 \cdot \mathbf{a_1} + \cdots + \alpha_t \cdot \mathbf{a_t} = \mathbf{0} \quad (\text{for some } \alpha_1, \ldots, \alpha_t \in \{0, 1\}) \quad \Rightarrow \quad \alpha_1 = \cdots = \alpha_t = 0 \tag{5.29}$$

where $\mathbf{0}$ is the all-zero vector.

The rank of a matrix over GF[2] is the maximal number of linearly independent columns.

The rank of a matrix is computed using the well-known Gauss elimination method.

Definition 5.49. *A graph* G *has rankwidth* k *if there exists a decomposition tree* (T, f) *such that every cutmatrix has rank over* $\mathrm{GF}[2]$ *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 5.50. *Let* A *be a matrix and let* k *be its rank over* $\mathrm{GF}[2]$. *Then* A *has at most* 2^k *different columns.*

Proof. Since the rank over $\mathrm{GF}[2]$ 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. □

Lemma 5.51. *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 $\mathrm{GF}[2]$ 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 $J + C_e$, which switches the zeroes and ones in C_e into ones and zeroes.

Since the rank over $\mathrm{GF}[2]$ 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. □

Let's look at an example.

Lemma 5.52. *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 3.34, a graph $G = (V, E)$ is a cograph if and only if it has a cotree (see Section 3.3.1). 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. □

Remark 5.53. Notice that the class $\mathcal{R}(k)$ is closed under creating twins.

In the same manner as in the proof of Lemma 5.52 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 5.21.

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 5.50 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.[18] We omit the description of this algorithm.

We have seen in Theorem 5.16 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, based on

Local Complementation.

Definition 5.54. *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.*

[18] P. Hliněný and S. Oum, Finding branch-decompositions and rank-decompositions, *SIAM Journal on Computing* **38** (2008), pp. 1012–1032.

Definition 5.55. *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 5.22 we ask you to prove that $\mathcal{R}(k)$ is closed under taking vertex-minors.

Oum and Seymour proved the following theorem.

Theorem 5.56. *Let* $k \in \mathbb{N} \cup \{0\}$ *and let*

$$G_1,\ G_2,\ \ldots \tag{5.30}$$

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.

5.6 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.[19]

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.[20]

[19] `Hint:` Use the alternative formulation of a minor, which appears after Definition 5.3.

[20] B. Courcelle, The monadic second-order logic or graphs. I. Recognizable sets of finite graphs, *Information and Computation* **85** (1990), pp. 12–75.

Theorem 5.57. *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 a restricted monadic second-order logic, where one does not allow quantification over subsets of edges but where one can use a test whether the cardinality of a subset of vertices is even or odd. Using this logic one can formulate, for example, whether a fixed graph H is a vertex-minor of a graph G.[21]

Remark 5.58. 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. For example, the Hamiltonian cycle problem can be formulated in monadic second-order logic but not in C_2MS-logic.

Theorem 5.59. *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.*

Remark 5.60. The Hamiltonian cycle problem can be solved in polynomial time for graphs of bounded rankwidth. However, this algorithm does not run in linear time (not even when a rank-decomposition tree is a part of the input).[22] [23]

[21] 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.

[22] E. Wanke, k-NLC graphs and polynomial algorithms, *Discrete Applied Mathematics* **54** (1994), pp. 251–266.

[23] F. Fomin, P. Golovach, D. Lokshtanov and S. Saurabh, Intractability of clique-width parameterizations, *SIAM Journal on Computing* **39** (2010), pp. 1941–1956.

5.7 Problems

5.1. Consider an infinite sequence of trees

$$T_1,\ T_2,\ \ldots \tag{5.31}$$

Let $\preceq$ denote the induced subgraph relation. Is it true that for any sequence of trees, as in (5.31) 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).

5.2. Prove the alternative definition of a minor, which appears after Definition 5.3.

5.3. Prove that, for two trees T_1 and T_2, if $T_1 \preceq_m T_2$ then there exists a sequence of edge contractions in T_2 that produces T_1.

5.4. Check that the class of all planar graphs is closed under taking minors.

5.5. 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.)

5.6. 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

$$\Omega = \{\ K_4,\ K_{2,3}\ \}.$$

5.7. 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 5.6 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.

5.8. 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.

5.9. Show that $K_{3,3}$ is a minor of the Petersen graph.

5.10. 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 $\mathcal{C}(k)$ 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?

5.11. 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 H which is in the obstruction set of $\mathcal{K}(k)$?

5.12. 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 $G \in \mathcal{D}(k)$.

5.13. Let T be a tree and let

$$\{ T_x \mid x \in V \} \tag{5.32}$$

be a collection of subtrees of T. Define a graph $G = (V, E)$ by

$$\{x, y\} \in E \quad \text{if and only if} \quad |T_x \cap T_y| \geqslant 2. \tag{5.33}$$

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?[24]

5.14. Let G be a graph. Prove that

$$\omega(G) \leqslant tw(G) + 1 \quad \text{and} \quad \chi(G) \leqslant tw(G) + 1.$$

5.15. 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 be 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.

5.16. 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 5.24.

5.17. Let G be a k-tree. Prove that every minimal separator in G is the intersection of two maximal cliques of cardinality $k+1$.

5.18. 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.

5.19. Prove that the number of edges in a k-tree is

$$\binom{k+1}{2} + (n-k-1)k = nk - \binom{k+1}{2}.$$

5.20. Let T be a tree. prove that T is an interval graph if and only if it is a caterpillar.

5.21. 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 $V \setminus W_e$.

5.22. Let $k \geqslant 1$. Prove that the class $\mathcal{R}(k)$ of graphs of rankwidth at most k is closed under taking vertex-minors.

[24] 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.

5.23. 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.

5.24. 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 $\mathcal{G}$.

5.25. 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 5.14, that is,

$$tw(G) \leqslant k \quad \text{implies that} \quad \chi(G) \leqslant k + 1.$$

5.26. 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. Bellman, *Dynamic programming*, Princeton University Press, Landmarks in Mathematics, 2012.
2. R. Niedermeier, *Invitation to fixed-parameter algorithms*, Oxford Lecture Series in Mathematics **31**, 2006.
3. R. Diestel, *Graph theory*, Springer-Verlag, Graduate Text in Mathematics **173**, 2010.
4. M. Golumbic, *Algorithmic graph theory and perfect graphs*, Elsevier, Annals of Discrete Mathematics **57**, 2004.
5. T. Kloks, *Treewidth – computations and approximations*, Springer-Verlag, Lecture Notes in Computer Science **842**, 1994.
6. J. Edmonds, A glimpse of heaven. In (J. Lenstra, A. Rinnooy Kan and A. Schrijver, eds.) *History of Mathematical Programming – A collection of personal reminiscences*, CWI Amsterdam and North-Holland, Amsterdam 1991, pp. 32–54.
7. P. Seymour, How the proof of the strong perfect graph conjecture was found, *Gazette des Mathématiciens* **109** (2006), pp. 69–83.
8. F. Fomin and D. Kratsch, *Exact exponential algorithms*, Springer-Verlag, 2010.

Index

(k, ℓ)-coloring, 114
$2K_2$, 65
3-coloring, 12
3-edge-coloring, 12
C_2MS-logic, 155
$E(G)$, 1, 2
G-decomposition, 48
$G - F$, 2
$G - N[x]$, 7
$G - S$, 2, 7
$G - e$, 2
$G - x$, 2, 7
$G = (V, E)$, 5
GF[2], 151
$G[S]$, 2, 11
K_5, 159
$K_{3,3}$, 159
$L(G)$, 2, 37
$N(C)$, 1, 59
$N(x)$, 2, 7
$N[C]$, 1
$N[x]$, 2, 7
O^*-notation, 5
P_3, 54
P_3-free graph, 84
P_4, 53
$V(G)$, 1, 2
X-partition, 14
$\bar{G}$, 2, 8
Δ, 38
$\Theta(G)$, 40
$\alpha(G)$, 5, 21, 37, 57
$\chi(G)$, 10, 37
$\gamma(G)$, 13, 18, 57
$\kappa(G)$, 4, 37
$\mathbb{N}$, 139
$\mathcal{D}(k)$, 159
$\mathcal{K}(k)$, 159
$\mathcal{R}(k)$, 152
$\mathcal{T}(k)$, 131, 154
$\nu(G)$, 38
$\omega(G)$, 8, 37, 56, 57
$\oplus$-operator, 55, 61
$\otimes$-operator, 55, 61
$\preceq_m$-ordering, 124
$\tau(G)$, 84
$\vartheta(G)$, 40
$\zeta(G)$, 93
a, b-path, 2, 137
$cp(G)$, 142
h-wall, 143
k-cover, 23
k-tree, 135
k-tree embedding, 136, 138
k-tuple, 23
t-cycle, 101
$tw(G)$, 129
x, y-separator, 65
1-1 correspondence, 68
3-hitting set, 118
3-hitting set problem, 118
4-cycle, 63
5-wheel, 159

Ackermann, W., 118
acyclic orientation, 82

addition rule, 151
adjacency, 1
adjacency matrix, 5
adjacent, 1, 32
algorithm, 8
all-ones matrix, 151
all-zero vector, 151
Alon, N., 117
antichain, 43
antihole, 39
antisymmetric, 41
arc, 79
Ariyoshi, H., 6
Arnborg, S., 137
asteroidal triple, 74
AT-free, 75
AT-free graph, 75, 150

backtrack, 94
Baker, K., 81
Bandelt, H., 60
basic classes of perfect graphs, 39
Bellman, R., 15, 163
Berge, C., 38
big deal, 90
bijection, 55
bijective map, 150
binary entropy function, 20
binary field, 151
binary rank, 152
binary tree, 7
bipartite graph, 3, 12, 21
bitvector, 115
Björklund, A., 15, 21
Blair, J., 64
Bodlaender, H., 25, 133
Boland, J., 76
boolean variable, 141
bottom line, 82
bounded search tree technique, 94
branch, 8
branch-decomposition, 154
Bron, C., 10
bucket sort, 138
Byskov, J. M., 32

Catalan number, 102
Catalan, E. C., 102
caterpillar, 145
Chandran, L., 100
charateristic equation, 32
Chen, J., 98, 100
child, 61
chord, 2, 3
chord in a circle, 87
chordal embedding, 101, 140
chordal graph, 64, 101
chordal graphs, 64
chordless cycle, 3, 73
chordless path, 2, 57, 66
chromatic index, 3
chromatic number, 3, 10, 12
Chudnovsky, M., 40, 114
circle diagram, 87
circle graph, 87
class of forests, 158
class of graphs, 35
class of planar graphs, 35
class one graph, 38
clique, 3, 8, 101
clique cover, 4, 37
  with overlapping cliques, 4
clique cover number, 37, 40
clique cover of edges, 159
clique cover problem, 37
clique number, 56
clique problem, 3, 37, 91
clique separator, 66
clique tree, 68
closed neighborhood, 1, 2, 7
closed neighborhood of set, 1
cograph, 53, 55
color class, 21, 48, 114
color coding, 116
coloring of a bipartite graph, 21
coloring of a graph, 3
coloring problem, 3, 10, 12, 35, 37, 91
column basis, 152
comparability, 41
comparability graph, 40, 79, 81, 83
comparable, 41
complement, 8, 37
complement of a graph, 2
complete bipartite graph, 115
complete graph, 82
complete multipartite graph, 115
component, 9
component of a graph, 3

connected, 53
connected component, 9
connected graph, 3, 53
consecutive clique arrangement, 74
constant time, 6
constraint satisfaction, 12
containment graph, 83
contraction of edge, 124
Corneil, D., 41, 55, 133, 137
Cornuéjols, G., 40
cotree, 55
Courcelle, B., 156, 157
crash course, 64
crossing linesegments, 80
cubic algorithm, 4
cutmatrix, 151
cycle, 3, 9
  chord of a cycle, 3
  edge of a cycle, 3
cycle packing, 142

data structure, 140
decomposition tree, 55, 121
decomposition tree for DH-graph, 60
degree, 1, 9
degree of a vertex, 2
Dehne, F., 113
deletion of edge, 124
deletion of vertex, 124
depth-first search, 36
DH-graph, 60
Diestel, R., 163
Dilworth's theorem, 44
Dilworth, R., 43
dimension of a poset, 46, 81
Dirac, G., 64
directed edge, 79
directed graph, 81
directed triangle, 82
disconnected, 55, 156
disjoint cycles, 142
disjoint separators, 65
disjoint sets, 65
distance, 57
distance-hereditary graph, 57
domatic number, 161
domatic partition, 13
domatic partition problem, 13
dominating set, 13, 17, 57
dominating set in cographs, 57
dominating set on bipartite graphs, 21
dominating set problem, 13, 17, 91
domination problem, 13, 17
domino, 58
donut, 158
Downey, R., 92
Dushnik, B., 46, 82
dynamic programming, 15, 139

edge, 1, 2
edge bipartization, 119
edge coloring problem, 3
edge contraction, 123, 124
edge set, 2
edge-intersection graph, 159
Edmonds, J., 21, 97, 163
elementary school, 118
elementary wall, 143
elimination process, 66
empty graph, 1, 2, 8
endpoints (of a path), 2
endpoints (of an edge), 2
entropy function, 20, 28
Eppstein, D., 12
equivalence class, 48
Euclidean geometry, 83
even cycle, 2
Even, S., 80
exhaustive search, 35
exponential algorithm, 5, 10
exponential space, 10
exponential time hypothesis (ETH), 100
extension, 18

fast matrix multiplication, 42
Feder, T., 114
feedback vertex set, 62, 108, 127
feedback vertex set in DH-graphs, 62
feedback vertex set problem, 63, 108
Fellows, M., 92, 113
Fernau, H., 98
Fibonacci number, 32
Fibonacci, L., 32
finite basis theorem, 125
finite class, 35
finite graph, 1
Fishburn, P., 81
fixed-parameter algorithm, 89

fixed-parameter tractable, 91
flower, 97
Fomin, F., 13, 157, 163
Ford, L., 108
Ford–Fulkerson algorithm, 108
forest, 3, 62
four color theorem, 11
Fredman, M., 118
Fulkerson, D., 108
full binary tree, 94
function intersection-model, 46

Gallai, T., 44
Gasparian, G., 37
Gauss elimination method, 152
Gauss, J. C. F., 152
gem, 58
Gilmore, P., 79
Gioan, E., 62
Golovach, P., 157
Golumbic, M., 42, 45, 47, 55, 84, 163
Grötschel, M., 40
Gramm, J., 108, 119, 160
Grandoni, F., 100
graph, 1, 5, 8
graph class, 35
graph coloring, 3, 86
graph minor, 121, 124
graph minor theorem, 124
graph minor theory, 121
Guo, J., 108, 119, 160

Hüffner, F., 108, 119, 160
Habib, M., 41, 82
Halin, R., 73, 76
Hamilton, W. R., 33
Hamiltonian cycle, 33
Hamiltonian path, 33
Hamiltonian path problem, 117
Hammer, P., 60
Hamming distance, 115
Hamming, R., 115
Heggernes, P., 115
Held, M., 15
Hell, P., 114
Helly property, 74
hereditary class, 35
Hliněný, P., 154
Hoffman, A., 79
hole, 39, 53, 58
homogeneous coloring, 114
house, 58
Howorka, E., 57
Hunt, J. W., 83
Husfeldt, T., 15, 21

Ide, M., 6
Impagliazzo, R., 100
implication class, 48
inclusion - exclusion, 15, 21
inclusion – exclusion formula, 22
incomparable, 41
independence number, 40
independent set, 3, 5
independent set in bipartite graph, 21
independent set in linegraph, 38
independent set problem, 3, 5, 6, 92
induced cycle, 64
induced path, 66
induced subgraph, 2, 7
induced subgraph relation, 122
infinite class, 35
infinite sequence, 122
infinite sequence of cycles, 123
infinite sequence of graphs, 122
infinite sequence of paths, 123
infinite sequence of trees, 158
infinite subsequence, 122
internal node, 56
internal vertex, 55, 66
intersection graph, 70
interval, 73
interval containment graph, 82
interval embedding, 145
interval graph, 72
inverse Ackermann function, 118
isolated vertex, 3, 8, 58, 85, 87
isomorphic, 122
iterative compression, 108

Jian, T., 10
join, 56, 63
join-operation, 61
joke, 158

Köhler, E., 75
König-Egerváry theorem, 38, 97
Kézdy, S., 114

Kőnig, D., 37, 38
Kanj, I., 98, 100
Karp, R., 15
Kawarabayashi, K., 127
Kerbosch, J., 10
kernel, 95, 96
kernelization, 95
Kobayashi, Y., 127
Koivisto, M., 15, 21
Komlós, J., 118
Kooten Niekerk, M. van, 25
Kratsch, D., 13, 115, 163
Kruskal, J., 124
Kuratowski theorem, 126
Kuratowski, K., 126

label, 55
Lampis, M., 98
Langston, M., 113
Lawler, E. L., 12
leaf, 3, 55
left child, 8
left endpoint, 82
left subtree, 56
Lekkerkerker, C., 76
Lempel, A., 80
length of cycle, 9
length of path, 9
Lenstra, J., 163
Lerchs, H., 55
Liedloff, M., 21
line, 1, 2
line of a tree, 3, 68
linear algorithm, 4
linear order, 41
linear-time algorithm, 9
linearly independent columns, 151
linegraph, 2, 37
linegraph of bipartite graph, 37
linesegment, 80
linked list, 5
listing algorithm, 6
Liu, X., 40
local complement, 154
logic, 155
Lokshtanov, D., 115, 157
loop, 41
Lovász number, 40
Lovász, L., 37, 40
Lueker, G., 66

Möhring, R., 150
Maffray, F., 60
Mahajan, M., 120
Marx, D., 100
matching, 3, 21, 38, 96
matrix partition, 114
max flow – min cut theorem, 106
maximal clique, 3, 69
maximal independent set, 3, 6
  of minimal cardinality, 57
maximum clique, 3, 71
maximum clique in cographs, 57
maximum cut, 119
maximum cut problem, 119
maximum flow problem, 108
maximum independent set, 3, 5, 71
maximum independent set in cographs, 57
maximum independent set problem, 10, 37
maximum matching, 21, 38, 96
maximum matching problem, 97
membership, 35
Menger's theorem, 106
Menger, K., 106
Miller, E., 46, 82
Ming, A., 102
minimal chordal embedding, 150
minimal embedding, 103
minimal separator, 65, 137
minimum extension, 18
minimum fill-in, 101
minimum fill-in problem, 101
minor, 121, 124
minor ordering, 124
minor test, 126
minor, alternative definition, 124
minor-closed class of graphs, 126
modular decomposition, 82
monadic second-order logic, 155, 156
Moon, J. W., 6
Moser, L., 6
Mulder, H., 60
multiplication rule, 151

natural number, 30, 122
neighbor, 1, 2, 5

neighborhood, 1, 2, 7
neighborhood of set, 1
new ballgame, 37
Niedermeier, R., 108, 119, 160
Niermeier, R., 163
non-chordal, 101
nondecreasing, 122
nonneighbor, 2, 54
NP-complete, 13
NP-completeness, 4, 5

obstruction set, 126
obstruction set of vertex-minors, 155
odd antihole, 39
odd cycle, 2, 38
odd cycle transversal, 104
odd cycle transversal problem, 104
odd hole, 39
ordering by induced subgraph relation, 122
orientation, 40, 48, 79
orientation of a graph, 48
Oum, S., 154, 157
outerface, 158
outerplanar graph, 158

parameterized clique problem, 89
parameterized complexity, 92
parameterized decomposition tree, 121
parameterized feedback vertex set problem, 127
parameterized graph minor problem, 126
parameterized problem, 89
parameterized reduction, 94
parent, 61
partial order, 41, 83, 122
  dimension two, 81
partially ordered set, 41, 46, 82, 83
partition of a set, 13
path, 2, 9
  chord of a path, 2
  edge of a path, 2
pathwidth, 145
Paturi, R., 100
Paul, C., 41, 62, 82
paw, 124
pendant vertex, 8, 58, 85, 87
perfect elimination ordering, 67, 134
perfect graph, 36, 114
perfect graph theorem, 37
Perković, L., 132
Perl, Y., 55
permutation diagram, 80
permutation graph, 80, 114
Petersen graph, 159
Peyton, B., 64
piece of cake, 131
planar graph, 11
Pnueli, A., 80
point, 1, 2
point of a tree, 3, 68
polynomial algorithm, 4
polynomial delay, 6
polynomial factor, 19
polynomial space, 10
poset, 41, 46, 81
poset dimension, 46
Prieto, E., 120
problem kernel, 95
proportional, 11
Proskurowski, A., 133, 137
pruned search tree, 10

quadratic algorithm, 4
quantification, 155
quantifier, 155
quasi-order, 122
quick-and-dirty, 83

Raman, V., 115, 120
rank of a matrix, 152
rank-decomposition, 154
rankwidth, 150, 152
real line, 73
realizer, 46
recognition algorithm, 21, 47
recognition of planar graph, 36
recognition problem, 35
recurrence relation, 9
reduced graph, 9
reduction rule, 109
Reed, B., 104, 127, 132
reflexive, 41
reflexive graph, 41
relation, 122
relational symbol, 155

restricted monadic second-order logic, 156
right child, 8
right endpoint, 82
right subtree, 56
rigid circuit graph, 64
Rinnooy Kan, A., 163
Roberts, F., 81
Robertson, N., 39, 124, 143
Robson, J. M., 10
Rooij, J. van, 25
rooted binary tree, 7
Rosamund, F., 113
Rose, D., 66
Rotem, D., 45

sandwich, 40
Satyanarayana, A., 133
Saurabh, S., 115, 157
Scheffler, P., 149
Schrijver, A., 40, 163
search tree, 10
search-tree for vertex cover, 99
Seese, D., 156
Seidel switch, 161
Seidel, J., 161
separator, 65
set cover, 16
set cover problem, 16
set partitioning, 15
Seymour, P., 39, 40, 114, 124, 143, 163
Shannon capacity, 40
Shannon, C., 40
shape of matrix, 151
Shirakawa, I., 6
sibling, 153
sibling leaves, 153
simplicial, 66, 134
simplicial vertex, 66
Smith, K., 104
Snevily, H., 114
square of a graph, 87
square of linegraph, 87
squeeze, 123
star, 159
star of the show, 72
Stevens, K., 113
Stewart, L., 55
Stirling formula, 20
Stirling, J., 20
Strassen's algorithm, 42
Strassen, V., 42
strong perfect graph theorem, 39
subdivision of edge, 143
subgraph, 2, 124
subgraph relation, 123
subset sum problem, 30
subtree, 56
switch, 161
symmetric closure, 48
Szemerédi, E., 118
Szymanski, T. G., 83

Tarjan, R., 66, 68
Tedder, M., 41, 82
Thomas, R., 39
Thomassé, S., 113
three-coloring, 12
time-complexity, 5
timebound, 8
topline, 82
toroidal graph, 158
torus, 158
total order, 41
transitive, 41
transitive closure, 48
transitive orientation, 41, 48, 79, 81
transitive relation, 122
traveling salesman problem, 15
traveling salesman tour, 16
tree, 3, 68
tree decomposition, 121, 139
treewidth, 128, 129
triangle, 6
triangle packing, 33
triangle packing problem, 33
triangle partition, 25
triangle partition problem, 25
triangulated graph, 66
triple, 103
trivially perfect graph, 54
Trotter, W. Jr., 83
Tsukiyama S., 6
Tung, L., 133
twin, 58, 84, 85, 87, 153
twinset, 60, 150

union, 57, 63

union of cliques, 84
union-operation, 61
Urrutia, J., 45, 83

valid bitvector, 115
valid partition, 106
Vazsonyi's conjecture, 125
vertex, 1, 2, 7
vertex coloring, 10
vertex cover, 38, 84, 92, 159
vertex cover problem, 38, 92
vertex elimination, 66
vertex ordering, 115
vertex set, 2
vertex-disjoint cycles, 159
vertex-minor, 154
Vetta, A., 104
Vizing, V., 37
Vušković, K., 40

Wagner, K., 114, 126
wall, 143
Wang, C., 114
Wanke, E., 157
Wernicke, S., 108, 119
wheel, 159
Williams, V., 42
Williamson, S., 36
Wolk, E., 55
Wollan, P., 127
worst-case timebound, 9

Xia, G., 98, 100

Yannakakis, M., 46, 68
Yuster, R., 117

zero matrix, 151
Zwick, U., 117