Foundations and Trends® in Optimization
Vol. 1, No. 4 (2014) 241–433
© 2015 L. Vandenberghe and M. S. Andersen
DOI: 10.1561/2400000006

Chordal Graphs and Semidefinite Optimization

Lieven Vandenberghe
University of California, Los Angeles
vandenbe@ucla.edu

Martin S. Andersen
Technical University of Denmark
mskan@dtu.dk

Contents

Abstract

Chordal graphs play a central role in techniques for exploiting sparsity in large semidefinite optimization problems and in related convex optimization problems involving sparse positive semidefinite matrices. Chordal graph properties are also fundamental to several classical results in combinatorial optimization, linear algebra, statistics, signal processing, machine learning, and nonlinear optimization. This survey covers the theory and applications of chordal graphs, with an emphasis on algorithms developed in the literature on sparse Cholesky factorization. These algorithms are formulated as recursions on elimination trees, supernodal elimination trees, or clique trees associated with the graph. The best known example is the multifrontal Cholesky factorization algorithm, but similar algorithms can be formulated for a variety of related problems, including the computation of the partial inverse of a sparse positive definite matrix, positive semidefinite and Euclidean distance matrix completion problems, and the evaluation of gradients and Hessians of logarithmic barriers for cones of sparse positive semidefinite matrices and their dual cones. The purpose of the survey is to show how these techniques can be applied in algorithms for sparse semidefinite optimization, and to point out the connections with related topics outside semidefinite optimization, such as probabilistic networks, matrix completion problems, and partial separability in nonlinear optimization.

L. Vandenberghe and M. S. Andersen. *Chordal Graphs and Semidefinite Optimization*. Foundations and Trends® in Optimization, vol. 1, no. 4, pp. 241–433, 2014.

DOI: 10.1561/2400000006.

1
Introduction

This survey gives an introduction to techniques from graph and sparse matrix theory that are important in semidefinite optimization. The results from graph theory we discuss are related to *chordal graphs* and *graph elimination*.

A *chordal graph* is an undirected graph with the property that every cycle of length greater than three has a chord (an edge between nonconsecutive vertices in the cycle). Chordal graphs have attracted interest in graph theory because several combinatorial optimization problems that are very difficult in general turn out to be easy for chordal graphs and solvable by simple greedy algorithms. Examples are the graph coloring problem and the problem of finding the largest clique in a graph. Chordal graphs have been studied extensively since the 1950s and their history shares some key events with the history of semidefinite optimization. In particular, it was Shannon's 1956 paper [203] that led Berge to the definition of perfect graphs, of which chordal graphs are an important subclass [28], and Lovász to one of the most famous early applications of semidefinite optimization [158].

Chordal graphs in applications often result from *graph elimination*, a process that converts a general undirected graph into a chordal graph by adding edges. Graph elimination visits the vertices of the graph in a certain order, called the *elimination order*. When vertex v is visited, edges are added between the vertices that are adjacent to v, follow v in the elimination order, and are not yet mutually adjacent. If no edges are added during graph elimination the elimination order is called a *perfect elimination order*. It has been known since the 1960s that chordal graphs are exactly the graphs for which a perfect elimination order exists. A variety of algorithms based on different forms of 'variable elimination' can be described and analyzed via graph elimination. Examples include the solution of sparse linear equations (Gauss elimination), dynamic programming (eliminating optimization variables by optimizing over them), and marginalization of probability distributions (eliminating variables by summation or integration). Variable elimination is a natural approach in many applications and this partly explains the diversity of the disciplines in which chordal graphs have been studied.

The first part of this survey (Chapters 2–7) covers the basic theory of chordal graphs and graph elimination, with an emphasis on tree data structures developed in sparse matrix theory (elimination trees and supernodal elimination trees) and efficient analysis algorithms based on elimination trees.

The second part (Chapters 8–11) describes applications of chordal graphs to sparse matrices. The sparsity pattern of a symmetric sparse matrix can be represented by an undirected graph and graph elimination describes the fill-in during Cholesky factorization of a sparse positive definite matrix. Hence, the sparsity pattern of a Cholesky factor is chordal, and positive definite matrices with chordal sparsity patterns can be factored with zero fill-in. This fact underlies several classical decomposition results, discovered in the 1980s, that characterize sparse positive semidefinite matrices, sparse matrices with a positive semidefinite completion, and sparse matrices with a Euclidean distance matrix completion, when the sparsity pattern is chordal. We give an overview of these chordal decompositions and completion problems. We also present practical algorithms for solving them and several related problems, including computing the partial inverse of a sparse positive definite matrix, the inverse factorization of the maximum determinant positive definite completion, and calculating derivatives of logarithmic barriers for cones of sparse symmetric matrices. We refer to these algorithms as *multifrontal* algorithms because they use similar recursions on elimination trees as the multifrontal Cholesky factorization algorithm. A library with implementations of most of the algorithms described in these chapters can be found at `http://cvxopt.github.io/chompack` [13].

In the last three chapters (Chapters 12–14) we discuss applications of chordal sparse matrix techniques in continuous optimization. The main focus is on decomposition results that exploit *partial separability* in nonlinear and conic optimization problems, and on techniques that exploit sparsity in interior-point methods for semidefinite optimization.

A summary of the notation used in the paper can be found on page 413.

We assume the reader is familiar with semidefinite optimization and its applications. Introductions and surveys of semidefinite optimization can be found in several books and articles, including [15, 223, 235, 24, 217, 226, 43]. On the other hand we do not assume any background in graph or sparse matrix theory. The main purpose of this survey is to give an extended introduction to the theory of chordal graphs and graph elimination, and the algorithms and elimination tree structures developed in the literature on sparse Cholesky factorization, and to describe the different ways in which these techniques can be applied in semidefinite and conic optimization algorithms. In addition, we aim to show the connections with related topics outside semidefinite optimization, such as probabilistic networks, matrix completion problems, and partial separability in nonlinear optimization.

2

Graphs

In this chapter we collect some basic definitions and results from graph theory, and introduce the notation that will be used in the paper.

2.1 Undirected graphs

An undirected graph is a pair $G = (V, E)$, with V a finite set and E a subset of $\{\{v, w\} \mid v, w \in V\}$. The elements of V are the *vertices* of the graph, the elements of E its *edges*. The notation $\{v, w\}$ for the edge between vertices v and w is used to emphasize that edges are *unordered* pairs of *distinct* vertices. Hence, edges are not directed, there are no self-loops, and at most one edge exists for every pair of vertices. We do not distinguish between $\{v, w\}$ and $\{w, v\}$.

In pictures of undirected graphs the vertices are represented by circles or rectangles and edges are shown as lines connecting the vertices. Figure 2.1 is an example with $V = \{a, b, c, d, e\}$ and

$$E = \{\{a, b\}, \{a, c\}, \{a, e\}, \{b, d\}, \{b, e\}, \{c, d\}, \{c, e\}, \{d, e\}\}.$$

The *subgraph* of G *induced by* a subset $W \subseteq V$ is the undirected graph with vertices W and all edges between vertices in W. The graph

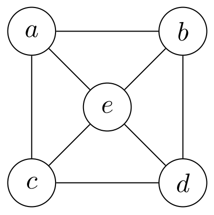


Figure 2.1: Undirected graph.

induced by W is denoted $G(W) = (W, E(W))$ and has edge set

$$E(W) = \{\{v, w\} \in E \mid v, w \in W\}.$$

One does not always distinguish between a subset of the vertices and the subgraph induced by it. It is common to refer to 'the subgraph W' instead of 'the subgraph $G(W)$' or 'the subgraph induced by W'.

Two vertices v and w are *adjacent* if $\{v, w\} \in E$. A *path* between v and $w \neq v$ is a sequence of *distinct* vertices $(v_0 = v, v_1, \ldots, v_k = w)$ with $\{v_i, v_{i+1}\} \in E$ for $i = 0, \ldots, k-1$. (Some authors omit the requirement that the vertices are distinct, and refer to a path with distinct vertices as a *simple* path.) The length of the path is the number of edges k. The vertices v_1, ..., v_{k-1} are called the *interior* vertices on the path. Two vertices of V are *connected* if there exists at least one path that has the two vertices as its end points. Note the distinction between *adjacent* vertices (connected by a path of length one) and *connected* vertices (connected by a path of any length). A graph is connected if every pair of vertices is connected. A *connected component* of a graph is a maximal subset of vertices that induces a connected subgraph.

A sequence $(v_0 = v, v_1, \ldots, v_k = v)$ with $\{v_i, v_{i+1}\} \in E$ for $i = 0, \ldots, k-1$ and $k > 1$, is called a *cycle* if no edges are repeated and the vertices v_0, ..., v_{k-1} are distinct. (Some authors allow cycles with repeated vertices and use the terms *simple cycle* or *circuit* if the vertices v_0, ..., v_{k-1} are distinct.)

A graph is *complete* if all vertices are pairwise adjacent, *i.e.*, $E = \{\{v, w\} \mid v, w \in V, v \neq w\}$. A *clique* is a set of vertices $W \subseteq V$ that induces a *maximal* complete subgraph. Many authors define a clique as any set of vertices that induces a complete subgraph, maximal or not.

In this survey we use the term *complete subgraph* for this purpose, and reserve the term clique to denote *maximal* complete subgraphs. The cliques of the graph in Figure 2.1 are $\{a, b, e\}$, $\{b, e, d\}$, $\{e, c, d\}$, and $\{a, c, e\}$.

The *neighborhood* or *adjacency set* $\operatorname{adj}(v)$ of a vertex v is the set of vertices adjacent to it:

$$\operatorname{adj}(v) = \{w \mid \{v, w\} \in E\}.$$

The set $\{v\} \cup \operatorname{adj}(v)$ is called the *closed neighborhood* of v. The *degree* of vertex v is the number of vertices adjacent to it: $\deg(v) = |\operatorname{adj}(v)|$.

2.2 Ordered undirected graphs

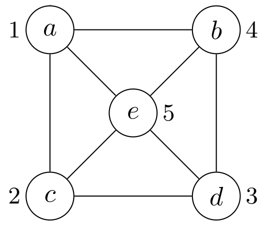
An *ordering* of an undirected graph $G = (V, E)$ is a numbering of its vertices (a bijection from $\{1, 2, \ldots, n\}$ to V if n is the number of vertices). An ordering $\sigma : \{1, 2, \ldots, n\} \to V$ can also be interpreted as a sequence of vertices $\sigma = (\sigma(1), \ldots, \sigma(n))$. We refer to $\sigma^{-1}(v)$ as the *index* of vertex v. An *ordered graph* is an undirected graph $G = (V, E)$, plus an ordering σ of its vertices. We write this as $G_\sigma = (V, E, \sigma)$. We will often use inequality notation $v \prec_\sigma w$ for $\sigma^{-1}(v) < \sigma^{-1}(w)$ and $v \preceq_\sigma w$ for $\sigma^{-1}(v) \leq \sigma^{-1}(w)$. The subscripts in $\prec_\sigma$ and $\preceq_\sigma$ are omitted if the ordering σ is clear from the context.

To visualize an ordered graph G_σ, we annotate the unordered graph G with the vertex indices $\sigma^{-1}(v)$. It is also convenient to represent ordered graphs as triangular tables or arrays, with the vertex names on the diagonal and a dot in row i and column j of the table if $i > j$ and the vertices $\sigma(i)$ and $\sigma(j)$ are adjacent. An example is shown in Figure 2.2.

Two types of monotone neighborhoods can be distinguished in an ordered graph:

$$\begin{aligned}
\operatorname{adj}^+(v) &= \{w \in \operatorname{adj}(v) \mid w \succ v\}, \\
\operatorname{adj}^-(v) &= \{w \in \operatorname{adj}(v) \mid w \prec v\}.
\end{aligned}$$

The first set is called the *higher neighborhood* or *higher adjacency set* of v; the second set is the *lower neighborhood* or *lower adjacency set.*



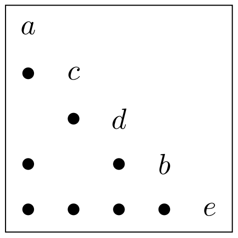


Figure 2.2: Ordered undirected graph.

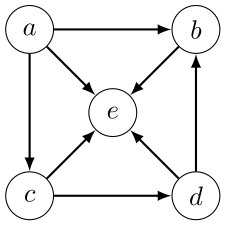


Figure 2.3: The ordered graph of Figure 2.2 as a directed acyclic graph.

We also define the *higher* and *lower degrees* as

$$\deg^+(v) = |\mathrm{adj}^+(v)|, \qquad \deg^-(v) = |\mathrm{adj}^-(v)|.$$

In the array representation of ordered graphs the higher and lower neighborhoods of v are easily identified. The indices of the elements of $\mathrm{adj}^-(v)$ are the column indices of the entries in row $\sigma^{-1}(v)$ of the array; the indices of $\mathrm{adj}^+(v)$ are the row indices of the entries in column $\sigma^{-1}(v)$. In the example of Figure 2.2, $\mathrm{adj}^-(c) = \{a\}$ and $\mathrm{adj}^+(c) = \{d, e\}$. We will use the notation $\mathrm{col}(v)$ and $\mathrm{row}(v)$ for the *closed* monotone neighborhoods

$$\mathrm{col}(v) = \{v\} \cup \mathrm{adj}^+(v), \qquad \mathrm{row}(v) = \{v\} \cup \mathrm{adj}^-(v).$$

An ordered undirected graph can be converted into a directed graph by orienting the edges from lower to higher index; see Figure 2.3. (A directed graph is a pair (V, E) with E a set of ordered tuples (v, w), with $v, w \in V$ and $v \neq w$.) Such a directed graph is necessarily *acyclic*, *i.e.*, it contains no directed cycles. Conversely, it is always possible to interpret a directed acyclic graph as an ordered undirected graph by numbering the vertices so that the edges go from lower to higher la-

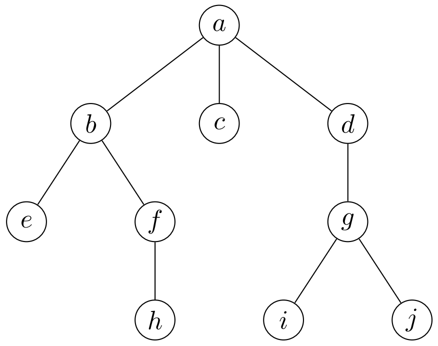


Figure 2.4: Rooted tree.

bel. To see this, we first note that every finite directed acyclic graph has at least one *source* vertex, *i.e.*, a vertex with only outgoing edges, and at least one *sink*, *i.e.*, a vertex with only incoming edges. (This follows from the observation that a directed path $(v_0, v_1, \dots, v_k)$ with $(v_i, v_{i+1}) \in E$ of maximum length starts at a source and ends at a sink. Such a path exists if the graph is acyclic and has a finite number of vertices.) We can define a numbering σ of the vertices as follows. We take $\sigma(1) = v$ where v is any source vertex, then remove v and its incident edges from the graph, pick a source vertex w in the reduced graph, set $\sigma(2) = w$, et cetera. Continuing in this way, we obtain a numbering with the desired property.

2.3 Trees

An undirected graph is a *tree* if it is connected and does not contain any cycles. A unique path exists between any two vertices of a tree (there is a path because the tree is connected; the path is unique because the concatenation of two distinct paths between the same vertices would create a cycle). An undirected graph is a *forest* if it does not contain cycles. Each connected component in a forest is a tree. A *rooted tree* is a tree with a special vertex designated as the root of the tree. Rooted trees are usually displayed with the root at the top, as in Figure 2.4.

If v is a non-root vertex and w is the first vertex on the path from v to the root, then w is called the *parent* of v and v is a *child* of w.

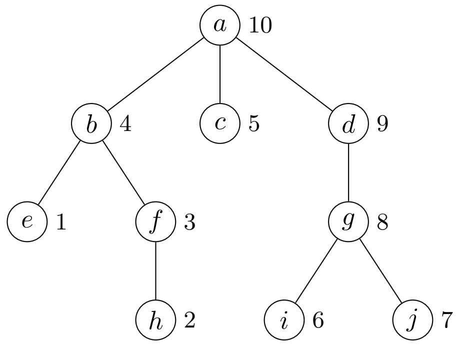


Figure 2.5: A postordering of the tree in Figure 2.4.

We often use the convention that the parent of the root is itself, so that the parent function is a function $p : V \to V$. Parents of degree k, with k a nonnegative integer, are defined recursively as

$$p^0(v) = v, \qquad p^{k+1}(v) = p(p^k(v)).$$

In the example of Figure 2.4, we have $p(h) = f$, $p^2(h) = b$, $p^3(h) = a$. If $w = p^k(v)$ for some $k \geq 0$ then w is an *ancestor* of v and v is a *descendant* of w. An ancestor w of v is a *proper ancestor* if $w \neq v$. A descendant w of v is a *proper descendant* if $w \neq v$. The set of children of v is denoted $\mathrm{ch}(v)$. A childless vertex is called a *leaf* of the tree.

The *depth* or *level* $\mathrm{lev}(v)$ of a vertex v in a rooted tree is its distance to the root (the number of edges in the unique path that connects v to the root). If $\mathrm{lev}(v) = k$ then $p^k(v)$ is the root of the tree. The depth can be defined recursively: $\mathrm{lev}(v) = 0$ if v is the root; for the other vertices $\mathrm{lev}(v) = \mathrm{lev}(p(v)) + 1$.

An ordering σ of a rooted tree is a *topological* ordering if $v \prec_\sigma p(v)$ for all non-root vertices v. If there are n vertices, then in a topological ordering the root has index n and every other vertex has a lower index than its parent. A *postordering* is a topological ordering in which the descendants of a vertex are given consecutive numbers: if $\sigma^{-1}(v) = j$ and v has k proper descendants, then the proper descendants of v are numbered $j-k$, $j-k+1$, $\ldots$, $j-1$. A postordering can be generated by assigning the indices n, $n-1$, $\ldots$, 1 in decreasing order during a depth-first search traversal of the tree. An example is shown in Figure 2.5.

Other useful information that can be gathered during a depth-first search traversal includes, for each vertex v, the level $\operatorname{lev}(v)$, the number of descendants, and the *first descendant* $\operatorname{fdesc}(v)$, *i.e.*, the descendant with the lowest index. Several common questions simplify considerably when a postordering is used. For example, to check whether a vertex w is a proper descendant of v one only needs to verify the inequalities $\operatorname{fdesc}(v) \preceq w \prec v$.

2.4 Path decomposition

In this section we discuss a specialized topic that will be important only in §6.4. Suppose we are given a family of subtrees (connected subgraphs) of a large rooted tree T. Each subtree is specified by its extreme vertices: its root and leaf vertices. We are interested in efficient algorithms for two problems. First, the problem of computing the size (number of edges) of each subtree, and second, the problem of computing the number of subtrees each vertex of T belongs to. We will describe efficient methods with a complexity that is nearly linear in the total number of extreme vertices of the subtrees [94], [65, chapter 4]. (In contrast, straightforward algorithms based on traversing all the subtrees have a complexity proportional to the total number of edges.) We assume T is described by its parent function $p(v)$, that a postordering of T is given, and that we know the level $\operatorname{lev}(v)$ of every vertex v in T.

The *least (nearest) common ancestor* $\operatorname{lca}(v, w)$ of two distinct vertices v, w in a rooted tree is the deepest vertex that is an ancestor of v and w, *i.e.*, the first vertex on the intersection of the path from v to the root and the path from w to the root. The complexity of finding the least common ancestors of a set of m vertex pairs of a tree with n vertices is almost linear in $n + m$ [214, page 698].

We first consider a single subtree R of T. Assume R has $k + 1$ extreme vertices, sorted as $v_1 \prec v_2 \prec \cdots \prec v_{k+1}$ in the postordering of T, so $v_1, \ldots, v_k$ are leaves and v_{k+1} is the root of R. For $i = 1, \ldots, k$, we define $P(v_i)$ as the set of edges on the path from v_i to the least common ancestor $\operatorname{lca}(v_i, v_{i+1})$ of v_i and v_{i+1}. (For the vertex v_k, the

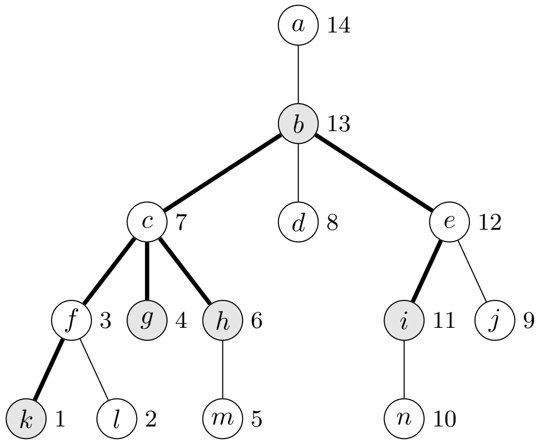


Figure 2.6: A tree T and a subtree R with root b and leaves k, g, h, i. The number next to each vertex is its index in a postordering.

edges in $P(v_k)$ form the path from v_k to the root v_{k+1} of R, since $\text{lca}(v_k, v_{k+1}) = v_{k+1}$.) This is illustrated in Figures 2.6 and 2.7. The sets $P(v_1)$, …, $P(v_k)$ form a partition of the edges in R [94, lemma 2.1], called the *path decomposition* of R. To see this, consider an edge $\{u, p(u)\}$, where u is a non-root vertex of R. Let v_i be the highest leaf vertex that is a descendant of u, *i.e.*, the highest leaf that satisfies $\text{fdesc}(u) \preceq v_i \preceq u$. We show that the edge $\{u, p(u)\}$ is contained in $P(v_i)$ and in no other set $P(v_j)$, $j \neq i$. (For example, in Figure 2.6, the edge $\{c, b\}$ is contained in $P(h)$ because h is the highest among the leaves that are descendants of c.) If $i < k$ then, by definition of i, the leaf v_{i+1} is not a descendant of u. Therefore $u \prec \text{lca}(v_i, v_{i+1})$ and $p(u) \preceq \text{lca}(v_i, v_{i+1})$. Hence $\{u, p(u)\} \in P(v_i)$. If $i = k$, then u is on the path from v_k to the root and therefore $\{u, p(u)\} \in P(v_k)$. This shows that $\{u, p(u)\} \in P(v_i)$. We now show that $P(v_i)$ is the only set in the partition that contains $\{u, p(u)\}$. Suppose $P(v_j)$, $j \neq i$, also contains the edge $\{u, p(u)\}$. The leaf v_j is a descendant of u and not the highest descendant among the leaves of R (the highest descendant is v_i). Therefore $v_j \prec v_{j+1} \preceq u$. In a postordering this means that v_{j+1} is a descendant of u. Therefore $\text{lca}(v_j, v_{j+1}) \preceq u$. This contradicts the assumption that $\{u, p(u)\} \in P(v_j)$.

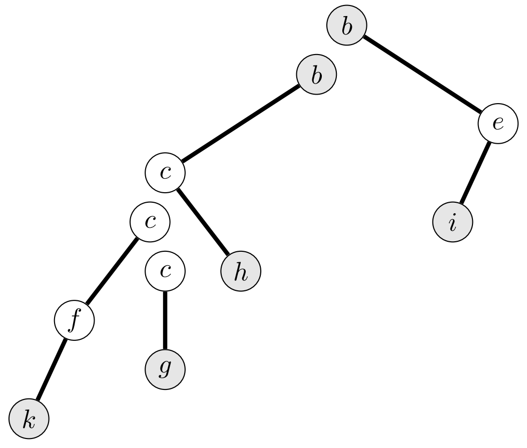


Figure 2.7: Path decomposition of the subtree in Figure 2.6. The edges are partitioned in four sets $P(k)$, $P(g)$, $P(h)$, $P(i)$ that form paths in the subtree. Each of the first three paths starts at a leaf (k, g, h, respectively) and ends at the least common ancestor of that leaf and the next leaf in the postordering. The fourth path $P(i)$ starts at the highest leaf vertex (vertex i) and ends at the root of the subtree.

We now make two observations about the path decomposition that will become useful when we apply them to a family of subtrees. First, the number of edges m_R of R is equal to the sum of the sizes of the sets $P(v_i)$ in the partition. The number of edges of $P(v_i)$ is the difference of the levels of its end points. Therefore

$$m_R = \sum_{i=1}^{k} (\operatorname{lev}(v_i) - \operatorname{lev}(\operatorname{lca}(v_i, v_{i+1}))) \,. \tag{2.1}$$

To evaluate the sum we only need to compute the least common ancestors $\operatorname{lca}(v_i, v_{i+1})$, $i = 1, \ldots, k$.

Second, consider the following function on the vertices of T:

$$\begin{aligned} \eta_R(v) &= \delta_R(v) - \sum_{u \in \operatorname{ch}(v)} \delta_R(u) \\ &= \delta_R(v) - \text{the number of children of } v \text{ in } R \end{aligned}$$

where $\operatorname{ch}(v)$ is the set of children of v in T and $\delta_R(v)$ is the 0-1 indicator function of R,

$$\delta_R(v) = \begin{cases} 1 & v \text{ is a vertex of } R \\ 0 & \text{otherwise.} \end{cases}$$

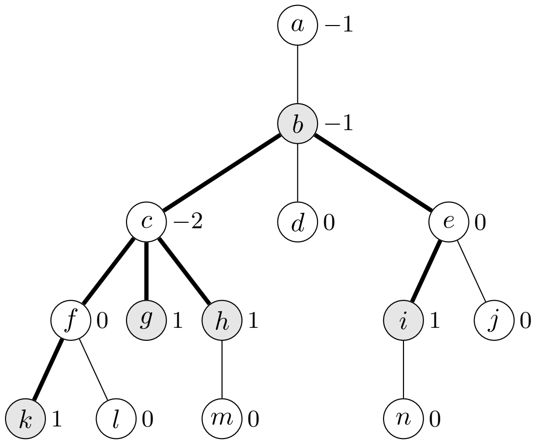


Figure 2.8: The number next to each vertex v is the function $\eta_R(v)$ for the subtree R of Figure 2.6. If v is a vertex in R, then $\eta_R(v)$ is defined as one minus the number of children of v in R. For the parent of the root of R, $\eta_R(v) = -1$. For all other vertices, $\eta_R(v) = 0$.

Summing the function η_R in topological order (*i.e.*, visiting every vertex before its parent) gives $\delta_R(v)$:

$$\delta_R(v) = \sum_{u \in T_v} \eta_R(u)$$

where $T_v = \{u \mid \text{fdesc}(v) \preceq u \preceq v\}$ is the set of descendants of v in T. These definitions are illustrated in Figures 2.8 and 2.9.

The function $\eta_R(v)$ is readily computed from the path decomposition. If v is a vertex in R, and not the root or a leaf, then $\eta_R(v)$ is the negative of the number of paths in the path decomposition that end at v:

$$\eta_R(v) = -\left|\{i \in \{1, 2, \ldots, k-1\} \mid v = \text{lca}(v_i, v_{i+1})\}\right|. \qquad (2.2)$$

This expression is also valid if v is the root of R. In that case it is equal to one minus the number of paths that end at v, because the last path $P(v_k)$ is not counted in (2.2). We can therefore compute η_R as follows. We initialize $\eta_R(v)$ as $\eta_R(v) = 0$ for all v. We add one to $\eta_R(v_i)$ for $i = 1, \ldots, k$. For $i = 1, \ldots, k-1$, we subtract one from $\eta_R(\text{lca}(v_i, v_{i+1}))$. If v_{k+1} is not the root of T, we also subtract one from $\eta_R(p(v_{k+1}))$.

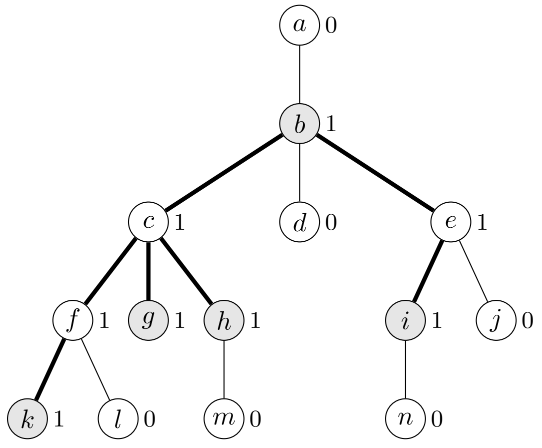


Figure 2.9: The number next to each vertex is $\delta_R(v)$, the 0-1 indicator function of R. This number can be obtained as the sum of the values of $\eta_R(u)$ over all descendants u of v in T.

Now consider a family $\mathcal{R}$ of subtrees of T, with each subtree specified by its extreme vertices. Path decompositions for all subtrees in $\mathcal{R}$ can be computed with a complexity that is almost linear in the total number of extreme vertices [214]. From the path decompositions, we immediately obtain the number of edges in each subtree, via the formula (2.1). We can also compute, for each vertex v of T, the number of subtrees v belongs to. This number is given by

$$\delta(v) = \sum_{R \in \mathcal{R}} \delta_R(v) = \sum_{u \in T_v} \sum_{R \in \mathcal{R}} \eta_R(u).$$

The function $\eta(v) = \sum_{R \in \mathcal{R}} \eta_R(v)$ is easily computed from the path decompositions. We initialize $\eta(v)$ as zero and enumerate the subtrees $R \in \mathcal{R}$. For each R we add $\eta_R(v)$ to $\eta(v)$ via the simple rule described above. Finally we compute $\delta(v)$ from $\eta(v)$ by a recursion on the vertices of T. We enumerate the vertices in topological order and for each vertex v, compute

$$\delta(v) = \eta(v) + \sum_{u \in \mathrm{ch}(v)} \delta(u).$$

3

Chordal Graphs

3.1 Chordal graph

A *chord* in a path of an undirected graph is an edge between non-consecutive vertices on the path. A chord in a path $(v_0, v_1, \ldots, v_k)$ is an edge $\{v_i, v_j\}$ with $|j - i| > 1$. One can think of a chord as a one-edge 'shortcut'. If a path $(v, v_1, \ldots, v_k, w)$ from v to w has chords, then it can be reduced to a shorter chordless path $(v, v_{i_1}, \ldots, v_{i_j}, w)$ with $1 \le i_1 < i_2 < \cdots < i_j \le k$ and $0 \le j < k$, *i.e.*, a path that skips some or all of the interior vertices v_i. Clearly, a shortest path between two vertices must be chordless, but a chordless path is not necessarily a shortest path. Figure 3.1 illustrates the definition.

A chord in a cycle $(v_0, v_1, \ldots, v_{k-1}, v_0)$ is an edge $\{v_i, v_j\}$ with $(j - i) \bmod k > 1$ (Figure 3.1). An undirected graph is *chordal* if every cycle of length greater than three has a chord. Using chords to take shortcuts, every cycle $(v_0, v_1, \ldots, v_{k-1}, v_0)$ of length greater than three in a chordal graph can therefore be reduced to a cycle of length three, *i.e.*, a triangle (v_0, v_i, v_j, v_0) with $1 \le i < j \le k - 1$. The last graph in Figure 3.1, for example, is not chordal, because the cycle (a, b, d, c, a) has length four but does not have a chord. Adding the edge $\{b, c\}$ or $\{a, d\}$ would make the graph chordal.

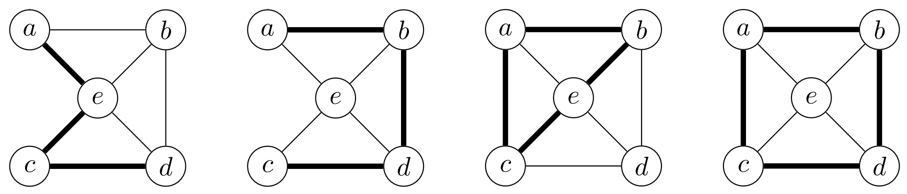


Figure 3.1: The edge $\{e, d\}$ is a chord in the path (a, e, c, d) of the first graph. The path (a, b, d, c) is chordless. The edge $\{a, e\}$ is a chord in the cycle (a, b, e, c, a) of the second graph. The cycle (a, b, d, c, a) is chordless.

We note the useful property that subgraphs $G(W) = (W, E(W))$ of a chordal graph $G = (V, E)$ are chordal. This is an immediate consequence of the definition, because cycles and chords in the subgraph $G(W)$ are also cycles and chords in G.

The fundamental theory of chordal graphs was developed in the 1960s and 1970s, and while the term 'chordal' is common today, several other names have appeared in the literature. Chordal graphs have been called *rigid-circuit graphs* [74, 88, 46], *triangulated graphs* [27, 193, 101], *perfect elimination graphs*, *decomposable graphs* [146, 58], and *acyclic graphs* [149].

Two complementary approaches to the theory of chordal graphs exist. The first approach, reviewed in §3.3 and §3.4, focuses on properties of vertex separators and cliques in chordal graphs. The second approach emphasizes the role of chordal graphs in sparse matrix factorizations and is discussed in §4. Surveys of the theory of chordal graphs can be found in the articles [193, 38] and the book [101].

3.2 Examples

Examples of graphs that satisfy the definition of chordal graph in a trivial way are the complete graphs, trees and forests (undirected graphs with no cycles), and undirected graphs with no cycles of length greater than three (such graphs are sometimes called *cactus graphs* [27, page 160]).

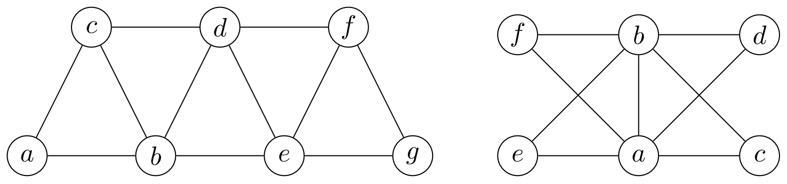


Figure 3.2: Two 2-trees. Vertices are named in the order they were created in the recursive definition.

***k*-trees.** A more interesting class of chordal graphs are the *k-trees*, which are defined recursively as follows [193, 194]. Every k-tree has at least k vertices and the only k-tree with k vertices is the complete graph. To construct a k-tree with $n > k$ vertices, one adds a vertex to a k-tree with $n - 1$ vertices in such way that the new vertex has degree k and its neighborhood is complete. Figure 3.2 shows two 2-trees constructed as prescribed in the definition, starting with the complete graph of two vertices a, b. As the name suggests, k-trees generalize ordinary trees (the 1-trees).

Chordality of k-trees can be shown by induction. A k-tree with k vertices is chordal because it is complete. Suppose it is true that all k-trees with $n - 1$ vertices or less are chordal, and that G is a k-tree with n vertices. Suppose G was constructed by adding the vertex v to a k-tree G' with $n - 1$ vertices in such a way that $\mathrm{adj}(v)$ is complete and $\deg(v) = k$. All cycles in G that do not pass through v are cycles in G'. By the induction assumption these cycles have a chord if they have length greater than three. A cycle of length greater than three in G that passes through v must contain at least two vertices in $\mathrm{adj}(v)$ (namely, the two neighbors of v in the cycle). By construction, $\mathrm{adj}(v)$ is complete, so these two vertices are adjacent and define a chord in the cycle.

Interval graphs. Let V be a finite family of sets. The *intersection graph* of V is the undirected graph (V, E) with $E = \{\{v, w\} \mid v, w \in V, v \cap w \neq \emptyset\}$. The intersection graph of a family of intervals on the real line is called an *interval graph*. Figure 3.3 shows an example.

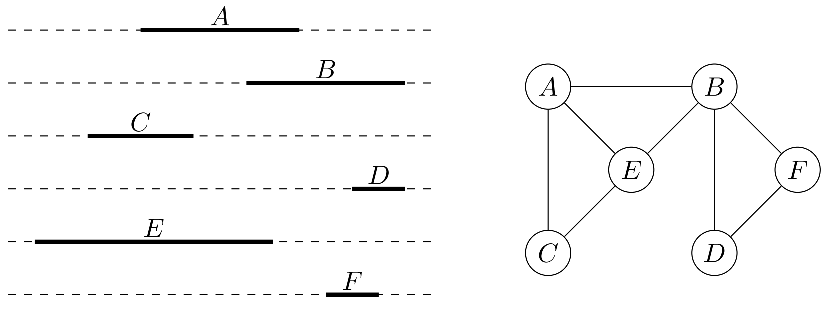


Figure 3.3: Interval graph.

Interval graphs have been studied since the 1960s [149, 88, 82] and often arise in scheduling problems. Here, each interval represents the period of time occupied by a certain activity; edges indicate activities that overlap in time [101, 67].

All intervals graphs are chordal [101, page 15]. Indeed, a chordless cycle of length greater than three would correspond to a sequence of four or more intervals on the real line, such that consecutive intervals in the sequence intersect, non-consecutive intervals do not intersect, and the last interval is identical to the first. This is clearly impossible. To prove this more formally, suppose there exists a chordless cycle $(v_0, v_1, \ldots, v_{k-1}, v_k = v_0)$ of length $k \geq 4$. Choose points $a_i \in v_i \cap v_{i+1}$ for $i = 0, \ldots, k-1$. This is possible, since $v_i \cap v_{i+1} \neq \emptyset$. Moreover the points a_0, …, a_{k-1} are distinct because non-consecutive intervals v_i, v_j in the cycle have an empty intersection. Now consider any three points a_{i-1}, a_i, a_{i+1} for $0 < i < k-1$. Since $a_i \neq a_{i-1}$ we either have $a_i > a_{i-1}$ or $a_i < a_{i-1}$. Suppose $a_i > a_{i-1}$. Then we must have $a_{i+1} > a_i$, because otherwise the interval (a_{i+1}, a_i), which is a subset of v_{i+1} because its two endpoints are in v_{i+1}, and the interval (a_{i-1}, a_i), which is a subset of v_{i-1}, intersect, contradicting the assumption that v_{i+1} and v_{i-1} do not intersect. Similarly, one shows that if $a_i < a_{i-1}$ then $a_{i+1} < a_i$. Applying this recursively, we see that either $a_0 < a_1 < \cdots < a_{k-1}$ or $a_0 > a_1 > \cdots > a_{k-1}$. However, by the same argument applied to a_{k-1}, a_0, a_1 we find that $a_{k-1} < a_0$ implies $a_0 < a_1$ and $a_{k-1} > a_0$ implies $a_0 > a_1$, which leads to a contradiction.

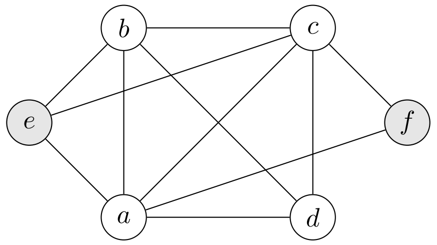


Figure 3.4: Split graph. The vertex set can be partitioned in two sets: the first set $\{a, b, c, d\}$ induces a complete subgraph; the second set $\{e, f\}$ is independent.

Later we will encounter a more general class of chordal intersection graphs, the tree intersection graphs (§3.7). We will also see that every chordal graph can be represented as a tree intersection graph.

Split graphs. A graph $G = (V, E)$ is a *split graph* if its vertices can be partitioned in two sets such that one set induces a complete subgraph and the other set is independent, *i.e.*, its vertices are mutually non-adjacent. Therefore the complement of a split graph, the graph $G^c = (V, E^c)$ with edge set

$$E^c = \{\{v, w\} \mid v, w \in V, \{v, w\} \notin E\},$$

is also a split graph. Figure 3.4 shows an example.

Threshold graphs form a class of split graphs [101, 163]. A graph $G = (V, E)$ is a threshold graph if there exists a function $f : V \to \mathbf{R}$ such that $\{u, v\} \in E$ if and only if $f(u) + f(v) > 0$.

It is easy to see that split graphs are chordal. In fact a graph is a split graph if and only if it is chordal and has a chordal complement [101, theorem 6.3].

3.3 Minimal vertex separator

A set $S \subset V$ is a *vertex separator* for two vertices v, w of an undirected graph $G = (V, E)$, or simply a *vw-separator*, if the vertices v, w are in different connected components of the subgraph $G(V \setminus S)$. A vertex separator for two vertices v, w is a *minimal* vertex separator if no proper subset of it is also a vw-separator. For example, in the graph of

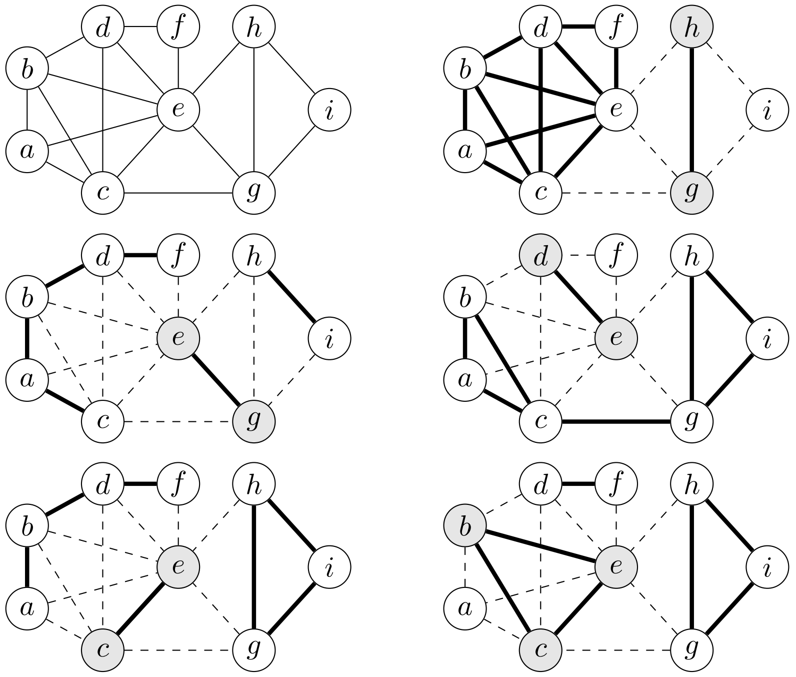


Figure 3.5: A chordal graph and its five minimal vertex separators.

Figure 3.5, $\{c, e\}$ is a minimal ag-separator and $\{b, c, e\}$ is a minimal ad-separator. As the example shows, a minimal vertex separator can contain another minimal vertex separator, if they are vertex separators for different vertex pairs. Minimal vertex separators are also called *relatively minimal cutsets* [74, 46].

Our first theorem is perhaps the most fundamental result in the theory of chordal graphs and is due to Dirac [74, theorem 1].

Theorem 3.1. A graph is chordal if and only if all minimal vertex separators are complete.

Proof. Suppose every minimal vertex separator in $G = (V, E)$ is complete. Consider a cycle of length greater than three. We show that the cycle has a chord. Let v and w be two non-consecutive vertices in the cycle. If v and w are adjacent, the edge $\{v, w\}$ is a chord. If v and w are not adjacent, there exists a vw-separator (for example, $S = V \setminus \{v, w\}$)

and every vw-separator must contain at least two other non-consecutive vertices of the cycle. Let S be a minimal vw-separator and let $x, y \in S$ be two non-consecutive vertices in the cycle different from v, w. By assumption, S is complete, so x, y are adjacent. The edge $\{x, y\}$ is a chord in the cycle.

Conversely, suppose S is a minimal vw-separator in a chordal graph $G = (V, E)$. Let C_v and C_w be the connected components of $G(V \setminus S)$ that contain v and w, respectively. Consider two arbitrary vertices $x, y \in S$. We show that x and y are adjacent. Since S is a minimal vertex separator, x is adjacent to a vertex in C_v and to a vertex in C_w (otherwise $S \setminus \{x\}$ would still be a vw-separator, contradicting the assumption that S is minimal). For the same reason, y is adjacent to a vertex in C_v and to a vertex in C_w. Choose a path $P_v = (x, v_1, \ldots, v_k, y)$ of minimum length with $k \geq 1$ interior vertices v_i in C_v, and a path $P_w = (y, w_1, \ldots, w_l, x)$ of minimum length with $l \geq 1$ interior vertices in C_w. Concatenate the two paths to create a cycle $(x, v_1, \ldots, v_k, y, w_1, \ldots, w_l, x)$ of length at least four. Since G is chordal the cycle has a chord. We show that the only possible chord is $\{x, y\}$. First, since C_v and C_w are different connected components of $G(V \setminus S)$, there are no edges $\{v_i, w_j\}$. Moreover, since P_v was chosen to have minimum length, there exist no edges $\{v_i, v_j\}$ for $|i - j| > 1$, $\{v_i, x\}$ for $i > 1$, or $\{v_i, y\}$ for $i < k$. Similarly, since P_w has minimum length, there are no edges $\{w_i, w_j\}$ for $|i - j| > 1$, $\{w_i, x\}$ for $i < l$, or $\{v_i, y\}$ for $i > 1$. This leaves the edge $\{x, y\}$ as only possible chord. □

Note that in the second half of the proof we actually have $k = l = 1$, since $(x, v_1, \ldots, v_k, y, x)$ and $(y, w_1, \ldots, w_l, x, y)$ are chordless cycles, so they have length three. Therefore $x, y \in \operatorname{adj}(\hat{v}) \cap \operatorname{adj}(\hat{w})$ for $\hat{v} = v_1 \in C_v$ and $\hat{w} = w_1 \in C_w$. The following theorem strengthens this result and states that there exist vertices $\hat{v} \in C_v$ and $\hat{w} \in C_w$ that are adjacent to *all* vertices in the minimal vw-separator S. This result appears in [46].

Theorem 3.2. Let S be a minimal vw-separator in a chordal graph. Denote by C_v and C_w the connected components of $G(V \setminus S)$ that contain v and w, respectively. Then there exist vertices $\hat{v} \in C_v$, $\hat{w} \in C_w$ such that $S = \operatorname{adj}(\hat{v}) \cap \operatorname{adj}(\hat{w})$.

Proof. First note that if S is a vw-separator (in any graph, chordal or not) then $\mathrm{adj}(\hat{v}) \cap \mathrm{adj}(\hat{w}) \subseteq S$ for all $\hat{v} \in C_v$ and $\hat{w} \in C_w$. It is therefore sufficient to show that a minimal vw-separator S in a chordal graph satisfies $S \subseteq \mathrm{adj}(\hat{v}) \cap \mathrm{adj}(\hat{w})$ for some $\hat{v} \in C_v$ and $\hat{w} \in C_w$. Define $\hat{v}$ and $\hat{w}$ as the points in C_v and C_w that have the maximum numbers of neighbors in S. We show that $\hat{v}$ and $\hat{w}$ are adjacent to every vertex in S. The proof is by contradiction. Suppose there is a point x in S not adjacent to $\hat{v}$. Choose a path $P = (x, v_1, \ldots, v_k, \hat{v})$ of minimum length with interior vertices $v_i \in C_v$. We have $k \geq 1$ because x is not adjacent to $\hat{v}$. Also there are no chords in the path because it has minimum length. Choose an arbitrary $y \in \mathrm{adj}(\hat{v}) \cap S$ and make P into a cycle $(x, v_1, \ldots, v_k, \hat{v}, y, x)$ by adding the edges $\{\hat{v}, y\}$ and $\{y, x\}$. The cycle has length at least four, so it has a chord. Since P is chordless and $\{x, \hat{v}\} \notin E$, all chords are between y and a vertex v_j. In particular, $\{y, v_1\}$ is a chord. Since the point y was chosen arbitrarily in $\mathrm{adj}(\hat{v}) \cap S$, we have shown that the vertex v_1 is adjacent to all vertices in $\mathrm{adj}(\hat{v}) \cap S$. However v_1 is also adjacent to $x \notin \mathrm{adj}(\hat{v})$, and this contradicts the choice of $\hat{v}$ as a vertex in C_v with the maximum number of neighbors in S. We conclude that if $\hat{v} \in C_v$ has the maximum number of neighbors in S of all the vertices in C_v, then it is adjacent to all vertices $x \in S$, *i.e.*, $S \subseteq \mathrm{adj}(\hat{v})$. By the same argument $S \subseteq \mathrm{adj}(\hat{w})$ and therefore $S \subseteq \mathrm{adj}(\hat{v}) \cap \mathrm{adj}(\hat{w})$. □

We note the following important consequence of Theorems 3.1 and 3.2 [46, lemma 2.3]. Since the sets $S \cup \{\hat{v}\}$ and $S \cup \{\hat{w}\}$ define complete subgraphs, and $\hat{v}$ and $\hat{w}$ are not adjacent, the minimal vertex separator S is contained in at least two cliques: there is at least one clique that contains $S \cup \{\hat{v}\}$ and at least one clique that contains $S \cup \{\hat{w}\}$. Cliques that contain $S \cup \{\hat{v}\}$ have vertices in $S \cup C_v$; cliques that contain $S \cup \{\hat{w}\}$ have vertices in $S \cup C_w$.

3.4 Simplicial vertex

A vertex v of an undirected graph is *simplicial* if its neighborhood $\mathrm{adj}(v)$ is complete. The closed neighborhood $\mathrm{adj}(v) \cup \{v\}$ of a simplicial vertex v is a clique and it is the only clique that contains v. To see

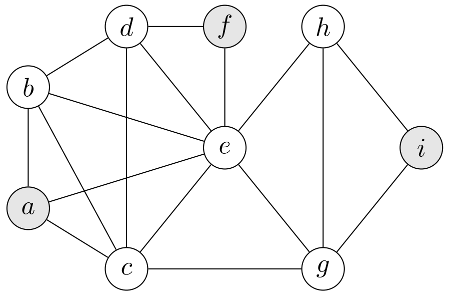


Figure 3.6: The simplicial vertices in the graph of Figure 3.5 are a, f, i. Note that these are the vertices that do not appear in any of the minimal vertex separators of Figure 3.5.

this, simply note that every complete vertex set W that contains v is included in $\operatorname{adj}(v) \cup \{v\}$. Therefore if $\operatorname{adj}(v) \cup \{v\}$ is complete, it is the unique maximal complete subgraph that contains v, *i.e.*, a clique. This characterization of simplicial vertices holds regardless of whether the graph is chordal or not.

If v is not simplicial and x, y are non-adjacent vertices in $\operatorname{adj}(v)$, then v is in every xy-separator. Every non-simplicial vertex therefore belongs to at least one minimal vertex separator. This holds for all graphs, chordal or not. However, for chordal graphs, the converse also holds. If there exists a minimal vertex separator that contains v, then, as we saw at the end of the previous section, v is contained in at least two cliques, so it is not simplicial. The vertices of a chordal graph can therefore be partitioned in two groups: the simplicial vertices, which belong to a unique clique, and the elements of minimal vertex separators, which belong to at least two different cliques [193, corollary 2]. This is illustrated in Figure 3.6.

The following two theorems on the existence of simplicial vertices are due to Dirac [74, theorem 4] and Rose [193, lemma 6].

Theorem 3.3. Every chordal graph has at least one simplicial vertex. Every non-complete chordal graph has at least two non-adjacent simplicial vertices.

Proof. The proof is by induction on the number of vertices n. If $n = 1$ the graph is complete and the unique vertex is simplicial. Suppose

$G = (V, E)$ is a chordal graph with $n \geq 2$ vertices and the theorem holds for chordal graphs with less than n vertices. If G is complete, all vertices are simplicial and adjacent. Suppose G is not complete and v, w are two non-adjacent vertices. Let S be a minimal vw-separator and denote by C_v, C_w the connected components of $G(V \setminus S)$ containing v and w, respectively. By Theorem 3.1, S is complete. The two graphs $G_v = G(C_v \cup S)$ and $G_w = G(C_w \cup S)$ are chordal with less than n vertices. Consider G_v. By the induction hypothesis G_v is either complete and all its vertices are simplicial (in G_v) or it has two non-adjacent vertices that are simplicial (in G_v). In the second case, at least one of the simplicial vertices must be in C_v because S is complete and the simplicial vertices are not adjacent. Since no vertex in C_v is adjacent to a vertex in another connected component of $G(V \setminus S)$, a vertex of C_v that is simplicial in G_v is also simplicial in G. We conclude that C_v contains at least one vertex $\hat{v}$ that is simplicial (in G). By the same argument, C_w contains a simplicial vertex $\hat{w}$. Moreover $\hat{v}$ and $\hat{w}$ are not adjacent because C_v and C_w are different connected components of $G(V \setminus S)$. $\square$

Theorem 3.4. If W is a complete subgraph of a non-complete chordal graph G, then G has a simplicial vertex outside W.

This is an immediate consequence of Theorem 3.3. The graph G has at least two non-adjacent simplicial vertices, and if W is complete, at least one simplicial vertex must be outside W.

3.5 Clique tree

A clique tree of a graph $G = (V, E)$ is a tree which has the cliques of G as its vertices. A clique tree T has the *induced subtree property* if for every $v \in V$, the cliques that contain v form a subtree (connected subgraph) of T. We will use the notation R_v for this subtree (or, equivalently, for the vertices of this subtree, *i.e.*, the set of cliques that contain v). An example is shown in Figure 3.7.

Buneman [46, theorem 2.7] and Gavril [92, theorem 3] have shown that chordal graphs are exactly the graphs for which a clique tree with the induced subtree property exists. In this section we show one di-

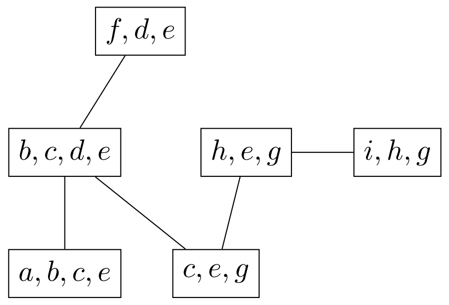


Figure 3.7: Clique tree with the induced subtree property for the graph of Figure 3.6.

rection of this result and discuss some of its implications. The other direction is shown in §3.7.

We will show that when applied to a chordal graph, the following recursive algorithm returns a clique tree with the induced subtree property. The algorithm is 'conceptual' because at this point we are not concerned with practical details. Practical methods for computing clique trees will be discussed in Chapter 4.

Algorithm 3.1 (Clique tree).

Input. A chordal graph $G = (V, E)$.

Output. A clique tree T for G with the induced subtree property.

Algorithm. If V is a singleton, return $T = (\{V\}, \emptyset)$. Otherwise, find a simplicial vertex v of G and define the clique $W = \operatorname{adj}(v) \cup \{v\}$. Construct a clique tree T' with the induced subtree property for $G' = G(V \setminus \{v\})$. Distinguish two cases.

Case 1. If $\operatorname{adj}(v)$ is a clique of G', define T as the tree T' with W substituted for $\operatorname{adj}(v)$.

Case 2. If $\operatorname{adj}(v)$ is not a clique of G', let W' be any clique of G' that contains $\operatorname{adj}(v)$. Define T as the tree T' with the vertex W and an edge $\{W, W'\}$ added.

Note that we do not assume that G is connected. If G is not connected, then at some point during the recursion we encounter a simplicial vertex v with $\operatorname{adj}(v)$ empty. Case 2 then applies and since $\operatorname{adj}(v) = \emptyset$ is contained in all cliques of G', we make $W = \{v\}$ adjacent to an arbitrary clique W' of G'.

Theorem 3.5. Every chordal graph has a clique tree with the induced subtree property.

Proof. We use induction on the number of vertices to show correctness of Algorithm 3.1. If $n = 1$, the algorithm returns the clique tree $T = (\{V\}, \emptyset)$ which certainly has the induced subtree property. Suppose that G has n vertices and the algorithm is correct for chordal graphs with less than n vertices. As in the algorithm outline, let T' be a clique tree with the induced subtree property for the subgraph $G' = G(V \setminus \{v\})$, where v is a simplicial vertex of G.

We first verify that the vertices of T are the cliques of G. Since v is a simplicial vertex, $W = \operatorname{adj}(v) \cup \{v\}$ is a clique of G and it is the only clique of G that contains v. All other cliques of G must be cliques of G'. If W' is a clique of G' and not equal to $\operatorname{adj}(v)$, then it includes at least one vertex that is not adjacent to v. Therefore $W' \cup \{v\}$ is not a complete subgraph of G, and W' is also a clique of G. If $W' = \operatorname{adj}(v)$ is a clique of G', then it is not a clique of G because it is strictly contained in the clique W. We conclude that the cliques of G are W plus all the cliques of G' with one possible exception: if $\operatorname{adj}(v)$ happens to be a clique of G', then it is not a clique of G. The vertices of T generated by the algorithm are therefore exactly the cliques of G.

Next we show that T satisfies the induced subtree property. Let R'_u be the set of cliques in G' that contain a given vertex $u \in V \setminus \{v\}$. By the induction assumption each R'_u forms a tree. Let R_u be the set of cliques in G that contain a given vertex $u \in V$. We need to show that R_u is a tree for all $u \in V$. First consider $u = v$. The only clique of G that contains v is W, therefore $R_v = \{W\}$, a single-vertex tree. Next consider a vertex $u \in V \setminus \operatorname{adj}(v)$. We have $R_u = R'_u$ which is a tree by assumption. Finally, consider $u \in \operatorname{adj}(v)$. We distinguish the two cases in the algorithm. In case 1, R_u is the tree R'_u with $\operatorname{adj}(v)$ replaced by W. In case 2, R_u is the tree R'_u with the vertex W added and made adjacent to a vertex W' of R_u. In both cases, R_u is a tree. $\square$

From Algorithm 3.1 we obtain a simple bound on the number of cliques in a chordal graph. Since in each cycle of the algorithm a different vertex v is considered and at most one new clique is added, the number of cliques is bounded by the number of vertices n. (In fact, the only chordal graph with n vertices and n cliques is the graph with n isolated vertices. A connected chordal graph has at most $n-1$ cliques.)

The induced subtree property is also referred to as the *running intersection property* [38] because it can be rephrased as follows. If two cliques W and W' intersect, then $W \cap W'$ is included in all the cliques on the path in the clique tree between W and W'. This is easily seen to be equivalent to the induced subtree property. For every vertex v in the intersection of W and W', the cliques W and W' are in the induced subtree R_v. Therefore all cliques on the path between W and W' are also in the subtree.

3.6 Rooted clique tree

Clique trees provide very useful information about the structure of the graph. For example, the edges in the clique tree provide a complete list of the minimal vertex separators. This is most easily explained in terms of a rooted clique tree. Let T be a clique tree with the induced subtree property. Pick an arbitrary clique as the root of T and denote the parent function in the rooted clique tree as $p_{\mathrm{c}}(W)$. Each non-root clique W can then be partitioned in two sets:

$$\mathrm{sep}(W) = W \cap p_{\mathrm{c}}(W), \qquad \mathrm{res}(W) = W \setminus \mathrm{sep}(W). \tag{3.1}$$

For the root clique W we define $\mathrm{res}(W) = W$. The sets $\mathrm{sep}(W)$ will be called the *clique separators* and the sets $\mathrm{res}(W)$ the *clique residuals*. These definitions are illustrated in Figure 3.8 for the clique tree in Figure 3.7. The two rows shown at each vertex of the clique tree correspond to the partitioned clique: the top row is $\mathrm{sep}(W)$, the bottom row is $\mathrm{res}(W)$.

Two important properties follow immediately from the induced subtree property and the definition of clique separators and residuals. For future reference we state them as a theorem.

Theorem 3.6. Let $G = (V, E)$ be a chordal graph and let T be a rooted clique tree with the induced subtree property. Define $\mathrm{sep}(W)$ and $\mathrm{res}(W)$ as in (3.1).

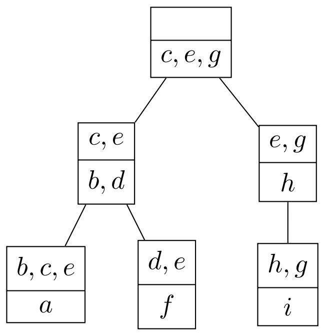


Figure 3.8: Clique tree of Figure 3.7 as a rooted clique tree with root $\{c, e, g\}$. The top row of each vertex is the intersection of the clique with its parent clique (the clique separator). These sets are the minimal vertex separators of the graph. The bottom row is the clique residual.

- The clique residuals partition V: every vertex $v \in V$ belongs to exactly one clique residual $\mathrm{res}(W)$.
- For each $v \in V$, the clique W for which $v \in \mathrm{res}(W)$ is the root of the induced subtree R_v of cliques that contain v. The other vertices of R_v are the cliques that contain v in $\mathrm{sep}(W)$.

Proof. If $v \in \mathrm{sep}(W)$ then $v \in p_{\mathrm{c}}(W)$, by definition of $\mathrm{sep}(W)$. This simple fact implies that if $v \in W$, then $v \in p_{\mathrm{c}}^j(W)$ for $j = 0, 1, \ldots, k$ where $k \geq 0$ satisfies $v \in \mathrm{res}(p_{\mathrm{c}}^k(W))$. Such an integer k exists because $\hat{W} = \mathrm{res}(\hat{W})$ for the root of the clique tree. This, combined with the fact that the cliques that contain v form a subtree (Theorem 3.5), implies that v is in exactly one set $\mathrm{res}(W)$ and that this clique W is the root of the induced subtree. □

The terminology for $\mathrm{sep}(W)$ is motivated by the following theorem, which states that the clique separators in the clique tree are the minimal vertex separators of the graph [38, theorem 4.3].

Theorem 3.7. Let $G = (V, E)$ be a chordal graph and let T be a rooted clique tree of G with the induced subtree property. The clique separators $\mathrm{sep}(W)$ where W ranges over all non-root cliques, are the minimal vertex separators of G.

Proof. Consider any non-root clique W. We show that the clique separator $\operatorname{sep}(W)$ is a minimal vertex separator. Removal of the edge $\{W, p_c(W)\}$ in the clique tree decomposes the clique tree in two connected subtrees. Let T_1 be the set of cliques in the subtree containing W, *i.e.*, W and all its descendants in the clique tree. Let T_2 the set of cliques in the subtree containing $p_c(W)$, *i.e.*, all cliques that are not descendants of W. The sets

$$\operatorname{sep}(W), \qquad V_1 = \bigcup_{\tilde{W} \in T_1} \tilde{W} \setminus \operatorname{sep}(W), \qquad V_2 = \bigcup_{\tilde{W} \in T_2} \tilde{W} \setminus \operatorname{sep}(W)$$

partition V. (It is obvious that the union of the three sets is V. Moreover V_1 and V_2 do not intersect because if $v \in V_1 \cap V_2$, then v is an element of a clique in T_1 and a clique in T_2, and, by the induced subtree property, an element of all the cliques on the path between these two cliques. Hence $v \in \operatorname{sep}(W)$, contradicting the definition of V_1 and V_2.) Moreover there exists no edge $\{v_1, v_2\}$ with $v_1 \in V_1$ and $v_2 \in V_2$, because this would mean there is a clique containing v_1 and v_2. This clique must be either in T_1 or in T_2, implying that $v_1, v_2 \in V_1$ or $v_1, v_2 \in V_2$. We conclude that $\operatorname{sep}(W)$ separates V_1 and V_2. To see that $\operatorname{sep}(W)$ is in fact a minimal vertex separator it is sufficient to note that $\operatorname{sep}(W)$ is a subset of two cliques W and $p_c(W)$, and therefore every vertex in $\operatorname{sep}(W)$ is adjacent to all vertices in $\operatorname{res}(W) = W \setminus \operatorname{sep}(W)$ and all vertices in $p_c(W) \setminus \operatorname{sep}(W)$. Therefore $\operatorname{sep}(W)$ is a minimal ab-separator for any $a \in \operatorname{res}(W)$ and $b \in p_c(W) \setminus \operatorname{sep}(W)$.

To show the converse, let S be a minimal uv-separator. Let R_u and R_v be the induced subtrees of the clique tree with the cliques that contain u and v, respectively. These subtrees are disjoint because u and v are not adjacent. Therefore there exists a (unique) path $(W_0, W_1, \ldots, W_r)$ in the clique tree with $u \in W_0$, $u \notin W_i$ for $1 \leq i \leq r$, $v \in W_r$, $v \notin W_i$ for $0 \leq i < r$. Define $S_i = W_i \cap W_{i+1}$ for $i = 0, \ldots, r-1$. From the first part of the proof, we know that each set S_i is a separator of two sets, one including u, the other including v, *i.e.*, every edge $\{W_i, W_{i+1}\}$ on the path defines a uv-separator S_i. We show that one of these uv-separators is equal to S. Since S is a minimal separator we cannot have $S_i \subset S$. Therefore, if $S_i \neq S$, we can choose a vertex $x_i \in S_i \setminus S$. If $S_i \neq S$ for $i = 0, \ldots, r-1$, this gives a path

$(u, x_0, \ldots, x_{r-1}, v)$ in G from u to v with interior vertices outside S, contradicting the assumption that S is a uv-separator. Hence we must have $S_i = S$ for at least one i. □

Note that a chordal graph can have several clique trees, and for the same clique tree the definition of the sets $\mathrm{sep}(W)$ and $\mathrm{res}(W)$ depends on the choice of root. However all clique trees have the same vertices (namely, the cliques of G) and the same clique separators (namely, the minimal vertex separators of G).

We can now also bound the number of minimal vertex separators. Since there are at most n cliques in a chordal graph with n vertices, there are at most $n-1$ edges in a clique tree and, hence, at most $n-1$ minimal vertex separators. The number of minimal vertex separators is not necessarily equal to the number of edges in the clique tree (the number of cliques minus one), because the same minimal vertex separator can appear more than once as a clique separator.

3.7 Tree intersection graph

In §3.5 (Algorithm 3.1) we showed how to construct a clique tree with the induced subtree property for a chordal graph G. Conversely, we can easily construct the graph $G = (V, E)$ from a clique tree T for it: the vertex set V is the union of all the vertices of T; two vertices u and v are adjacent if and only the induced subtrees R_u and R_v intersect, *i.e.*, there exists at least one clique that contains both u and v.

This construction of a graph G from the clique tree is an example of a more general class of graphs, known as *tree intersection graphs*. Let $\{R_v \mid v \in V\}$ be a family of subtrees of a tree T, indexed by a parameter v. The intersection graph associated with the family of subtrees is an undirected graph $G = (V, E)$ in which each vertex represents a subtree R_v and an edge $\{v, w\} \in E$ indicates that $R_v \cap R_w \neq \emptyset$ Here we do not distinguish between a tree and its vertex set: $R_v \cap R_w$ denotes the set of vertices common to R_v and R_w, as well as the subtree of T defined by those vertices; see Figure 3.9.

Gavril [92] and Buneman [46, theorem 2.1] have shown that all tree intersection graphs are chordal (see also [101, theorem 4.8]).

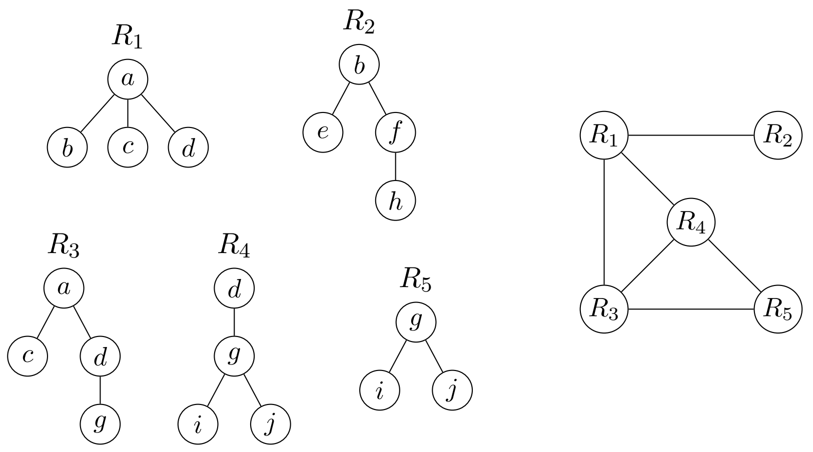


Figure 3.9: Five subtrees of the tree in Figure 2.4, and the associated tree intersection graph.

Theorem 3.8. Let $\{R_v \mid v \in V\}$ be a family of subtrees of a tree T and $G = (V, E)$ the associated tree intersection graph. Then G is chordal.

Proof. By contradiction. Suppose there exists a chordless cycle $(v_0, v_1, \ldots, v_{k-1}, v_k = v_0)$ of length $k \geq 4$ in G. We show that this implies there exists a cycle in T, contradicting the fact that T is a tree. To simplify the indexing we define $R_i = R_{v_i}$, $i = 0, \ldots, k$.

Since the cycle is chordless, the subtrees R_i and R_j intersect only if $|i - j| \bmod k \leq 1$. For $i = 0, \ldots, k-1$, define $S_i = R_i \cap R_{i+1}$. The sets S_i form nonempty subtrees of T, and are mutually disjoint because the intersection of any three or more subtrees R_j is empty. We can therefore define paths $P_1, \ldots, P_k, P'_0, \ldots, P'_{k-1}$ as follows. For $i = 1, \ldots, k-1$, the path P_i starts at a vertex in S_{i-1}, ends at a vertex in S_i, and has interior vertices that are in R_i but outside S_i and S_{i+1}. The path P_k starts at a vertex in S_{k-1}, ends at a vertex in S_0, and its interior vertices are in $R_0 = R_k$ but outside S_{k-1} and S_0. For $k = 1, \ldots, k-1$, the path P'_i starts at the end point of path P_i and ends at the starting point of P_{i+1}, and therefore has vertices in S_i. The path P'_0 starts at the end poin of P_k and ends at the starting point of P_1, and therefore has vertices in S_0. Concatenating the paths $P'_0, P_1, P'_1, \ldots, P_{k-1}, P'_{k-1}, P_k$ gives a cycle in T. □

3.8 Junction tree

Combining Theorems 3.5 and 3.8 we conclude that a graph is chordal if and only if it can be expressed as the tree intersection graph of a family of subtrees $\{R_v \mid v \in V\}$. In §3.5 we already described one method for representing a chordal graph as a tree intersection graph. In this representation the tree is a clique tree with the induced subtree property; the subtrees R_v are formed by the cliques that contain a specific vertex v. Later we will encounter several generalizations of clique trees, in which the condition that its vertices are *maximal* complete subgraphs is omitted. In the generalized definition a chordal graph $G = (V, E)$ is represented as the tree intersection graph of a tree T that satisfies the following properties.

- The vertices of T are subsets of V that induce complete subgraphs of G. The subgraphs induced by the vertices W of T cover G, *i.e.*, V is the union of the vertices W of T and E is the union of the sets $E(W)$.
- The vertices of T that contain a given vertex $v \in V$ form a subtree R_v of T.

In artificial intelligence and machine learning a tree with these properties is called a *junction tree* [58, section 4.3] or a *join tree* [22, 184, 64].

4

Perfect Elimination Ordering

In this chapter we develop a view of chordal graphs that is complementary to the clique tree representation of the previous chapter. We show that chordal graphs are exactly the undirected graphs for which a *perfect elimination ordering* exists.

4.1 Filled graph

An ordered graph $G_\sigma = (V, E, \sigma)$ is *filled* or *monotone transitive* if all higher neighborhoods $\mathrm{adj}^+(v)$ induce complete subgraphs: for all $v \in V$,

$$w, z \in \mathrm{adj}^+(v) \quad \Longrightarrow \quad \{w, z\} \in E. \tag{4.1}$$

Equivalently,

$$\begin{array}{c} i < j < k, \quad \{\sigma(i), \sigma(j)\} \in E, \quad \{\sigma(i), \sigma(k)\} \in E \\ \Downarrow \\ \{\sigma(j), \sigma(k)\} \in E. \end{array} \tag{4.2}$$

Figure 4.1 shows an example. Monotone transitivity is easily verified from the array representation in the figure: for each column i, if it contains entries in rows j and k, with $i < j < k$, then column j has an entry in row k.

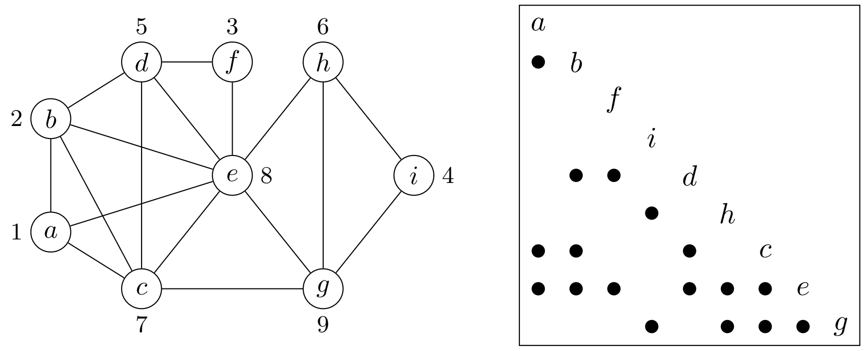


Figure 4.1: *Left.* Filled graph with 9 vertices. The number next to each vertex is the index $\sigma^{-1}(v)$. *Right.* Array representation of the same graph.

If the ordered graph is interpreted as a directed graph with edges oriented from lower to higher index (as described in §2.2), then monotone transitivity means that if there are edges from v to w and from v to z, then w and z are adjacent, *i.e.*, there is an edge from w to z or from z to w.

Another useful formulation of monotone transitivity is as follows. If two vertices w and z in a filled graph are connected by a path $(v_0 = w, v_1, \ldots, v_{k-1}, v_k = z)$ with interior vertices that precede w and z in the ordering (*i.e.*, $\sigma^{-1}(v_i) < \min\{\sigma^{-1}(w), \sigma^{-1}(z)\}$ for $i = 1, \ldots, k-1$), then w and z are adjacent. This follows by repeated application of monotone transitivity. Let v_i be the interior vertex of the path with the lowest index $\sigma^{-1}(v_i)$. By monotone transitivity its two neighbors in the path are adjacent and form a chord in the path. Therefore there exists a shorter path $(v_0 = w, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k = z)$ from w to z with interior vertices that precede w and z. Continuing this process one can remove all interior vertices and eventually arrives at a single-edge path (w, z).

The same property can be stated in another interesting form: if $(v_0 = w, v_1, \ldots, v_{k-1}, v_k = z)$ is a chordless path, then

$$\sigma^{-1}(v_i) \geq \min\{\sigma^{-1}(w), \sigma^{-1}(z)\}, \quad i = 1, \ldots, k-1.$$

The function $\sigma^{-1} : V \to \{1, 2 \ldots, n\}$ therefore satisfies a generalized quasiconcavity property. Recall that a function $f : \mathbf{R}^n \to \mathbf{R}$ is quasiconcave if $f(v) \geq \min\{f(w), f(z)\}$ for all v on the line segment $[w, z]$. In the generalized definition, $\mathbf{R}^n$ is replaced by the vertex set V of an undirected graph and line segments in $\mathbf{R}^n$ are replaced by chordless paths in the graph [204].

An immediate consequence of monotone transitivity is that $G = (V, E)$ is chordal. To see this, consider a cycle of length greater than three in G. Let v be the vertex in the cycle with least index $\sigma^{-1}(v)$ and let w, z be the two neighbors of v in the cycle. The vertices w and z are in $\mathrm{adj}^+(v)$ and are adjacent if the graph is monotone transitive. The edge $\{w, z\}$ is a chord in the cycle.

In the next section (Theorem 4.1) we will see that the converse holds: if G is chordal then there exists an ordering σ for which G_σ is a filled graph.

4.2 Perfect elimination ordering

An ordering σ of an undirected graph $G = (V, E)$ is called a *perfect elimination ordering* of G if the ordered graph $G_\sigma = (V, E, \sigma)$ is filled. (The term 'elimination' will be explained in Chapter 6.) Fulkerson and Gross have shown that the graphs for which a perfect elimination ordering exists are exactly the chordal graphs [88, page 851]. This explains why chordal graphs are also known as *perfect elimination graphs*.

Theorem 4.1. A graph is chordal if and only if it has a perfect elimination ordering.

Proof. The 'if'-part was shown in §4.1. The 'only if'-part can be proved by induction on the number of vertices n. The case $n = 1$ is trivial. Suppose G is a chordal graph with $n > 1$ vertices, and every chordal graph with fewer than n vertices has a perfect elimination ordering. By Dirac's simplicial vertex theorem (Theorem 3.3) the graph G has a simplicial vertex v. The subgraph $G' = G(V \setminus \{v\})$ is chordal with less than n vertices, so by the induction hypothesis it has a perfect elimination ordering σ'. Define an ordering σ for G by

$$\sigma(1) = v, \qquad \sigma(i) = \sigma'(i - 1), \quad i = 2, \ldots, n.$$

We verify that σ is a perfect elimination ordering for G. Since v is the first vertex in the ordering, its higher neighborhood is $\mathrm{adj}(v)$ and this set is complete because v is simplicial in G. The higher neighborhoods of the vertices $w \neq v$ in G_σ are the same as in $G'_{\sigma'}$ and are complete because σ' is a perfect elimination ordering of G'. $\square$

The inductive proof suggests a conceptual recursive algorithm for finding a perfect elimination ordering: for $i = 1, \ldots, n$, choose $\sigma(i) = v$ where v is a simplicial vertex of $G(V \setminus \{\sigma(1), \ldots, \sigma(i-1)\})$. The method succeeds in finding a perfect elimination ordering if and only if the graph is chordal. This is called *simplicial elimination.*

A perfect elimination ordering for a chordal graph is also easily obtained from a rooted clique tree with the induced subtree property. Recall from §3.6 that the clique residuals $\mathrm{res}(W)$ partition the vertex set V. Suppose there are l cliques, and let β be a topological ordering of the clique tree ($W \preceq_\beta p_{\mathrm{c}}(W)$ for every clique W). We order the vertices in V by visiting the cliques W in topological order and assigning consecutive numbers to the vertices in the clique residuals. More specifically, for $i = 1, \ldots, l$, let $W = \beta(i)$ and number the vertices in $\mathrm{res}(W)$ so that

$$\{\sigma^{-1}(v) \mid v \in \mathrm{res}(W)\} = \{n_i + 1, \ldots, n_i + |\mathrm{res}(W)|\}$$

where $n_i = \sum_{W' \prec_\beta W} |\mathrm{res}(W')|$. To see that this is a perfect elimination ordering, we have to verify that for every vertex v, the higher neighborhood $\mathrm{adj}^+(v)$ is complete. Consider a vertex $w \in \mathrm{adj}^+(v)$. Since w is adjacent to v, it belongs to at least one clique that also contains v, *i.e.*, a clique in the induced subtree of cliques that contain v. The root of the induced subtree is the clique W for which $v \in \mathrm{res}(W)$. Suppose $w \in W'$ and W' is a proper descendant of W in the clique tree. If $w \in \mathrm{res}(W')$, then by construction of the ordering, $w \prec_\sigma v$ so $w \notin \mathrm{adj}^+(v)$. Therefore if $w \in \mathrm{adj}^+(v) \cap W'$ and W' is a proper descendant of W, then $w \in \mathrm{sep}(W')$ and hence $w \in p_{\mathrm{c}}(W')$. By repeated application of this argument, we find that w is also an element of W, the root of the induced subtree of cliques that contain v. We conclude that $\mathrm{adj}^+(v) \subset W$ and therefore $\mathrm{adj}^+(v)$ is complete.

The perfect elimination ordering derived from a clique tree is generally not unique. We can use any topological ordering β of the clique tree to number the cliques, and for the same ordering β, use any consecutive ordering for the vertices in the clique residuals.

More practical methods for finding perfect elimination orderings and verifying chordality will be discussed in §4.7. Rose's simplicial vertex theorem (Theorem 3.4) is important in this context. It can be used to show, using an inductive argument similar to the proof of Theorem 4.1, that if W is any complete subgraph of a chordal graph, then there exists a perfect elimination ordering that orders the vertices in W last. In particular, for any given vertex w, there is a perfect elimination ordering that orders w last ($\sigma(n) = w$). Therefore, while the first vertex in a perfect elimination ordering must be simplicial, the last vertex can be chosen arbitrarily.

4.3 Elimination tree

In §3.6 and §3.7 we described how every chordal graph G can be represented as the tree intersection graph of an associated junction tree (namely, a clique tree with the induced subtree property). If a perfect elimination ordering for G is known, there is a more straightforward representation of G as the tree intersection graph of a junction tree, based on the *elimination tree* associated with the filled graph G_σ. For simplicity we will assume that G is connected (however, all results extend to unconnected graphs).

Let $G_\sigma = (V, E, \sigma)$ be a connected filled (monotone transitive) graph with n vertices. The elimination tree of G_σ is a rooted tree with vertices V and root $\sigma(n)$. The parent $p(u)$ of a non-root vertex u is defined as the first vertex in $\mathrm{adj}^+(u)$:

$$p(u) = \mathrm{argmin}\{\sigma^{-1}(v) \mid v \in \mathrm{adj}^+(u)\}.$$

The parent function is well defined because $\mathrm{adj}^+(u)$ is non-empty for $u \neq \sigma(n)$. Indeed, suppose $u \neq \sigma(n)$ and $\mathrm{adj}^+(u)$ is empty. Since G is connected, there exists a path $(u, v_1, \ldots, v_{k-1}, w)$ from u to some vertex $w \succ u$ with interior vertices v_i that precede u and w. By monotone transitivity, this implies u and w are adjacent, contradicting our

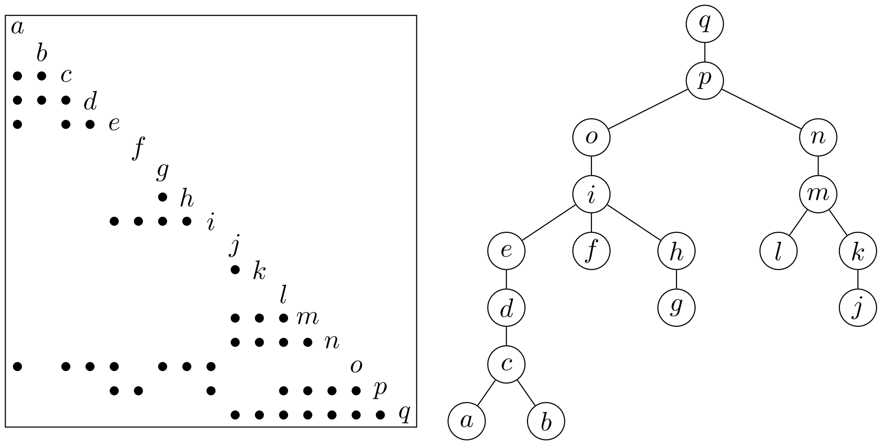


Figure 4.2: Filled graph and elimination tree.

assumption that $\mathrm{adj}^+(u)$ is empty. From the definition of the parent function, it is also clear that $u \prec_\sigma p(u)$ for all non-root vertices u, so the graph with vertices V and edges $\{u, p(u)\}$ is acyclic, *i.e.*, a tree, and the ordering σ is a topological ordering of the tree. An example is shown in Figure 4.2.

Monotone transitivity implies that all elements of $\mathrm{adj}^+(u)$ are mutually adjacent. In particular,

$$\mathrm{adj}^+(u) \subseteq \mathrm{col}(p(u)) = \{p(u)\} \cup \mathrm{adj}^+(p(u)) \tag{4.3}$$

for every non-root vertex u. Repeated application of this inclusion gives

$$\mathrm{adj}^+(u) \subseteq \{p(u), p^2(u), \ldots, p^k(u)\} \cup \mathrm{adj}^+(p^k(u)) \tag{4.4}$$

for $k = 1, \ldots, \mathrm{lev}(u)$, where $p^i(u)$ is the parent of u of degree i and $\mathrm{lev}(u)$ is the depth of u in the elimination tree. For $k = \mathrm{lev}(u)$, $p^k(u)$ is the root of the elimination tree and (4.4) reduces to

$$\mathrm{adj}^+(u) \subseteq \{p(u), p^2(u), \ldots, p^k(u)\}.$$

This shows that the elements of $\mathrm{adj}^+(u)$ are proper ancestors of u in the elimination tree. In other words, adjacent vertices in a filled graph are ancestor-descendant pairs in the elimination tree: if u and v are adjacent then u is a proper ancestor of v (if $u \succ v$) or a proper descendant (if $u \prec v$). In Figure 4.2, for example, we can conclude from the elimination tree that $\operatorname{adj}(h) \subseteq \{g, i, o, p, q\}$.

There exists a close similarity between elimination trees and clique trees. To develop this connection, it is useful to identify each vertex u in the elimination tree with its closed higher neighborhood $\operatorname{col}(u)$. Each of these sets $\operatorname{col}(u)$ is a complete subgraph of G and can be partitioned in two sets: $\operatorname{adj}^+(u)$ (which, by (4.3), is included in the parent set $\operatorname{col}(p(u))$) and the remainder, $\{u\}$. This corresponds to the partitioning of a clique W into a clique separator $\operatorname{sep}(W)$ (the intersection of W and the parent of W in the clique tree) and the clique residual $\operatorname{res}(W)$. When interpreted this way, the expanded elimination tree is a junction tree, as defined in §3.8 [153, section 4.2]. Its vertices $\operatorname{col}(v)$ are complete subgraphs of G, and for every given v, the set

$$\operatorname{row}(v) = \{u \mid v \in \operatorname{col}(u)\}$$

forms a subtree of the elimination tree, with root v. This follows from (4.4) which implies that if $v \in \operatorname{adj}^+(u)$, then $v \in \operatorname{col}(p^j(u))$ for $j = 1, \ldots, k$, where k satisfies $p^k(u) = v$. In other words, if $v \in \operatorname{adj}^+(u)$, then $v \in \operatorname{col}(w)$ for all vertices w on the path between u and v in the elimination tree. For this reason the closed lower neighborhoods $\operatorname{row}(v)$ are also called the *row subtrees*. This is illustrated in Figure 4.3 for the example of Figure 4.2. As an example,

$$\operatorname{row}(o) = \{o\} \cup \operatorname{adj}^-(o) = \{a, c, d, e, g, h, i, o\}$$

is a subtree of the elimination tree in (4.2), with root o and leaves a, g. In the expanded elimination tree of Figure 4.3 the vertices of this subtree are the sets $\operatorname{col}(v)$ that contain o.

The inclusions (4.4) imply some important inequalities involving the higher degrees $\deg^+(v) = |\operatorname{adj}^+(v)|$. Comparing the number of elements in the sets on the left- and right-hand sides gives

$$\deg^+(u) \leq k + \deg^+(p^k(u)), \quad k = 1, \ldots, \operatorname{lev}(u). \tag{4.5}$$

The inequality in (4.5) holds with equality if and only if

$$\operatorname{adj}^+(u) = \{p(u), p^2(u), \ldots, p^k(u)\} \cup \operatorname{adj}^+(p^k(u)).$$

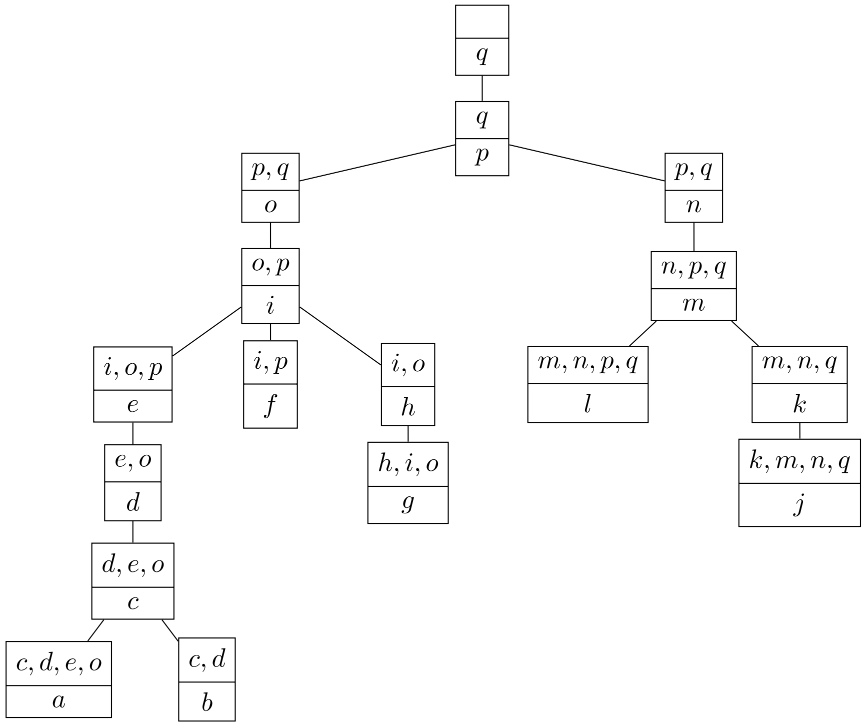


Figure 4.3: The junction tree obtained from the elimination tree of Figure 4.2. In the junction tree we replace each vertex v of the elimination tree with its closed neighborhood $\mathrm{col}(v)$. Each vertex $\mathrm{col}(v)$ is partitioned as $\mathrm{col}(v) = \{v\} \cup \mathrm{adj}^+(v)$, with v shown in the bottom row and $\mathrm{adj}^+(v)$ in the top row. Monotone transitivity implies that $\mathrm{adj}^+(v) \subset \mathrm{col}(p(v))$ for every non-root vertex v: the elements of the top row of a tree vertex are included in the parent vertex.

Moreover, if the inequality in (4.5) holds with equality for $k = j$, then it holds with equality for $1 \leq k \leq j$. If the inequality in (4.5) is a strict inequality for $k = j$, then it is a strict inequality for $j \leq k \leq \mathrm{lev}(u)$.

The higher degree inequalities are important in applications where filled graphs are constructed by graph elimination of non-chordal graphs (see Chapter 6), and the elimination tree and the higher degrees can be efficiently computed from the original, non-filled graph, without explicitly constructing the filled graph. For example, the relations between the higher degrees allow us to identify the leading elements of the higher neighborhoods by examining the elimination tree and the higher degrees. Suppose, for example, that we are given the elimination tree in Figure 4.2, plus the additional information that

$$\deg^+(a) = 4, \qquad \deg^+(c) = 3, \qquad \deg^+(d) = 2, \qquad \deg^+(e) = 3.$$

Then

$$\deg^+(a) = 1 + \deg^+(p(a)) = 2 + \deg^+(p^2(a)) < 3 + \deg^+(p^3(a)),$$

and without knowing more about the graph we can conclude that

$$\mathrm{adj}^+(a) = \{c, d\} \cup \mathrm{adj}^+(d) = \{c, d, e, \ldots\}.$$

4.4 Clique tree from elimination tree

In this section we describe how the cliques in a filled graph can be found from the elimination tree and the higher degrees of the vertices. We also show how to construct a clique tree with the induced subtree property based on the same information.

Consider a clique W in a filled graph $G_\sigma = (V, E, \sigma)$ and let $v = \mathrm{argmin}\{\sigma(w) \mid w \in W\}$ be the lowest vertex in W. Every vertex $w \neq v$ in the clique belongs to $\mathrm{adj}^+(v)$, since $w \succ v$ by choice of v, and w and v are adjacent because w and v belong to the same clique. Therefore $W \subseteq \mathrm{col}(v)$ and, since W is a maximal complete subgraph, $W = \mathrm{col}(v)$. We conclude that every clique can be expressed as $W = \mathrm{col}(v)$ where v is the lowest vertex of W [88, page 852] [91, page 183] [150, proposition 2]. This fact provides another easy proof that a connected chordal graph with n vertices has at most $n - 1$ cliques. The vertex v is called the *representative vertex* of the clique $W = \mathrm{col}(v)$.

Enumerating the cliques in a filled graph therefore amounts to finding the representative vertices. The following simple criterion only requires the elimination tree and the higher degrees [188, page 185].

Theorem 4.2. Let T be the elimination tree of a connected filled graph. A vertex v is a representative vertex if and only if

$$\deg^+(w) < \deg^+(v) + 1 \quad \forall w \in \mathrm{ch}(v), \tag{4.6}$$

where $\mathrm{ch}(v)$ denotes the set of children of v in T.

Proof. It follows from (4.5) that $\deg^+(w) \leq \deg^+(v) + 1$ for all $w \in \mathrm{ch}(v)$. We show that v is not a representative vertex if and only if $\deg^+(w) = \deg^+(v) + 1$ for at least one child w of v.

The set $\mathrm{col}(v)$ is not a clique if and only if $\mathrm{col}(v) \subset \mathrm{col}(u)$ for a representative vertex $u \neq v$. The vertex u must be a proper descendant of v, because otherwise v and u cannot be adjacent. From (4.4) this implies that $\mathrm{col}(v) \subset \mathrm{col}(w)$ for all w on the path from u to v in T and, in particular, for the child w of v on this path. In other words, $\mathrm{col}(v)$ is not a clique if and only if $\mathrm{col}(v) \subset \mathrm{col}(w)$ for some $w \in \mathrm{ch}(v)$. Since, $\mathrm{adj}^+(w) \subseteq \mathrm{col}(v)$ always holds for every child w of v (this is (4.4) for $k = 1$) the property $\mathrm{col}(v) \subset \mathrm{col}(w)$ is equivalent to the statements $\mathrm{col}(v) = \mathrm{adj}^+(w)$ and $\deg^+(v) + 1 = \deg^+(w)$. □

The representative vertex criterion in Theorem 4.2 can be further developed into an algorithm for constructing a clique tree with the induced subtree property [150, 188]. Suppose there are l cliques in G. Let V^{c} be the set of representative vertices. The vertex set V of G can be partitioned in l sets

$$\mathrm{snd}(v) = \{v, p(v), \ldots, p^{n_v}(v)\} \tag{4.7}$$

where $v \in V^{\mathrm{c}}$ and $n_v = |\mathrm{snd}(v)| - 1$ satisfies

$$\deg^+(v) = \deg^+(p^{n_v}(v)) + n_v. \tag{4.8}$$

This last condition is equivalent to $\deg^+(v) = \deg^+(p^k(v)) + k$ for $k = 1, \ldots, n_v$. The sets $\mathrm{snd}(v)$ are called *maximal supernodes* and the first vertex v is called the *representative vertex* of the supernode $\mathrm{snd}(v)$. (In [150, 188] the notation $\mathrm{new}(v)$ is used for $\mathrm{snd}(v)$.) A maximal supernode partition can be constructed recursively, by visiting the vertices in the elimination tree in topological order. We initialize the supernodes $\mathrm{snd}(u)$, $u \in V^{\mathrm{c}}$, as empty sets. We use a topological order (*i.e.*, every vertex is visited before its parent) to assign the vertices v of the elimination tree to supernodes. If v is a representative vertex, we add v to $\mathrm{snd}(v)$. If v is not a representative vertex, then, by Theorem 4.2, v has at least one child w with $\deg^+(w) = \deg^+(v) - 1$. We arbitrarily select a child w with this property and add v to the maximal supernode $\mathrm{snd}(u)$ that contains w. After processing all the vertices

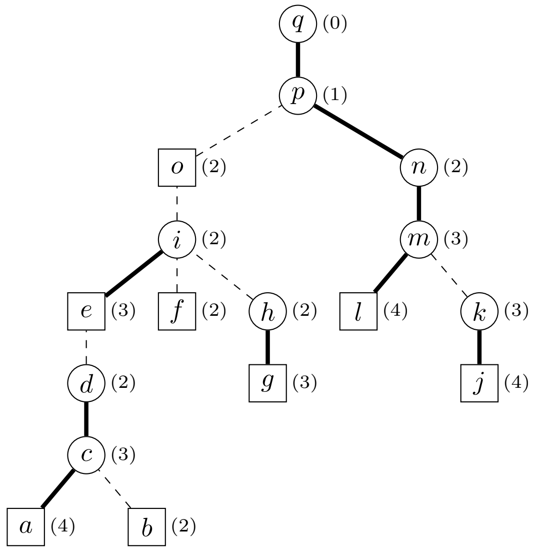


Figure 4.4: A maximal supernode partition of the elimination tree of Figure 4.2. The numbers next to the vertices are the higher degrees $\deg^+(v)$. The clique representative vertices are shown in rectangles. For each representative vertex v, the maximal supernode $\operatorname{snd}(v)$ contains the vertices on the solid path that starts at v.

in the elimination tree, the sets $\operatorname{snd}(v)$ partition V and satisfy (4.7) and (4.8). Note that the partition is not unique in general, because for every non-representative vertex v there may be more than one child w with $\deg^+(w) = \deg^+(v) - 1$. Figures 4.4 and 4.5 show two maximal supernode partitions for the elimination tree of Figure 4.2.

The maximal supernodes can be arranged in a *supernodal elimination tree* T^c. The vertex set of T^c is V^c (or, equivalently, the maximal supernodes $\{\operatorname{snd}(v) \mid v \in V^c\}$). The root of T^c is the representative of the supernode $\operatorname{snd}(v)$ that contains the root of the elimination tree $\sigma(n)$. For every other representative $v \in V^c$, we define the *first ancestor* $a(v)$ as the parent in the elimination tree of the highest element of $\operatorname{snd}(v)$: if $\operatorname{snd}(v) = \{v, p(v), \ldots, p^{n_v}(v)\}$ and $\operatorname{lev}(v) > n_v$, then $a(v) = p^{n_v+1}(v)$ (the name 'first ancestor' is from [150, 188]). The parent $q(v)$ of v in T^c is the representative vertex of the supernode that contains $a(v)$, *i.e.*, $q(v)$ is defined by the relation

$$a(v) \in \operatorname{snd}(q(v)). \tag{4.9}$$

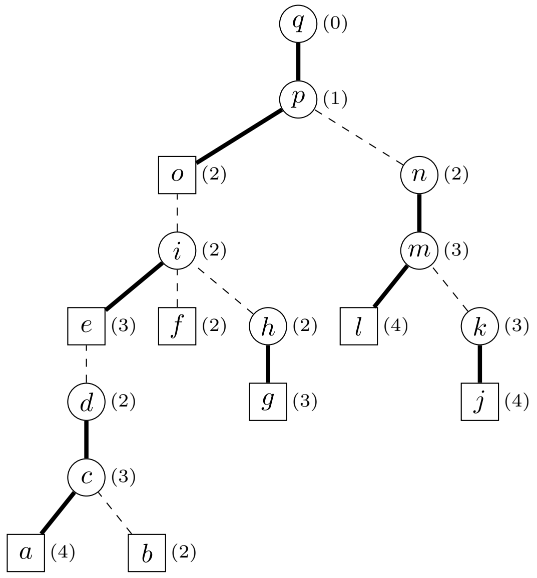


Figure 4.5: Another maximal supernode partition of the elimination tree of Figure 4.2.

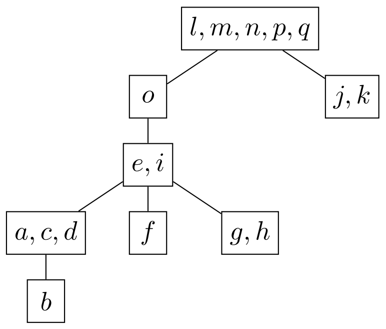
This defines $q(v)$ unambiguously because the sets $\operatorname{snd}(u)$ partition V so $a(v)$ belongs to exactly one supernode $\operatorname{snd}(u)$. The graph with vertex set V^{c} and edges $\{v, q(v)\}$ is acyclic (*i.e.*, a tree), because the definition (4.9) implies the ordering $a(v) \prec_\sigma a(q(v))$. The supernodal elimination trees for the maximal supernode partitions of Figures 4.4 and 4.5 are shown in Figure 4.6.

The next theorem states the most important properties of supernodal elimination trees.

Theorem 4.3. Let T^{c} be a supernodal elimination tree of a filled graph $G_\sigma = (V, E, \sigma)$, based on a partition of V in maximal supernodes $\operatorname{snd}(v)$. Then for all non-root vertices $v \in V^{\mathrm{c}}$,

$$\operatorname{col}(v) \setminus \operatorname{snd}(v) \subset \operatorname{col}(q(v)), \tag{4.10}$$

where $q(v)$ is the parent of v in T^{c}. For any given $u \in V$, the set $\{v \in V^{\mathrm{c}} \mid u \in \operatorname{col}(v)\}$ is a subtree of T^{c}, with as root the vertex v that contains u in $\operatorname{snd}(v)$.



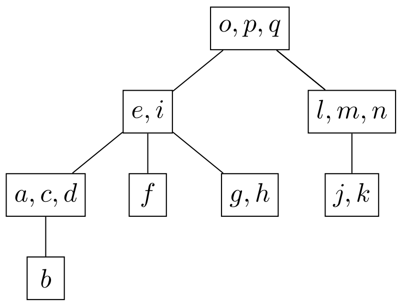


Figure 4.6: The supernodal elimination trees for the maximal supernode partition in Figure 4.4 (left) and Figure 4.5 (right). The first vertex in each rectangle is the representative vertex of the supernode.

Proof. The identity $\mathrm{col}(v) = \mathrm{snd}(v) \cup \mathrm{adj}^+(p^{n_v}(v))$ holds by definition of $\mathrm{snd}(v)$ (see equation (4.7)). Therefore, if v is not the root of T^c,

$$\begin{array}{rcl} \mathrm{col}(v) \setminus \mathrm{snd}(v) & = & \mathrm{adj}^+(p^{n_v}(v)) \\ & \subseteq & \{a(v)\} \cup \mathrm{adj}^+(a(v)) \\ & \subseteq & \mathrm{col}(q(v)). \end{array}$$

The second line follows from the fact that $a(v) = p^{n_v+1}(v)$. The last line follows from $a(v) \in \mathrm{col}(q(v))$. Moreover, (4.10) is a strict inclusion because v and $q(v)$ are different representative vertices, and the clique $\mathrm{col}(q(v))$ cannot be subset of another clique $\mathrm{col}(v)$.

The induced subtree property follows from (4.10). For any given $u \in V$, there is a unique representative vertex v for which $u \in \mathrm{snd}(v)$. Suppose $u \in \mathrm{col}(w)$ where w is a representative vertex different from v. Then $u \not\in \mathrm{snd}(w)$, and therefore $u \in \mathrm{col}(w) \setminus \mathrm{snd}(w)$. By (4.10), we have $w \in \mathrm{col}(q(w))$. Repeating this argument, we reach the conclusion that $u \in \mathrm{col}(q^i(w))$, for $i = 0, \ldots, k$, where $q^k(w) = v$. □

From the maximal supernodes and a supernodal elimination tree one immediately obtains a clique tree with the induced subtree property. The parent function p_c of the clique tree is defined as $p_c(\mathrm{col}(v)) = \mathrm{col}(q(v))$. The clique residuals and separators are

$$\mathrm{res}(\mathrm{col}(v)) = \mathrm{snd}(v), \qquad \mathrm{sep}(\mathrm{col}(v)) = \mathrm{col}(v) \setminus \mathrm{snd}(v).$$

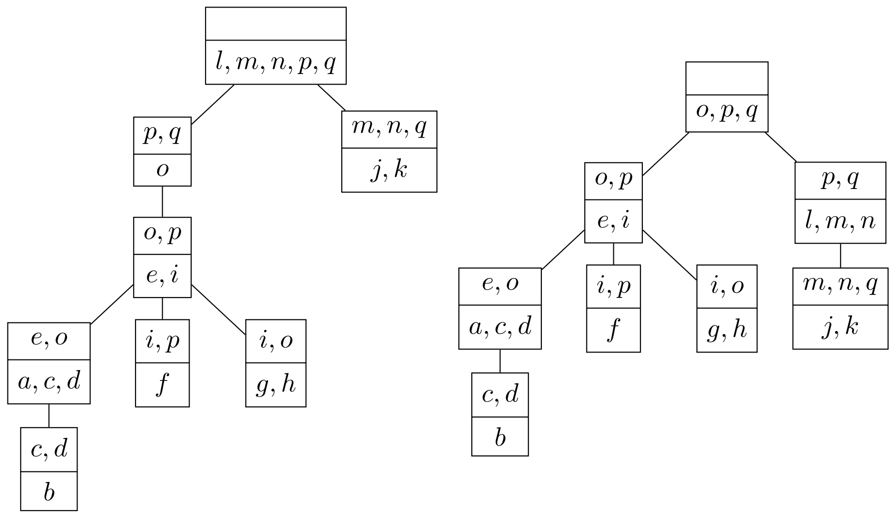


Figure 4.7: The two clique trees obtained from the maximal supernode partitions of Figures 4.4 and 4.5. The bottom rows are the maximal supernodes $\mathrm{snd}(v)$. The top rows are the sets $\mathrm{col}(v) \setminus \mathrm{snd}(v)$. The first element of each top row is the first ancestor $a(v)$.

This follows from (4.10) and the fact that $\mathrm{snd}(v) \cap \mathrm{col}(q(v)) = \emptyset$. (Since the vertices in $\mathrm{snd}(v)$ precede $a(v)$ in the ordering σ, they cannot be in $\mathrm{col}(q(v))$, because this would mean they are in $\mathrm{snd}(q(v))$, contradicting the fact the sets $\mathrm{snd}(u)$ partition V.) The clique trees for the two supernodal elimination trees of Figure 4.6 are shown in Figure 4.7.

We will sometimes use the term *nodal* elimination tree for the elimination tree (as defined in §4.3) to avoid confusion with the supernodal elimination tree. Note that knowledge of the nodal elimination tree and the higher degrees is sufficient to identify the representative vertices, the maximal supernodes $\mathrm{snd}(v)$, the first ancestors $a(v)$, and the supernodal elimination tree. However, in general the elimination tree and the higher degrees do not give enough information to find all the elements of the cliques (*i.e.*, the top rows in the vertices of Figure 4.7).

We summarize this section with an algorithm that computes the representative vertices, maximal supernodes $\mathrm{snd}(v)$, first ancestors $a(v)$, and the parent function $q(v)$ of the supernodal elimination tree.

The algorithm was proposed by Pothen and Sun [188, page 185] as a simplification of an earlier algorithm by Lewis, Peyton, and Pothen [150, page 1154]. It only requires knowledge of the elimination tree and the higher degrees of all vertices.

Algorithm 4.1 (Maximal supernodes and supernodal elimination tree)**.**

Input. An elimination tree for a connected filled graph $G_\sigma = (V, E, \sigma)$ and the higher degrees $\deg^+(v)$ for all $v \in V$.

Output. The set V^{c} of clique representatives, the maximal supernodes $\mathrm{snd}(v)$ for $v \in V^{\mathrm{c}}$, the parent function $q(v)$ of the supernodal elimination tree, and the first ancestor $a(v)$ of each non-root representative vertex.

Algorithm. Initialize V^{c} as $V^{\mathrm{c}} = \emptyset$. Enumerate the vertices v of T in topological order. For each v, execute the following steps.

- If $\deg^+(v) > \deg^+(w) - 1$ for all $w \in \mathrm{ch}(v)$, then v is a representative vertex. Add v to V^{c}, set $\mathrm{snd}(v) := \{v\}$, and define $u := v$, $W := \mathrm{ch}(v)$.

  Otherwise, choose a vertex $\hat{w} \in \mathrm{ch}(v)$ with $\deg^+(\hat{w}) = \deg^+(v) + 1$. Let u be the vertex in V^{c} that satisfies $\hat{w} \in \mathrm{snd}(u)$. Add v to $\mathrm{snd}(u)$ and set $W := \mathrm{ch}(v) \setminus \{\hat{w}\}$.

- For all $w \in W$, set $q(z) := u$ and $a(z) := v$ where z is the vertex in V^{c} that satisfies $w \in \mathrm{snd}(z)$.

4.5 Supernodal elimination tree

The maximal supernode partition used in the previous section can be generalized as follows. A general supernode partition of a filled graph (V, E, σ) is a partition of V in sets $\mathrm{snd}(v)$ for $v \in V^{\mathrm{s}}$, where each set $\mathrm{snd}(v)$ is of the form (4.7) with n_v satisfying (4.8). The sets $\mathrm{snd}(v)$ are called *supernodes* and the elements of V^{s} are the representative vertices of the supernodes. The set of supernode representatives V^{s} contains the clique representatives V^{c}, but may also contain other vertices. The most important choices of V^{s} are the following.

- *Nodes.* If we take $V^{\mathrm{s}} = V$ and $\mathrm{snd}(v) = \{v\}$ the supernodes reduce to single-vertex sets (or 'nodes').

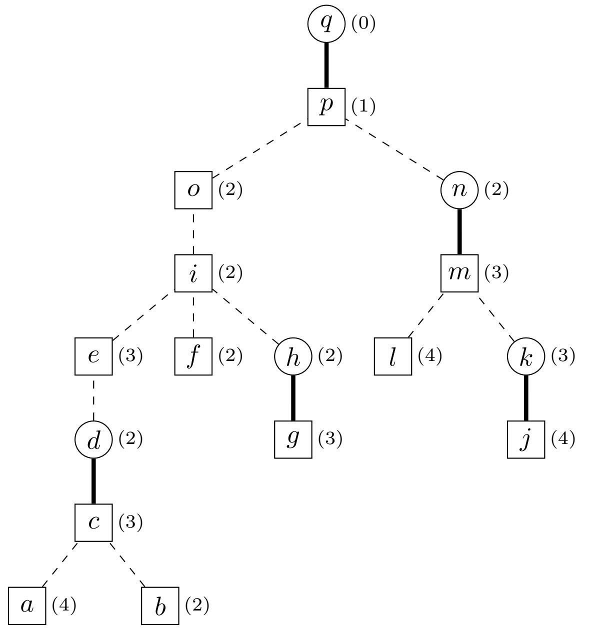


Figure 4.8: Fundamental supernode partition for the elimination tree of Figure 4.2. The numbers next to the vertices are the higher degrees $\deg^+(v)$. The representative vertices of the fundamental supernodes are shown in rectangles. For each representative vertex v, the set $\operatorname{snd}(v)$ contains the vertices on the solid path that starts at vertex v.

- *Fundamental supernodes.* The representative vertices of the *fundamental* supernodes include the representative clique vertices, plus the vertices with more than one child in the elimination tree. An example is shown in Figure 4.8. Liu, Ng, and Peyton [155, page 245-246] give reasons why fundamental supernodes are easier to work with than the maximal supernodes described in §4.4. One can note, for example, that fundamental supernodes are uniquely defined for a given filled graph, unlike maximal supernodes.

- *A variation on fundamental supernodes.* In the definition of Peyton [186] the supernode representatives V^{s} contain all vertices v that satisfy $\deg^+(v) = \deg^+(w) - 1$ for at most one child $w \in \operatorname{ch}(v)$. In the elimination tree of Figure 4.4, for example, we would take $V^{\mathrm{s}} = V^{\mathrm{c}} \cup \{p\}$.

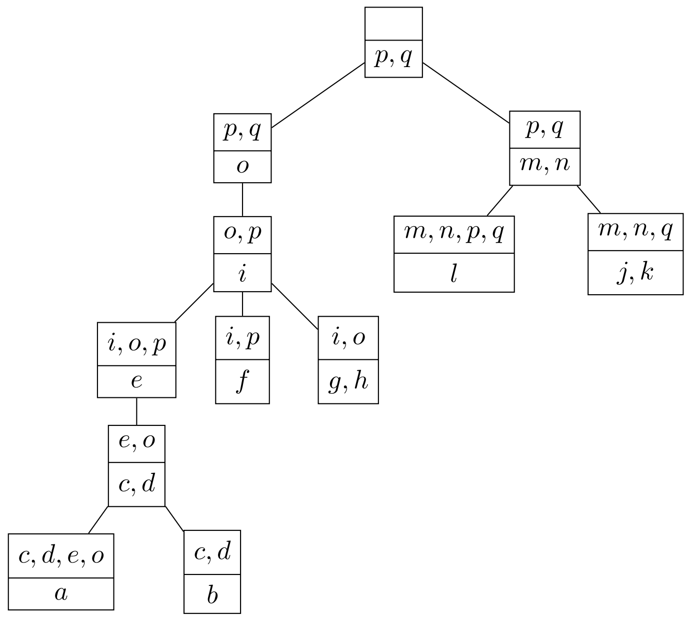


Figure 4.9: The junction tree corresponding to the fundamental supernode partition in Figure 4.8. The bottom row at each vertex is the fundamental supernode $\mathrm{snd}(v)$, the top row is the set $\mathrm{col}(v) \setminus \mathrm{snd}(v)$. The first element in the top row is the first ancestor $a(v)$.

- *Maximal supernodes (clique supernodes).* These are the supernodes discussed in §4.4. Here $V^{\mathrm{s}} = V^{\mathrm{c}}$, the set of clique representatives.

The supernodes (or, equivalently, their representative vertices) can be arranged in a supernodal elimination tree in exactly the same way as described in the previous section. The vertex set of the supernodal elimination tree is V^{s}. The root of the tree is the representative vertex v of the supernode that contains the root $\sigma(n)$ of the elimination tree. For every other representative vertex $v \in V^{\mathrm{s}}$, we define the first ancestor as $a(v) = p^{n_v+1}(v)$ and the parent $q(v)$ as the representative of the supernode $\mathrm{snd}(q(v))$ that contains $a(v)$. When $V^{\mathrm{s}} = V$ the supernodal elimination tree reduces to the nodal elimination tree. When $V^{\mathrm{s}} = V^{\mathrm{c}}$ it is the supernodal elimination tree for the maximal supernodes. An example for the fundamental supernodes is shown in Figure 4.9. It can be verified that Theorem 4.3 holds for supernodal elimination trees defined using any supernode partition, not only maximal supernodes.

Supernodal elimination trees generalize and unify several different representations of a chordal graph as the intersection graph of a junction tree. Every supernodal elimination tree defines a junction tree, with vertices $\operatorname{col}(v)$ for $v \in V^{\mathrm{s}}$ and edges $\{\operatorname{col}(v), \operatorname{col}(q(v))\}$. The subtrees that define the intersection graph are the subgraphs $R_v = \{u \in V^{\mathrm{s}} \mid v \in \operatorname{col}(u)\}$.

4.6 Topological reordering

If $G_\sigma = (V, E, \sigma)$ is a filled graph with elimination tree T, then σ is a topological ordering of T, because, by definition, a vertex precedes its parent in the ordering. If ρ is any other topological ordering of T, then it is also a perfect elimination ordering for G and the elimination tree for the filled graph $G_\rho = (V, E, \rho)$ is equal to T, the elimination tree for G_σ [153, section 6.1]. This can be seen as follows. We noted in §4.2 that adjacent vertices in G are ancestor-descendant pairs in an elimination tree: if $\{v, w\} \in E$, then w is a proper descendant or a proper ancestor of v. Now a topological reordering of the tree preserves the ordering of the vertices on every path from a leaf vertex to the root. Hence if $\{w \mid \{v, w\} \in E,\ w \succ_\sigma v\} = \{w_1, w_2, \ldots, w_k\}$ is the higher neighborhood of v for the ordering σ, with $w_1 \prec_\sigma w_2 \prec_\sigma \cdots \prec_\sigma w_k$, then it is also the higher neighborhood $\{w \mid \{v, w\} \in E,\ w \succ_\rho v\}$ of v in the ordering ρ and $w_1 \prec_\rho w_2 \prec_\rho \cdots \prec_\rho w_k$. It follows that the higher neighborhood in the ordering ρ is complete if the higher neighborhood in the ordering σ is complete, and that the parent vertex w_1 of v is the same for the two orderings.

In particular, for a given supernode partition $\{\operatorname{snd}(v) \mid v \in V^{\mathrm{s}}\}$, we can always find an ordering ρ (for example, by a suitable postordering of the elimination tree) with the following two properties.

- The elements of each supernode $\operatorname{snd}(v)$ are numbered consecutively: if $\rho^{-1}(v) = i$ and $n_v = |\operatorname{snd}(v)| - 1$, then

$$\operatorname{snd}(v) = \{\rho(i), \rho(i+1), \ldots, \rho(i+n_v)\}.$$

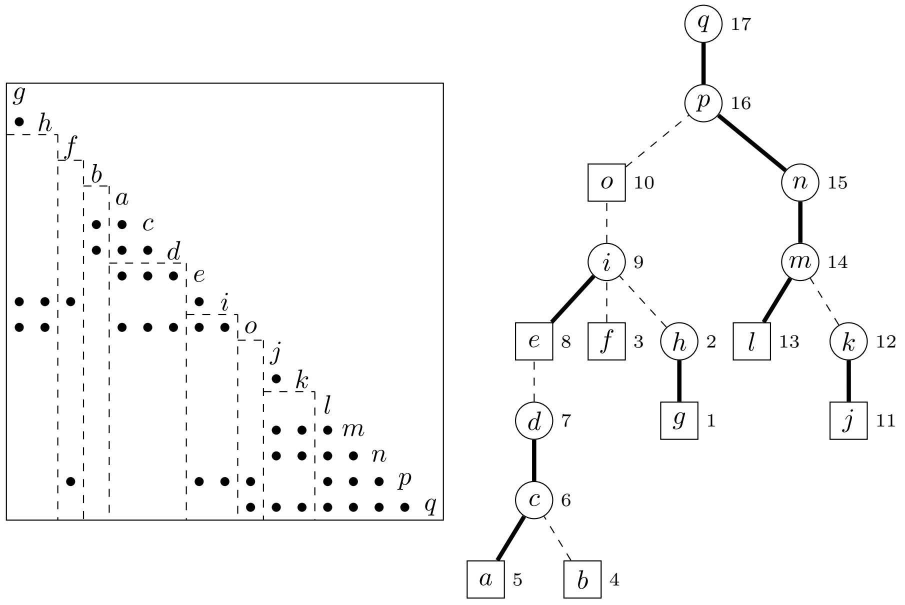


Figure 4.10: *Right.* A postordering ρ of the elimination tree in Figure 4.2. The number next to vertex v is the position $\rho^{-1}(v)$ in the ordering. This ordering gives the elements of the maximal supernodes in the left-hand partition of Figure 4.6 consecutive indices and defines a topological ordering of the supernodal elimination tree on the left-hand side of Figure 4.6. *Left.* The reordered filled graph of Figure 4.2 using the ordering ρ. The supernodes $\{g,h\}$, $\{f\}$, $\{b\}$, $\{a,c,d\}$, $\{e,i\}$, $\{o\}$, $\{j,k\}$, $\{l,m,n,p,q\}$ contain consecutive vertices with identical column structure.

- The ordering ρ defines a topological ordering of the representative vertices in the supernodal elimination tree: $v \prec_\rho q(v)$ if $v \in V^{\mathrm{s}}$ and $q(v)$ is the parent of v in the supernodal elimination tree.

This is illustrated in Figure 4.10.

4.7 Testing chordality

Perfect elimination orderings for chordal graphs can be found by algorithms that have an $O(n+m)$ complexity, where $n = |V|$ and $m = |E|$. (Note that $O(m+n) = O(m)$ for a connected graph since $m \geq n-1$.) This provides the easiest method for testing chordality: apply an algorithm that is guaranteed to find a perfect elimination ordering for chordal graphs, then check that the ordering is a perfect elimination ordering.

The oldest linear-time algorithm for constructing perfect elimination orderings is the *lexicographic breadth-first-search (Lex-BFS)* algorithm invented by Rose, Tarjan, and Lueker [195]. Later, Tarjan and Yannakakis developed a simpler algorithm, known as *maximum cardinality search (MCS)* [215]. MCS assigns numbers in decreasing order $\sigma^{-1}(v) = n, n-1, \ldots, 1$. At each step it selects the vertex that is adjacent to the largest number of already numbered vertices.

Algorithm 4.2 (Maximum cardinality search).

Input. A chordal graph $G = (V, E)$.

Output. A perfect elimination ordering σ of G.

Algorithm. For $i = n, n-1, \ldots, 1$, let $\sigma(i)$ be a vertex v that maximizes

$$|\mathrm{adj}(v) \cap \{\sigma(i+1), \ldots, \sigma(n)\}|.$$

In each step ties are broken arbitrarily. Implementation details and a derivation of the $O(m+n)$ complexity can be found in [215, page 569]. When applied to a chordal graph, MCS produces a perfect elimination ordering [215, theorem 2].

Theorem 4.4. MCS generates a perfect elimination ordering for chordal graphs.

Proof. The proof is by contradiction. Let σ be an ordering generated by MCS applied to the chordal graph G. Suppose σ is not a perfect elimination order. Then there exists a vertex u for which $\mathrm{adj}^+(u)$ is not complete, so it contains at least two vertices v, w that are not mutually adjacent. Define $v_0 = v$, $v_1 = u$, $v_2 = w$. The path (v_0, v_1, v_2) is a chordless path that satisfies $v_0 \succ v_1$, $v_2 \succ v_1$. We show that this is impossible. More generally, it is impossible that there exists a chordless path $P = (v_0, v_1, \ldots, v_{k-1}, v_k)$ with $k \geq 2$ and interior vertices $v_1, \ldots, v_{k-1}$ that precede v_0 and v_k in the ordering. Suppose such a path exists. Among all paths with this property choose P for which $\min\{\sigma^{-1}(v_0), \sigma^{-1}(v_k)\}$ is maximum. We assume without loss of generality that $v_0 \succ v_k$. We then have $v_0 \succ v_k \succ v_1$, with v_0 adjacent to v_1 but not to v_k. Since MCS selected v_k before v_1, there must be a vertex $z \succ v_k$ that is adjacent to v_k and not to v_1. Let $i = \min\{i = 2, \ldots, k \mid \{v_i, z\} \in E\}$. Since $z \succ v_k$ and the vertices $v_1, \ldots, v_{k-1}$ precede v_k and v_0 in the ordering, the path $(v_0, v_1, \ldots, v_i, z)$ has interior vertices that precede v_0 and z in the ordering. Moreover, the path is chordless because z is not adjacent to $v_1, \ldots, v_{i-1}$ (by choice of i) and has length at least three since $i \neq 1$. Therefore v_0 is not adjacent to z, since an edge $\{v_0, z\}$ would create a chordless cycle of length at least four. Hence $(v_0, v_1, \ldots, v_i, z)$ is a chordless path with interior vertices that precede v_0 and z in the ordering. However, $\min\{\sigma^{-1}(v_0), \sigma^{-1}(z)\} > \sigma^{-1}(v_k) = \min\{\sigma^{-1}(v_0), \sigma^{-1}(v_k)\}$ which contradicts the maximality property used to select P. $\square$

5

Combinatorial Optimization

A number of combinatorial problems that are NP-complete for general graphs can be solved in polynomial time for chordal graphs. We have already encountered an example in §4.4, where we saw that a chordal graph with n vertices has at most n cliques, and that the representative vertices and their monotone degrees (which are equal to the clique sizes minus one) can be found in linear time. The size of the largest clique (the *clique number* $\omega(G)$) of a chordal graph can therefore be computed in linear time. However determining the clique number of a general graph is NP-complete [199, page 1084].

5.1 Minimum clique cover

A set $S \subseteq V$ is a *stable* or *independent set* of an undirected graph $G = (V, E)$ if no two vertices in S are adjacent, *i.e.*, the induced edge set $E(S) = \{\{v, w\} \in E \mid v, w \in S\}$ is empty. The size of the largest stable set is called the *stable set number* of the graph, denoted $\alpha(G)$. A *clique cover* of G is a set of cliques that cover the vertex set V. The *clique cover number* $\bar{\chi}(G)$ is the minimum number of cliques in a clique cover.

If $\{C_1, \ldots, C_k\}$ is a clique cover with k cliques and S is a stable set, then no two elements of S can be in the same clique C_i. Therefore $|S| \leq k$. It follows that the stability number is a lower bound on the clique cover number, *i.e.*, the inequality

$$\alpha(G) \leq \bar{\chi}(G) \tag{5.1}$$

holds for any graph G. For chordal graphs this inequality holds with equality [112] and both numbers can be computed in polynomial time [91].

The following algorithm is Gavril's algorithm [91] formulated in terms of an elimination tree. It returns a stable set S with the property that V is the union of the sets $\mathrm{col}(v)$ for $v \in S$. The complete subgraphs induced by the sets $\mathrm{col}(v)$ are not necessarily maximal, but each is a subset of at least one clique and since the union of these cliques covers V, one obtains from S at least one clique cover with $k = |S|$ cliques. As $k \geq |S|$ in general, we have $|S| = \alpha(G) = \bar{\chi}(G)$.

Algorithm 5.1 (Minimum clique cover).

Input. A connected chordal graph $G = (V, E)$ and an elimination tree for G.

Output. A stable set S with $\bigcup_{v\in S} \mathrm{col}(v) = V$.

Algorithm. Initialize S as the empty set. Enumerate the vertices v of the elimination tree in topological order. For each v, execute the following steps.

- Form the set
$$W = \bigcup_{w\in \mathrm{ch}(v)} U_w$$
(with the convention that $W = \emptyset$ if v is a leaf of the elimination tree).
- If $v \notin W$, add v to S and define $U_v = W \cup \mathrm{col}(v)$. Otherwise, define $U_v = W$.

The set U_v is the union of the sets $\mathrm{col}(w)$ for the vertices $w \in S$ that are descendants of v, and is constructed recursively, from the sets U_w of the children w of v. By construction $w \in U_v$ for all descendants w of v. Therefore, at the end of the algorithm, when v is the root of the elimination tree, we have $U_v = \cup\{\mathrm{col}(w) \mid w \in S\} = V$.

Moreover, throughout the algorithm, the set S is a stable set, *i.e.*, no vertices $v, w \in S$ are adjacent. For vertices $v, w \in S$ that are not ancestor-descendant pairs this is a general property of elimination trees. For $v, w \in S$ with w a descendant of v, it follows from the condition used to determine whether v is added to S, which implies that $v \notin \mathrm{col}(w)$ for all $w \in S$ that are descendants of v.

5.2 Minimum vertex coloring

A partitioning of the vertices V into stable sets C_i, $i = 1, \ldots, k$, is called a *(vertex) coloring*. The partitioning can be interpreted as an assignment of colors to the vertices in such a way that no adjacent vertices use the same color. Following this interpretation the sets C_i in the partitioning are also called the *colors*. The *coloring number* or *chromatic number* $\chi(G)$ is the minimum number of colors needed for a vertex coloring.

If $C_1, \ldots, C_k$ is a vertex coloring and W is a clique, then all elements of W must receive different colors. Hence $|W| \leq k$. The clique number is therefore always a lower bound on the chromatic number:

$$\omega(G) \leq \chi(G). \tag{5.2}$$

For chordal graphs this holds with equality [27]. A simple polynomial algorithm for coloring chordal graphs was proposed by Gavril [91].

The following greedy algorithm colors a chordal graph with $\omega(G)$ colors.

Algorithm 5.2 (Minimum vertex coloring).

Input. A connected chordal graph $G = (V, E)$ and an elimination tree for G.

Output. A minimum vertex coloring.

Algorithm. Initialize $k = 0$. Enumerate the vertices v of the elimination tree in inverse topological order. For each vertex v, execute the following steps.

- If $\deg^+(v) \geq k$, increment k by one and define $C_k = \{v\}$.
- Otherwise, select a set C_i, $i = 1, \ldots, k$, with $C_i \cap \mathrm{adj}^+(v) = \emptyset$ and add v to C_i.

The algorithm guarantees that after each step the vertices in $\operatorname{col}(v)$ are assigned different colors. This can be shown by induction and the fact that $\operatorname{adj}^+(v) \subset \operatorname{col}(p(v))$ in a chordal graph. On exit the number of colors used is

$$k = 1 + \max_v \deg^+(v) = \max_v |\operatorname{col}(v)|.$$

This is equal to $\omega(G)$ because every clique is of the form $\operatorname{col}(v)$. The lower bound (5.2) then implies that the coloring is optimal.

Applications of minimum vertex coloring with chordal graphs are described in [67, 185].

5.3 Perfect graphs

The *complement* of a graph $G = (V, E)$ is the graph $G^{\mathrm{c}} = (V, E^{\mathrm{c}})$ with edge set

$$E^{\mathrm{c}} = \{\{v, w\} \mid v, w \in V, \{v, w\} \notin E\}.$$

Two vertices are adjacent in G^{c} if and only if they are not adjacent in G. Therefore stable sets of G induce complete subgraphs in G^{c} and vice-versa, and we have

$$\alpha(G) = \omega(G^{\mathrm{c}}), \quad \bar{\chi}(G) = \chi(G^{\mathrm{c}}), \quad \omega(G) = \alpha(G^{\mathrm{c}}), \quad \chi(G) = \bar{\chi}(G^{\mathrm{c}}).$$

For a general graph, determining any of these four numbers is an NP-complete problem [199, theorems 64.1, 64.2].

A graph is *perfect* if the inequality (5.2) holds with equality for the graph and all its subgraphs, *i.e.*,

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

for all $W \subseteq V$. Lovász [157] has shown that this is equivalent to the condition that (5.1) is an equality for G and all its induced subgraphs, *i.e.*,

$$\alpha(G(W)) = \bar{\chi}(G(W))$$

for all $W \subseteq V$. Equivalently, a graph is perfect if and only if its complement is perfect. The results of the previous sections, combined with the fact that subgraphs of chordal graphs are chordal, show that chordal graphs are perfect.

Perfect graphs were introduced by Berge in the early 1960s. As recounted in [28] the definition was motivated by Shannon's work on the zero-error capacity of a channel [203]. Suppose the vertices of the graph $G = (V, E)$ represent symbols and the edges $\{v, w\} \in E$ are pairs of symbols that can be confused during transmission over a communication channel. The kth power of G is defined as the graph $G^k = (V^k, E^k)$ with vertex set $V^k = V \times V \times \cdots \times V$ (all k-tuples of vertices of G) and an edge

$$\{(v_1, v_2, \ldots, v_k), (w_1, w_2, \ldots, w_k)\} \in E^k$$

if and only if $\{v_i, w_i\} \in E$ for $i = 1, \ldots, k$. The vertices V^k represent words of length k; the edges indicate pairs of words that can be confused during transmission. Stable set of G^k are therefore of interest as 'dictionaries' of words that cannot be confused. The *zero-error capacity* or *Shannon capacity* of a graph G is defined as

$$\Theta(G) = \sup_k \alpha(G^k)^{1/k}.$$

Now it is easy to see that $\alpha(G^k) \geq \alpha(G)^k$ (if S is a stable set of G, then $S \times \cdots \times S$ is a stable set of G^k). Also, cliques of G^k are of the form $C_1 \times \cdots \times C_k$ where $C_1, \ldots, C_k$ are cliques of G. Hence $\bar{\chi}(G^k) = \bar{\chi}(G)^k$. Applying (5.1) to G^k we obtain

$$\alpha(G)^k \leq \alpha(G^k) \leq \bar{\chi}(G^k) = \bar{\chi}(G)^k.$$

Therefore

$$\alpha(G) \leq \Theta(G) \leq \bar{\chi}(G)$$

and for perfect graphs, $\alpha(G) = \Theta(G) = \bar{\chi}(G)$.

Given the topic of this survey, it is interesting to note that the Shannon capacity of a graph later played a key role in the development of semidefinite optimization, when the calculation of Lovász's upper bound became one of the first applications of semidefinite optimization to combinatorial optimization [158, 110].

We refer the interested reader to the books [101], [111, section 9.2] [45] [189] [199, part VI] for more background on perfect graphs.

6

Graph Elimination

Filled graphs are often created from non-filled graphs by *graph elimination*, a process that models the progressive fill-in during Gauss elimination.

6.1 Elimination graph

Let $G_\sigma = (V, E, \sigma)$ be an ordered undirected graph. The *elimination graph* of G_σ is defined as follows [195, page 267]. We take $E_0 = E$ and, for $i = 1, \ldots, n-1$,

$$\begin{aligned} E_i = E_{i-1} \cup \\ \{\{v, w\} \mid v \succ \sigma(i),\ w \succ \sigma(i),\ \text{and } \{\sigma(i), v\}, \{\sigma(i), w\} \in E_{i-1}\}. \end{aligned} \tag{6.1}$$

The graph (V, E_{n-1}, σ) is the elimination graph of G_σ. In step i of the process described by (6.1), the higher neighborhood of vertex $\sigma(i)$ in the intermediate graph (V, E_{i-1}, σ) is made complete by the addition of edges between all non-adjacent vertices. An example is shown in Figure 6.1.

We use the notation $G^*_\sigma = (V, E_{n-1}, \sigma)$ for the elimination graph of G_σ, and $E^*_\sigma = E_{n-1}$ for its edge set. The set of added edges $E^*_\sigma \setminus E$ is

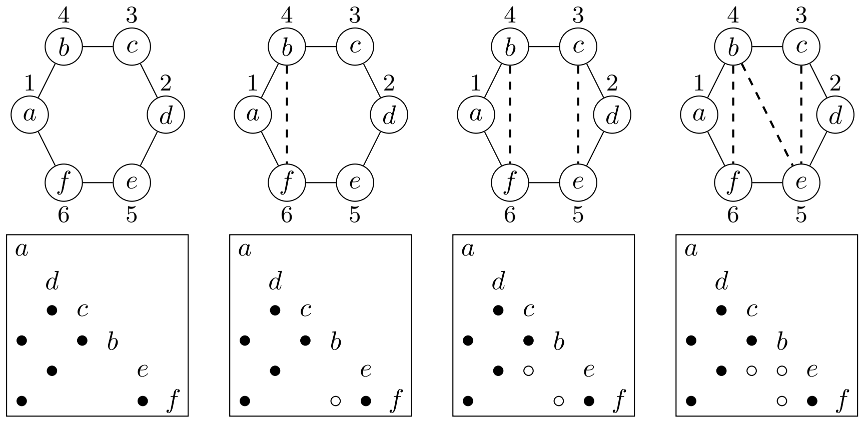


Figure 6.1: The first three steps in the graph elimination of the ordered graph on the left. Filled edges are shown as dashed lines and open circles.

called the *fill-in* or *fill*, and its elements are the *filled edges*. An elimination graph is filled (monotone transitive) because, by construction, the addition of the edges in (6.1) makes the higher neighborhood of vertex $\sigma(i)$ in (V, E_i, σ) complete. After completion of the process the higher neighborhoods of all vertices in (V, E_σ^*, σ) are complete. Hence (V, E_σ^*) is chordal and σ is a perfect elimination ordering for it. The elimination graph is also called the *filled graph* of G_σ, or its *monotone transitive extension* [193, page 600]. The unordered graph (V, E_σ^*) is a *chordal embedding* or *chordal extension* of G. (However, not all chordal extensions can be constructed by elimination; see §6.6).

If G is a chordal graph and σ a perfect elimination ordering for it, then the higher neighborhoods of all vertices in G_σ are complete. Therefore graph elimination does not add any edges and $G_\sigma = G_\sigma^*$.

The following theorem shows how adjacency in the elimination graph depends on the connectivity in the original graph [195, lemma 4].

Theorem 6.1. Let $G_\sigma = (V, E, \sigma)$ be an ordered graph and $G_\sigma^* = (V, E_\sigma^*, \sigma)$ its elimination graph. The edge $\{v, w\}$ is in E_σ^* if and only if there exists a path $(v, z_1, \ldots, z_k, w)$ in G with interior vertices z_1, $\ldots$, z_k that precede v and w in the ordering σ (*i.e.*, $z_j \prec v$ and $z_j \prec w$ for $j = 1, \ldots, k$).

Proof. We show by induction on i that the following holds for $i = 0, \ldots, n-1$: if $v \prec w$ then $\{v, w\} \in E_i$ (with E_i defined in (6.1)) if and only if there exists a path $(v, z_1, \ldots, z_k, w)$ in G with interior vertices z_j that precede v in the ordering *and* satisfy $\sigma^{-1}(z_j) \leq i$. For $i = n-1$, the second condition is redundant and we obtain the statement in the theorem.

This result clearly holds for $i = 0$, since all edges in $E_0 = E$ form a path (v, w) in G. Suppose the result holds for E_{i-1}. Let $\{v, w\}$ be an edge in E_i, with $v \prec w$. We show there exists a path $(v, z_1, \ldots, z_k, w)$ in G with $z_j \prec v$ and $z_j \preceq \sigma(i)$ for $j = 1, \ldots, k$. If $\{v, w\} \in E_{i-1}$ the result follows immediately from the induction hypothesis. Otherwise, $\{v, w\} \in E_i \setminus E_{i-1}$ and by (6.1) this requires that $\{\sigma(i), v\}, \{\sigma(i), w\} \in E_{i-1}$ and $\sigma(i) \prec v$. By the induction assumption there exist paths $(v, z_1, \ldots, z_r = \sigma(i))$ and $(z_r = \sigma(i), z_{r+1}, \ldots, z_k, w)$ with interior vertices that precede $\sigma(i)$ in the ordering. Concatenating the two paths gives a path $(v, z_1, \ldots, z_r, \ldots, z_k, w)$ with $z_j \prec v$ and $z_j \preceq \sigma(i) \prec v$ for all j.

To show the converse, let $(v, z_1, \ldots, z_k, w)$ be a path in G with interior vertices that precede v and w, and satisfy $\sigma^{-1}(z_j) \leq i$. If $\sigma^{-1}(z_j) \leq i - 1$ for all interior vertices, then by the induction assumption, $\{v, w\} \in E_{i-1}$. Otherwise, let z_r be the interior vertex with $\sigma^{-1}(z_r) = i$. By the induction assumption, the existence of the paths $(v, z_1, \ldots, z_r = \sigma(i))$ and $(z_r = \sigma(i), \ldots, z_k, w)$ in G implies that $\{\sigma(i), v\} \in E_{i-1}$ and $\{\sigma(i), w\} \in E_{i-1}$. From (6.1), $\{v, w\} \in E_i$. $\square$

As an example, in the graph on the left-hand side of Figure 6.1 there is a path (b, c, d, e) with interior vertices c, d that have indices less than $\min\{\sigma^{-1}(b), \sigma^{-1}(e)\} = 4$. Therefore $\{b, e\}$ is a filled edge.

Efficient algorithms exist for analyzing an elimination graph G^*_σ based on the structure of the original graph G_σ, and without explicitly constructing E^*_σ. These algorithms have a complexity that is linear or nearly linear in the size $|E| + |V|$ of the graph G_σ. They include algorithms for constructing the elimination tree, calculating the lower and higher degrees of all vertices, identifying the clique representatives, and creating supernode partitions. These algorithms are briefly described in the following sections.

We will use a subscript to distinguish attributes of the elimination graph from those of the original graph:

$$\begin{aligned}\mathrm{adj}^-_*(v) &= \{w \mid \{w,v\} \in E^*_\sigma,\ w \prec v\},\\ \mathrm{adj}^+_*(v) &= \{w \mid \{w,v\} \in E^*_\sigma,\ w \succ v\}\end{aligned}$$

will denote the monotone neighborhoods in the elimination graph, and $\mathrm{row}_*(v) = \{v\} \cup \mathrm{adj}^-_*(v)$ and $\mathrm{col}_*(v) = \{v\} \cup \mathrm{adj}^+_*(v)$ will denote the closed monotone neighborhoods. The monotone degrees in the elimination graphs are $\deg^-_*(v) = |\mathrm{adj}^-_*(v)|$ and $\deg^+_*(v) = |\mathrm{adj}^+_*(v)|$.

6.2 Elimination tree

We first consider the problem of constructing the elimination tree of an elimination graph G^*_σ, directly from G_σ and without first carrying out the graph elimination. This is described in more detail in [152, section 4.2] [153, section 5.2] [65, section 4.1]. As in §4.3 we assume that G is connected.

Since the elimination graph G^*_σ is filled, the results of §4.3 apply to the elimination tree of G^*_σ. In particular, the closed lower neighborhood $\mathrm{row}_*(v)$ of a vertex v induces a subtree of the elimination tree with v as its root. As an example, in Figure 6.2, $\mathrm{row}_*(o) = \{c, d, e, g, h, i, o\}$ induces a subtree of the elimination tree with root o and leaves c, g.

The following theorem characterizes the leaves of the row subtrees [152, corollary 2.5] [153, corollary 3.6] [65, theorem 4.6]. It follows from an earlier theorem by Tarjan and Yannakakis [215, theorem 3].

Theorem 6.2. Let w be a proper descendant of vertex v in the elimination tree of G^*_σ. Then w is a leaf of the row subtree $\mathrm{row}_*(v)$ if and only if $\{w, v\} \in E$ and w does not have a proper descendant u with $\{u, v\} \in E$.

Proof. We show the following equivalent statement: a proper descendant w of v is in the row subtree $\mathrm{row}_*(v)$ if and only if w has a descendant u with $\{u, v\} \in E$.

Assume u is a descendant of w in the elimination tree, and adjacent to v in G. We have $w = p^k(u)$ for some $k \geq 0$, where $p(\cdot)$ is

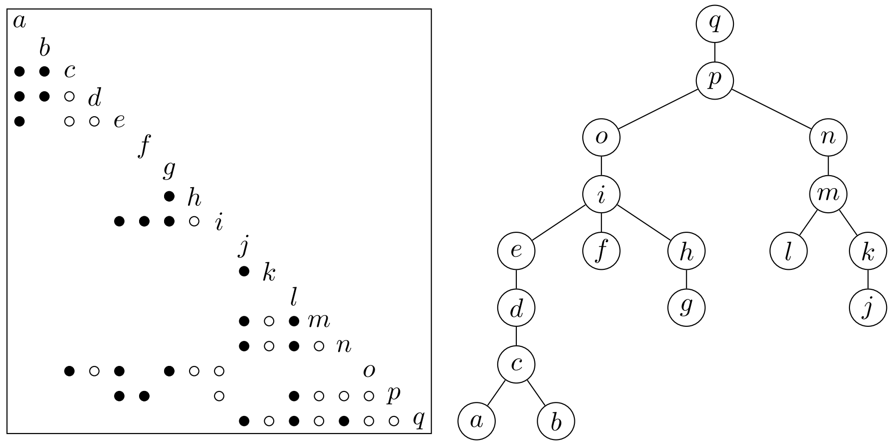


Figure 6.2: Ordered graph (filled circles), its elimination graph (with fill edges shown as open circles), and the elimination tree of the elimination graph.

the parent function in the elimination tree. For $j = 1, \ldots, k$, we have $\{p^{j-1}(u), p^j(u)\} \in E_\sigma^*$, so by Theorem 6.1 there exists a path from $p^{j-1}(u)$ to $p^j(u)$ in G, with interior vertices that precede $p^{j-1}(u)$ in the ordering. Concatenating these paths gives a path $(u, z_1, \ldots, z_k, w)$ in G with interior vertices that precede w in the ordering. Appending the edge $\{u, v\}$ gives a path $(v, u, z_1, \ldots, z_k, w)$ from v to $w \prec v$ in G with interior vertices that precede w. By Theorem 6.1 this implies $\{v, w\} \in E_\sigma^*$. This shows the 'if' direction of the theorem.

Next the 'only if' direction. Suppose $w \prec v$ and $\{w, v\} \in E_\sigma^*$. From Theorem 6.1, there exists a path $(w, z_1, \ldots, z_k, v)$ in G with interior vertices z_i that precede w in the ordering. If $k = 0$, we have $\{w, v\} \in E$ and the result holds with $u = w$. Suppose $k \geq 1$. Let z_j be the interior vertex of the path $(w, z_1, \ldots, z_k, v)$ with the highest index $\sigma^{-1}(z_j)$. If $j = k$, then by Theorem 6.1, $\{z_k, w\} \in E_\sigma^*$. Therefore z_k is a descendant of w in T and the result holds with $u = z_k$. If $j < k$, the existence of the paths $(w, z_1, \ldots, z_j)$ and $(z_j, z_{j+1}, \ldots, z_k)$ in G implies, again by Theorem 6.1, that $\{w, z_j\} \in E_\sigma^*$ and $\{z_j, z_k\} \in E_\sigma^*$. Moreover $z_k \prec z_j \prec w$, so z_k is a descendant of z_j in T and z_j is a descendant of w. Hence z_k is a descendant of w in T and the result holds with $u = z_k$. □

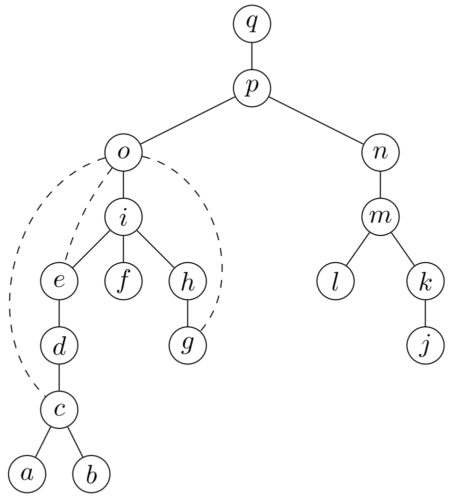


Figure 6.3: The elimination tree of Figure 6.2. The dashed lines are the edges of E that connect o to vertices in $\mathrm{adj}^-(o)$. These edges allow us to identify vertices c and g as the leaves of the row subtree $\mathrm{row}_*(o)$.

The theorem is illustrated in Figure 6.3. The leaves of $\mathrm{row}_*(o)$ are c and g, because these two vertices are adjacent to o in the graph G and no proper descendants of them are adjacent to o in G. The vertex e is also adjacent to o in G, but it has a descendant (namely, c) which is adjacent to o, and therefore e is not a leaf of the row subtree $\mathrm{row}(o)$.

The theorem leads to a simple recursive algorithm for constructing the elimination tree of the elimination graph. We visit the vertices in the order given by σ. The purpose of the cycle in which the vertex v is processed is to identify all the children of v in the elimination tree and to assign v as their parent. To do this we use the fact that $\mathrm{adj}^-(v)$ is a subset of the vertices of $\mathrm{row}_*(v)$ and that it includes all the leaves of $\mathrm{row}_*(v)$. Moreover when we process v, the children of every vertex in the row subtree, except v itself, are known, since they precede v in the ordering. We can therefore find the children of v simply by tracing the path from each vertex of $\mathrm{adj}^-(v)$ to v in the elimination tree until we reach a vertex w that has not been assigned a parent yet. Such a vertex must be a child of v. A straightforward but inefficient implementation of this idea is as follows [152, page 138].

Algorithm 6.1 (Elimination tree of elimination graph).

Input. An ordered graph $G_\sigma = (V, E, \sigma)$.

Output. The parent function $p(\cdot)$ of the elimination tree of the elimination graph G_σ^*.

Algorithm. Initialize the parent function as $p(v) := v$ for all v. For $i = 1, \ldots, n$, take $v = \sigma(i)$ and repeat the following steps for each $w \in \text{adj}^-(v)$.

- $u := w$.
- While $u \neq p(u)$, repeat $u := p(u)$.
- $p(u) := v$.

Algorithm 6.1 is inefficient because paths in the elimination tree are traced multiple times. Consider the example of Figures 6.2 and 6.3. When the algorithm arrives at vertex o, the elimination tree has been partially constructed, up to the vertices i and n, and their descendants. We have $\text{adj}^-(o) = \{c, e, g\}$, so in the iteration with $v = o$ we first trace the path from c to i and set $p(i) = v$. We then needlessly trace the paths from e to o and from g to o (both paths end at o because $p(o) = o$).

An improved version of Algorithm 6.1 keeps a record, for each vertex, of the most distant ancestor known from previous cycles in the algorithm. This information is used to skip segments of the paths in the search process. The technique is called *path compression* and is described in detail in [152, algorithm 4.2] [65, page 41].

The complexity of the elimination tree algorithm with path compression is $O(m \log n)$ in the worst case, where $m = |E|$ [153, theorem 5.2], and has been observed to be nearly linear in m in practice [65, page 41]. We refer the reader to [152, 153, 65] for further discussion.

6.3 Postordering

In §4.6 we mentioned that if σ is a perfect elimination ordering for a chordal graph G, then any topological reordering ρ of the elimination tree of G_σ is also a perfect elimination ordering for G, with the same elimination tree. A similar property holds for elimination graphs. If ρ is a topological reordering of the elimination tree of an elimination

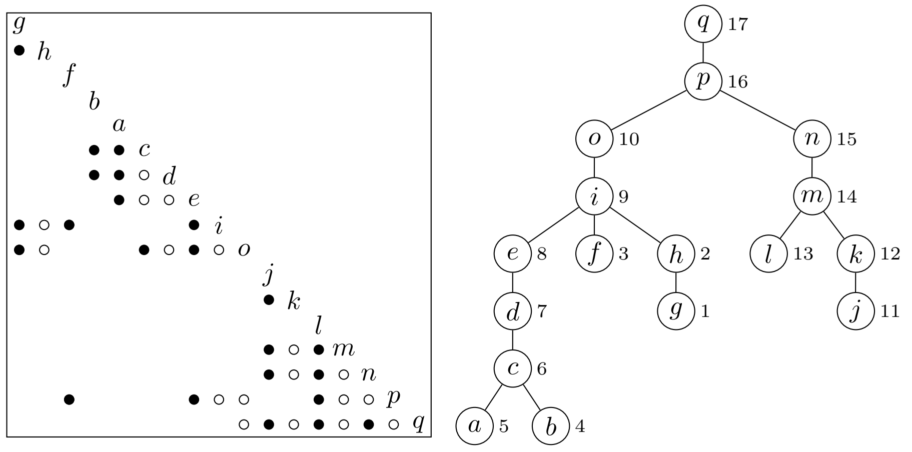


Figure 6.4: *Right.* Postordering of the elimination tree in Figure 6.2. Numbers next to the vertices indicate the position in the ordering. *Left.* Elimination graph for the topological order. Note that the fill-in is identical to the fill-in in the elimination graph of Figure 6.2.

graph $G^*_\sigma = (V, E^*_\sigma, \sigma)$, then the elimination graph $G^*_\rho = (V, E^*_\rho, \rho)$ has the same edge set $E^*_\rho = E^*_\sigma$ as G^*_σ and the same elimination tree [153, theorem 6.1][65, theorem 4.8]. This follows from the fact that a topological reordering preserves the ordering of the vertices along every path from leaf to root of the elimination tree (see §4.6). This is illustrated in Figure 6.4.

After computing the elimination tree, we are therefore free to reorder the vertices using any topological reordering of the elimination tree. Choosing a *postordering* of the elimination tree simplifies the analysis of elimination graphs considerably. We illustrate the simplifications that result from using a postordering with two examples. Assume σ is a postordering of the elimination tree of G^*_σ. In addition we assume that for every vertex v, we know $\mathrm{lev}(v)$ (the level in the elimination tree, *i.e.*, distance to the root) and the first descendant $\mathrm{fdesc}(v)$ (the descendant of v with the lowest index). As explained in §2.3 this information can be collected during the depth-first-search traversal of the elimination tree used to find a postordering. Suppose $\mathrm{adj}^-(v) = \{w_1, w_2, \ldots, w_k\}$ with $w_1 \prec w_2 \prec \cdots \prec w_k$.

- *Membership of row subtree.* Theorem 6.2 implies that a vertex u is in $\mathrm{row}_*(v)$ if u is a descendant of v and an ancestor of one of the vertices w_i. In a postordering this can be verified by checking that the inequalities $\mathrm{fdesc}(u) \preceq w_i \preceq u \preceq v$ hold for some w_i.

- *Leaves of row subtree.* Theorem 6.2 characterizes the leaves of the row subtrees $\mathrm{row}_*(v)$ as follows: w is a leaf of the $\mathrm{row}_*(v)$ if $w \in \mathrm{adj}^-(v)$ and no other vertex in $\mathrm{adj}^-(v)$ is a descendant of w. This is easy to check if σ is a postordering of the elimination tree: w_j is a leaf of $\mathrm{row}_*(v)$ if either $j = 1$, or $j > 1$ and $\mathrm{fdesc}(w_j) \succ w_{j-1}$ [152, theorem 4.1]. For example, in Figure 6.4, we have $\mathrm{adj}^-(o) = \{g, c, e\}$ with $g \prec c \prec e$. The first descendants are $\mathrm{fdesc}(g) = g$, $\mathrm{fdesc}(c) = \mathrm{fdesc}(e) = b$. Therefore the leaves of $\mathrm{row}_*(o)$ are g (the first vertex in $\mathrm{adj}^-(o)$) and c (because $\mathrm{fdesc}(c) = b \succ g$). The vertex e is not a leaf because $\mathrm{fdesc}(e) = b \prec c$.

6.4 Monotone degrees, cliques, and supernodes

The next task we consider is the calculation of the monotone degrees $\deg^-_*(v) = |\mathrm{adj}^-_*(v)|$ and $\deg^+_*(v) = |\mathrm{adj}^+_*(v)|$ for all vertices. Efficient algorithms for computing the monotone degrees were developed by Gilbert, Ng, and Peyton [94]. These algorithms are based on the path decomposition methods of §2.4 and have a complexity approximately linear in $|E|$.

The row subtrees of the filled graph form a family of subtrees of the elimination tree. The root of the row subtree $\mathrm{row}_*(v)$ is v; its leaves form a subset of $\mathrm{adj}^-(v)$ and are easily identified in a postordering, as we saw in the last section. The path decomposition methods of §2.4 can therefore be used to determine the number of edges in each row subtree $\mathrm{row}_*(u)$, *i.e.*, $\deg^-_*(u)$, and the number of row subtrees each vertex v belongs to, *i.e.*, $\deg^+_*(v) + 1$. See [94] for details.

Knowledge of the lower or higher degrees immediately gives us the size of the edge set $|E^*_\sigma|$ of the elimination graph, which is equal to the sum of the lower or the higher degrees, and the size of the fill-in $|E^*_\sigma| - |E|$. This also provides a practical test for checking whether a given ordering σ is a perfect elimination ordering.

As explained in §4.4 and §4.5 the higher degrees and the elimination tree are also sufficient to determine the cliques in the filled graph, construct a clique tree, and a supernodal elimination tree.

6.5 Filled graph

The edge set of the elimination graph can be generated in various ways, for example, directly from the definition (6.1). If the elimination tree of the filled graph is known, the calculations can be organized as a recursion on the elimination tree, using the equation

$$\mathrm{col}_*(v) = \mathrm{col}(v) \cup \bigcup_{w \in \mathrm{ch}(v)} \mathrm{adj}_*^+(w). \tag{6.2}$$

Here $\mathrm{ch}(v)$ is the set of children of v in the elimination tree of the elimination graph. To find the higher neighborhoods of the filled graph, one starts at the leaves of the elimination tree (with $\mathrm{adj}_*^+(v) = \emptyset$), enumerates the vertices in a topological ordering, and uses (6.2) to find the higher neighborhood of each vertex from the higher neighborhoods of its children.

6.6 Triangulation

A *triangulation*, or *chordal embedding*, or *chordal extension* of a graph $G = (V, E)$ is a chordal graph $G' = (V, E')$ with $E \subset E'$. The graph elimination process discussed in the previous sections gives a method for finding triangulations: we choose an ordering σ and apply graph elimination to the graph $G_\sigma = (V, E, \sigma)$; the graph (V, E_σ^*) is a triangulation of G. Not all triangulations can be generated by graph elimination; a simple example is shown in Figure 6.5 [180, page 627]. However, if $G' = (V, E')$ is a triangulation and σ is a perfect elimination ordering for G', then $E_\sigma^* \subseteq E'$, so by graph elimination with ordering σ we find a triangulation (V, E_σ^*) which is at least as efficient as G'. Put differently, all (inclusion-)*minimal* triangulations can be obtained by graph elimination [180, lemma 1]. In this section we therefore restrict the discussion to triangulations (V, E_σ^*) obtained by graph elimination.

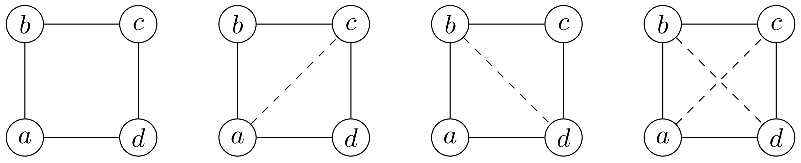


Figure 6.5: Graphs (b), (c), (d) are triangulations of the graph (a). Graphs (b) and (c) can be generated by applying graph elimination to (a) using an ordering that starts with vertices b or d, or with a or d, respectively. Graph (d), however, cannot be obtained by graph elimination of (a).

In the following chapters we describe several graph and sparse matrix problems that are easily solved when the graph is chordal, by algorithms that can be formulated as recursions over a junction tree. When applied to a triangulation these algorithms can be used to give exact or approximate solutions for non-chordal graphs. Clearly, the efficiency or accuracy of such approaches varies with the size and properties of the triangulation. Two criteria in particular are important when judging the quality of a triangulation. The first is the amount of fill, $|E^*_\sigma| - |E|$, or, equivalently, the sum of the higher degrees in the triangulation:

$$\sum_{v \in V} \deg^+_*(v) = |E^*_\sigma|.$$

An ordering that minimizes this quantity over all possible orderings is called a *minimum* ordering. Yannakakis has shown that finding minimum orderings is an NP-complete problem [240]. The second criterion is the size of the largest clique in the triangulation or, equivalently, the maximum higher degree

$$\max_{v \in V} \deg^+_*(v)$$

(this is one less than the maximum clique size; see §4.4). The minimum value of this quantity, over all possible orderings, is known as the *treewidth* of G [191, 39, 40]. Determining the treewidth, *i.e.*, finding the ordering that minimizes the largest clique size in the triangulation, is also NP-complete [16].

A more tractable problem is the computation of *minimal* orderings. An ordering σ is minimal if there exists no ordering ρ for which $E^*_\rho \subset E^*_\sigma$. Minimal ordering schemes also provide a test for chordality of a graph. Since a graph is chordal if and only if it has a perfect elimination ordering σ, *i.e.*, an ordering with zero fill-in ($F_\sigma = \emptyset$), all minimal orderings of a chordal graph are perfect elimination orderings. One can therefore test chordality by applying any minimal ordering algorithm and checking whether the result is a perfect elimination ordering.

The first minimal ordering algorithms were proposed in [180, 179, 195]. The algorithm by Ohtsuki [179] and Rose, Tarjan, and Lueker [195] have an $O(mn)$ complexity, where $m = |E|$ and $n = |V|$. The algorithm in [195] is an extension of the $O(m + n)$ Lex-BFS algorithm for finding a perfect elimination ordering for chordal graphs, and is now known as Lex-M. More recently, a similar generalization of the MCS algorithm, which finds a perfect elimination ordering of chordal graphs in $O(m+n)$ time, was presented in [29]. The complexity of this MCS-M algorithm for finding minimal orderings is also $O(mn)$.

Minimality of an ordering method is no guarantee for low fill-in. The most common ordering methods used in practice are heuristic fill-reducing ordering methods that are not necessarily minimal, such as the minimum degree or nested dissection ordering methods [65, chapter 7]. Several algorithms have also been developed for finding a minimal triangulation that improves on a given non-minimal triangulation, for example, the triangulation produced by a fill-reducing ordering heuristic. The algorithms in [37, 186, 30, 114] are examples. For more discussion of minimal triangulations we refer the reader to the survey [113].

7

Discrete Applications of Graph Elimination

The terminology of the previous sections (graph elimination, elimination tree, ...) refers in the first place to Gauss elimination and Cholesky factorization of sparse positive definite matrices, a subject that will be covered in Chapter 9. Graph elimination and the associated tree structures also appear in a wide range of applications outside matrix algebra, and they have a long and independent history in many disciplines. In this section we discuss a few examples.

We use the term *index set* for an ordered sequence of distinct integers. If x is an n-tuple $(x_1, x_2, \ldots, x_n)$ and $\beta = (\beta(1), \beta(2), \ldots, \beta(r))$ an index set with elements in $\{1, 2, \ldots, n\}$, then x_β denotes the r-tuple $(x_{\beta(1)}, x_{\beta(2)}, \ldots, x_{\beta(r)})$. If $X_1, \ldots, X_n$ are sets, then X_β denotes the set $X_{\beta(1)} \times X_{\beta(2)} \times \cdots \times X_{\beta(r)}$. The difference between the index set β and the set $\{\beta(1), \beta(2), \ldots, \beta(r)\}$ is the ordering of the elements in β.

The applications discussed in this section involve a set of functions $f_1(x_{\beta_1})$, $f_2(x_{\beta_2})$, $\ldots$, $f_l(x_{\beta_l})$ that depend on different, possibly overlapping, subsets of a variable $x = (x_1, x_2, \ldots, x_n)$. The index set β_k specifies the arguments of the function f_k. We associate with the functions an undirected graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$ and an edge $\{i, j\} \in E$ between vertices i and j if and only if there

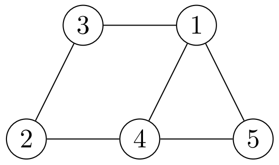


Figure 7.1: Interaction graph for 5 variables and functions $f_1(x_1, x_4, x_5)$, $f_2(x_1, x_3)$, $f_3(x_2, x_3)$, $f_4(x_2, x_4)$.

exists at least one index set β_k that contains the indices i and j. The graph G is called the *interaction graph* [32] or *co-occurrence graph* [59]. Figure 7.1 shows an example for five variables and functions

$$f_1(x_1, x_4, x_5), \quad f_2(x_1, x_3), \quad f_3(x_2, x_3), \quad f_4(x_2, x_4), \tag{7.1}$$

i.e., index sets $\beta_1 = (1, 4, 5)$, $\beta_2 = (1, 3)$, $\beta_3 = (2, 3)$, $\beta_4 = (2, 4)$. Note that the interaction graph does not uniquely determine the index sets. The graph in Figure 7.1, for example, is also the interaction graph for the functions $f_1(x_1, x_4)$, $f_2(x_1, x_5)$, $f_3(x_4, x_5)$, $f_4(x_1, x_3)$, $f_5(x_2, x_3)$, and $f_6(x_2, x_4)$. Throughout this section we assume that the interaction graph is connected. Problems with an interaction graph that is not connected can be decomposed by partitioning the functions $f_k(x_{\beta_k})$ in groups with connected interaction graphs.

Although many of the ideas discussed below extend to functions of continuous variables, we will assume that each variable x_i takes values in a finite set X_i of size $s_i = |X_i|$. Hence, the function $f_k(x_{\beta_k})$ can be thought of as a $|\beta_k|$-dimensional table with s_i cells in dimension i. The total number of cells in the table for f_k is $\prod_{i \in \beta_k} s_i$.

The algorithms discussed in this section can be described concisely in terms of the elimination graph of the interaction graph (or filled interaction graph) and depend on an elimination ordering of the vertices. We will see that the complexity of the algorithms depends on the sizes of the cliques in the filled interaction graph and is typically exponential as a function of the largest clique size. The methods are therefore practical only if the interaction graph has small treewidth and if an elimination ordering is known that results in an elimination graph with small cliques (see §6.6). The techniques for analyzing elimination graphs without explicitly constructing them that were presented in Chapter 6 are useful in this context, since they allow us to estimate the complexity for different elimination orderings at a cost that is roughly linear in the size of the interaction graph.

7.1 Dynamic programming

We first consider the problem of computing

$$\min_{x\in X} \sum_{k=1}^{l} f_k(x_{\beta_k}) \tag{7.2}$$

where $X = X_1 \times X_2 \times \cdots \times X_n$. The function $f(x) = \sum_k f_k(x_{\beta_k})$ is a sum of functions of subsets of the variables $x_1, x_2, \ldots, x_n$. In nonlinear optimization this type of structure is called *partial separability* (see §12.1). We discuss methods for evaluating (7.2) that exploit partial separability and are much more efficient than a complete enumeration of the $\prod_i s_i$ elements in X if the interaction graph is sufficiently sparse.

Graphical methods that take advantage of the interaction graph and their connection with graph elimination are discussed in [32], as an extension of dynamic programming for finite-state deterministic optimal control, and in [59], in a discussion of elimination algorithms for optimization over Boolean variables [118, chapter VI]. Deterministic dynamic programming is concerned with a special case in which $l = n-1$ and $\beta_k = (k, k+1)$, $k = 1, \ldots, n-1$, *i.e.*, the minimization of

$$f_1(x_1, x_2) + f_2(x_2, x_3) + \cdots + f_{n-1}(x_{n-1}, x_n). \tag{7.3}$$

The variable x_i typically represents a *state* at time i, and takes values in a finite set X_i; the function $f_i(x_i, x_{i+1})$ includes a cost of the state x_i and/or x_{i+1} plus the transition cost from state x_i to x_{i+1}. The interaction graph for this problem is a path: $E = \{\{i, i+1\} \mid i = 1, \ldots, n-1\}$. The *backward dynamic programming algorithm* computes

$$u_k(x_{k-1}) = \min_{x_k,\ldots,x_n} \left(f_{k-1}(x_{k-1}, x_k) + f_k(x_k, x_{k+1}) + \cdots + f_{n-1}(x_{n-1}, x_n)\right)$$

for $k = n, n-1, \ldots, 1$, via the iteration

$$\begin{aligned} u_n(x_{n-1}) &= \min_{x_n} f_{n-1}(x_{n-1}, x_n), \\ u_k(x_{k-1}) &= \min_{x_k} \left(f_{k-1}(x_{k-1}, x_k) + u_{k+1}(x_k) \right), \quad k = n-1, \ldots, 2, \\ u_1 &= \min_{x_1} u_2(x_1). \end{aligned}$$

The function $u_k(x_{k-1})$ is the optimal 'cost-to-go' from state x_{k-1} at time $k-1$. The final value u_1 is equal to the minimum of (7.3). The *forward* dynamic programming algorithm uses a similar recursion to compute the optimal 'cost-to-arrive' functions

$$u_k(x_{k+1}) = \min_{x_1,\ldots,x_k} \left(f_1(x_1, x_2) + f_2(x_2, x_3) + \cdots + f_k(x_k, x_{k+1}) \right)$$

for $k = 1, \ldots, n-1$, and then computes the minimum $\min_x f(x) = \min_{x_n} u_{n-1}(x_n)$. Both algorithms can be interpreted as eliminating the optimization variables (*i.e.*, optimizing over them) one by one, either in the numerical order (the forward algorithm) or inverse numerical order (the backward algorithm) [33, section 2.1].

The key difference between (7.3) and (7.2) is that the general problem allows functions $f_k(x_{\beta_k})$ of more than two arguments. As an example that illustrates the extension of the dynamic programming approach to problem (7.2) we take the functions (7.1) and the problem of minimizing

$$f(x) = f_1(x_1, x_4, x_5) + f_2(x_1, x_3) + f_3(x_2, x_3) + f_4(x_2, x_4). \tag{7.4}$$

To simplify the complexity analysis we will assume that $s_i = s$ for all i. We compare the cost of variable elimination with the complete enumeration of the s^5 elements in X. First consider the numerical elimination ordering $\sigma = (1, 2, 3, 4, 5)$. To eliminate x_1, we separate the first two terms in f, which depend on x_1, from the rest, and minimize their sum:

$$\min_{x_1} f(x) = u_1(x_3, x_4, x_5) + f_3(x_2, x_5) + f_4(x_2, x_4) \tag{7.5}$$

where $u_1(x_3, x_4, x_5) = \min_{x_1} (f_1(x_1, x_4, x_5) + f_2(x_1, x_3))$. Calculating u_1 requires an enumeration of the s^4 possible values of (x_1, x_3, x_4, x_5) in $X_1 \times X_3 \times X_4 \times X_5$. Next, we minimize (7.5) over x_2:

$$\min_{x_1, x_2} f(x) = u_1(x_3, x_4, x_5) + u_2(x_3, x_4) \tag{7.6}$$

where $u_2(x_3, x_4) = \min_{x_2}(f_3(x_2, x_3) + f_4(x_2, x_4))$. This calculation requires an enumeration of s^3 possible values of $(x_2, x_3, x_4) \in X_2 \times X_3 \times X_4$. Continuing in this manner with the other three variables, we obtain

$$\begin{aligned}\min_{x \in X} f(x) &= \min_{x_3, x_4, x_5} (u_1(x_3, x_4, x_5) + u_2(x_3, x_4)) \\ &= \min_{x_4, x_5} u_3(x_4, x_5) \\ &= \min_{x_5} u_4(x_5) \\ &= u_5\end{aligned}$$

where

$$\begin{aligned}u_3(x_4, x_5) &= \min_{x_3} (u_1(x_3, x_4, x_5) + u_2(x_3, x_4)), \\ u_4(x_5) &= \min_{x_4} u_3(x_4, x_5).\end{aligned}$$

Computing u_3, u_4, and u_5 requires enumerating s^3, s^2, and s values, respectively, so the total cost is dominated by the minimization over x_1, which has a complexity s^4. The algorithm can be summarized in one line by the following nested minimization formula for $\min_x f(x)$:

$$\begin{aligned}\min_x f(x) &= \min_{x_5} \min_{x_4} \min_{x_3} \Big(\min_{x_2} (f_3(x_2, x_5) + f_4(x_2, x_4)) \\ &\quad + \min_{x_1} (f_1(x_1, x_4, x_5) + f_2(x_1, x_3)) \Big).\end{aligned}$$

The complexity is lower for some other elimination orders. For example if we use $\sigma = (5, 1, 2, 3, 4)$, *i.e.*, calculate the minimum via the formula

$$\begin{aligned}\min_x f(x) &= \min_{x_4} \min_{x_3} \Big(\min_{x_2} (f_3(x_2, x_3) + f_4(x_2, x_4)) \\ &\quad + \min_{x_1} \Big(f_2(x_1, x_3) + \min_{x_5} f_1(x_1, x_4, x_5) \Big) \Big), \end{aligned} \tag{7.7}$$

then the most expensive steps are the minimizations over x_5, x_1, and x_2, which each require an enumeration of s^3 values.

The variable elimination algorithm has a simple graph interpretation, illustrated in Figure 7.2. In the first step, when we minimize over $x_{\sigma(1)}$, we enumerate all values of $x_{\sigma(1)}$ and of the variables associated with the vertices adjacent to $\sigma(1)$ in the interaction graph G.

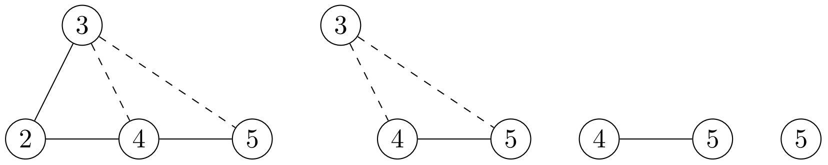


Figure 7.2: The four interaction graphs after elimination of the variables x_1, x_2, x_3, x_4 in Figure 7.1. Dashed lines indicate fill edges.

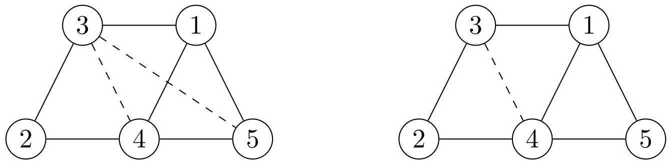


Figure 7.3: The elimination graphs for the graph of Figure 7.1 for the ordering $\sigma = (1, 2, 3, 4, 5)$ (left) and $\sigma = (5, 1, 2, 3, 4)$ (right).

To construct the interaction graph for the function $\min_{x_{\sigma(1)}} f(x)$ we add edges between the vertices adjacent to $\sigma(1)$ in G and then remove vertex $\sigma(1)$. In the notation of §6, this is the subgraph of the graph (V, E_1) induced by the vertices $\sigma(2)$, ..., $\sigma(n)$, where E_1 is defined in equation (6.1). In the second step, when we minimize over $x_{\sigma(2)}$, we enumerate all possible combinations of $x_{\sigma(2)}$ and the vertices adjacent to $\sigma(2)$ in the interaction graph of $\min_{x_{\sigma(1)}} f(x)$, add edges between the vertices adjacent to $\sigma(2)$ that are not already adjacent, and then remove $\sigma(2)$. The result is the interaction graph for $\min_{x_{\sigma(1)},x_{\sigma(2)}} f(x)$. Using the notation of (6.1), this interaction graph can be described as the subgraph of (V, E_2) induced by the vertices $\sigma(3)$, ..., $\sigma(n)$. The elimination process continues until all vertices have been removed.

The algorithm can therefore be described in terms of the filled interaction graph G^*_σ (Figure 7.3). We start with the set of functions $F = \{f_1(x_{\beta_1}), \ldots, f_l(x_{\beta_l})\}$ and enumerate the vertices in V in the order σ. When considering vertex $\sigma(i)$ we remove from F the functions that have the variable $x_{\sigma(i)}$ as one of their arguments and minimize the sum of these functions over $x_{\sigma(i)}$. This defines a new function, with the variables indexed by $\mathrm{adj}^+_*(\sigma(i))$ as its arguments, where $\mathrm{adj}^+_*(\sigma(i))$ is the higher neighborhood of $\sigma(i)$ in the filled interaction graph G^*_σ. We add this new function to F and move on to the next vertex.

The cycle in which we minimize over $x_{\sigma(i)}$ requires enumerating

$$s_{\sigma(i)} \prod_{j \in \mathrm{adj}_*^+(\sigma(i))} s_j = \prod_{j \in \mathrm{col}_*(\sigma(i))} s_j$$

values. If we assume that $s_j = s$ for $j = 1, \ldots, n$, this is equal to $s^{\deg_*^+(\sigma(i))+1}$. Hence, the complexity is exponential in the size of the largest clique of the filled interaction graph.

Algorithm 7.1 summarizes the method as a recursion on the elimination tree of the filled interaction graph. We use the notation α_i to denote the index set with the elements of $\mathrm{adj}_*^+(i)$, ordered numerically. We also define

$$F_i = \{k \in \{1, 2, \ldots, l\} \mid \operatorname*{argmin}_{j \in \beta_k} \sigma^{-1}(j) = i\}, \quad i = 1, \ldots, n.$$

We say that the functions $f_k(x_{\beta_k})$ with $k \in F_i$ are *assigned* to vertex i. The function $f_k(x_{\beta_k})$ is assigned to vertex i if i is the first index (in the order σ) of the elements of β_k.

Algorithm 7.1 (Minimization of a partially separable discrete function).

Input. A partially separable function $f(x) = \sum_{k=1,\ldots,l} f_k(x_{\beta_k})$ of variables $(x_1, \ldots, x_n) \in X = X_1 \times X_2 \times \cdots \times X_n$, and an elimination tree of a filled interaction graph.

Output. The minimum of $f(x)$ over X.

Algorithm. Enumerate the vertices of the elimination tree in topological order. For each vertex i, compute

$$u_i(x_{\alpha_i}) = \min_{x_i \in X_i} \left(\sum_{k \in F_i} f_k(x_{\beta_k}) + \sum_{j \in \mathrm{ch}(i)} u_j(x_{\alpha_j}) \right). \qquad (7.8)$$

The second sum is understood to be zero if i is a leaf vertex.

On completion, $u_{\sigma(n)}$ is a constant, equal to $\min_x f(x)$. The function minimized on the right-hand side of (7.8) depends on the variables indexed by the index sets β_k for $k \in F_i$ and α_j for $j \in \mathrm{ch}(i)$. The vertices in β_k, for $k \in F_i$, are in the closed neighborhood $\mathrm{col}(i)$ of vertex i in the interaction graph (by definition of F_i), and therefore also in the closed neighborhood $\mathrm{col}_*(i)$ of the filled interaction graph. The vertices in α_j

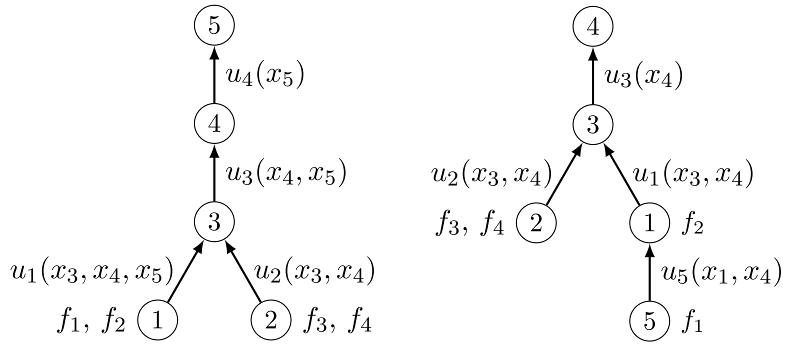


Figure 7.4: Elimination trees of the filled interaction graph of Figure 7.1 for the elimination orders $\sigma = (1, 2, 3, 4, 5)$ (left) and $\sigma = (5, 1, 2, 3, 4)$ (right). Each of the functions $f_1(x_1, x_4, x_5)$, $f_2(x_1, x_3)$, $f_3(x_2, x_3)$, $f_4(x_2, x_4)$ is assigned to the first vertex (in the ordering σ) of the vertices associated with its arguments. The function u_i is the sum of the functions assigned to the descendants of vertex i, minimized over all the variables associated with these descendants. Each function u_i is a local variable, and passed from vertex i to its parent, as indicated by the arrows.

for $j \in \mathrm{ch}(i)$ are included in $\mathrm{col}_*(i)$ by properties of the elimination tree (in particular, (4.3) applied to the filled interaction graph). After minimizing over x_i, the right-hand side of (7.8) is therefore a function of the variables indexed by $\mathrm{adj}_*^+(i) = \mathrm{col}_*(i) \setminus \{i\}$, *i.e.*, of x_{α_i}.

The functions u_i in (7.8) have the following meaning. For each i, the function $u_i(x_{\alpha_i})$ is the sum of the functions $f_k(x_{\beta_k})$ assigned to the descendants of vertex i in the elimination tree, minimized over the variables indexed by the descendants.

Figure 7.4 illustrates the algorithm for the two elimination orderings of the example. To apply Algorithm 7.1 with the second ordering, we take $F_5 = \{1\}$, $F_1 = \{2\}$, $F_2 = \{3, 4\}$, $F_3 = F_4 = \emptyset$, $\alpha_5 = (1, 4)$, $\alpha_1 = (3, 4)$, $\alpha_2 = (3, 4)$, $\alpha_3 = (4)$, $\alpha_1 = \emptyset$. The functions u_i are

$$\begin{aligned}
u_5(x_1, x_4) &= \min_{x_5} f_1(x_1, x_4, x_5), \\
u_1(x_3, x_4) &= \min_{x_1} \left(f_2(x_1, x_3) + u_5(x_1, x_4)\right) \\
&= \min_{x_1, x_5} \left(f_1(x_1, x_4, x_5 + f_2(x_1, x_3))\right), \\
u_2(x_3, x_4) &= \min_{x_2} \left(f_3(x_2, x_3) + f_4(x_2, x_4)\right),
\end{aligned}$$

$$\begin{aligned}
u_3(x_4) &= \min_{x_3} \left(u_2(x_3,x_4) + u_1(x_3,x_4)\right) \\
&= \min_{x_1,x_2,x_3,x_5} \left(f_1(x_1,x_4,x_5) + f_2(x_1,x_3) + f_3(x_2,x_3) \right. \\
&\quad \left. + \; f_4(x_2,x_4)\right), \\
u_4 &= \min_{x_4} u_3(x_4) \\
&= \min_{x} \left(f_1(x_1,x_4,x_5) + f_2(x_1,x_3) + f_3(x_2,x_3) + f_4(x_2,x_4)\right).
\end{aligned}$$

The algorithm evaluates the nested formula (7.7), with the variables u_i as intermediate variables for the results of the minimization steps.

Algorithm 7.2 is a supernodal version of Algorithm 7.1, based on a recursion over a supernodal elimination tree for the filled interaction graph G^*_σ. Here we assign each function $f_k(\beta_k)$ to the supernode $\mathrm{snd}(i)$ that contains the first index (in the ordering σ) of the arguments x_{β_k}. (Recall from §4.4 and §4.5 that the supernodes $\mathrm{snd}(i)$ in a supernode partition, form a partition of the vertex set V, so this definition assigns every function to a unique supernode.) The set of functions assigned to the supernode $\mathrm{snd}(i)$ is therefore given by

$$F_i = \{k \in \{1,2,\ldots,l\} \mid \operatorname*{argmin}_{j\in\beta_k} \sigma^{-1}(j) \in \mathrm{snd}(i)\}, \quad i \in V^{\mathrm{s}}.$$

For each supernode representative $i \in V^{\mathrm{s}}$ we define index sets ν_i, containing the elements of $\mathrm{snd}(i)$ ordered numerically, and α_i, containing the elements of $\mathrm{col}_*(i)\setminus\mathrm{snd}(i)$, also ordered numerically. At each step of the supernodal algorithm we minimize over a subset x_{ν_i} of the variables, as opposed to over a single variable x_i in the (nodal) Algorithm 7.1.

Algorithm 7.2 (Minimization of a partially separable discrete function)**.**

Input. A partially separable function $f(x) = \sum_k f_k(x_{\beta_k})$ of variables $(x_1,\ldots,x_n) \in X = X_1 \times X_2 \times \cdots \times X_n$, and a supernodal elimination tree T^{s} of a filled interaction graph.

Output. The minimum of $f(x)$ over X.

Algorithm. Enumerate the representative vertices $i \in V^{\mathrm{s}}$ of T^{s} in topological order. For each supernode representative i, compute

$$u_i(x_{\alpha_i}) = \min_{x_{\nu_i}\in X_{\nu_i}} \left(\sum_{k\in F_i} f_k(x_{\beta_k}) + \sum_{j\in\mathrm{ch}(i)} u_j(x_{\alpha_j}) \right). \tag{7.9}$$

The second sum is understood to be zero if i is a leaf vertex.

The function $u_i(x_{\alpha_i})$ in the recursion is the sum of the functions $f_k(x_{\beta_k})$ assigned to the descendants of the vertex i in the supernodal elimination tree, minimized over the variables in the supernodes represented by these descendants. On completion, i is the representative vertex of the root of the supernodal elimination tree and u_i is a constant equal to $\min_{x\in X} f(x)$. The nodal Algorithm 7.1 is a special case of the supernodal Algorithm 7.2, if we use the supernodal partition $V^{\mathrm{s}} = V$ (see §4.5). Algorithms 7.2 and 7.1 both have a complexity that is exponential in the size of the largest clique in the filled interaction graph. An advantage of the supernodal algorithm is that it requires fewer and smaller intermediate variables u_i.

Figures 7.5 and 7.6 illustrate the supernodal algorithm for an example with eight variables and five functions. The algorithm computes the minimum of

$$\begin{aligned} f(x) &= f_1(x_1,x_4,x_5) + f_2(x_2,x_3,x_4,x_6) + f_3(x_5,x_8) \\ &\quad + f_4(x_6,x_8) + f_5(x_7,x_8) \end{aligned}$$

via the nested minimization formula

$$\begin{aligned} \min_x f(x) = \min_{x_5,x_6,x_8} \Big(& f_3(x_5,x_8) + f_4(x_6,x_8) + \min_{x_7} f_5(x_7,x_8) \\ & + \min_{x_4}\Big(\min_{x_2,x_3} f_2(x_2,x_3,x_4,x_6) + \min_{x_1} f_1(x_1,x_4,x_5)\Big)\Big). \end{aligned} \tag{7.10}$$

The intermediate variables u_i in Figure 7.6 are defined as

$$\begin{aligned} u_1(x_4,x_5) &= \min_{x_1} f_1(x_1,x_4,x_5), \\ u_2(x_4,x_6) &= \min_{x_2,x_3} f_2(x_2,x_3,x_4,x_6), \\ u_4(x_5,x_6) &= \min_{x_4} \left(u_1(x_4,x_5) + u_2(x_4,x_6)\right), \\ u_7(x_8) &= \min_{x_7} f_5(x_7,x_8), \\ u_5 &= \min_{x_5,x_6,x_8} \left(f_3(x_5,x_8) + f_4(x_6,x_8) + u_4(x_5,x_6) + u_7(x_8)\right). \end{aligned}$$

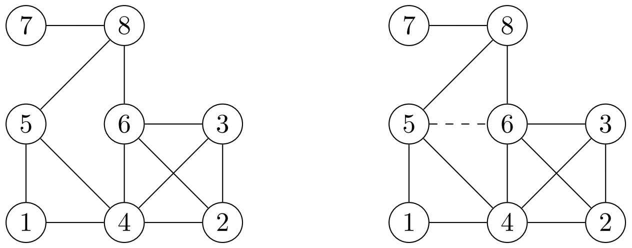


Figure 7.5: Interaction graph for 8 variables and functions $f_1(x_1, x_4, x_5)$, $f_2(x_2, x_3, x_4, x_6)$, $f_3(x_5, x_8)$, $f_4(x_6, x_8)$, $f_5(x_7, x_8)$ and elimination graph for the numerical elimination ordering.

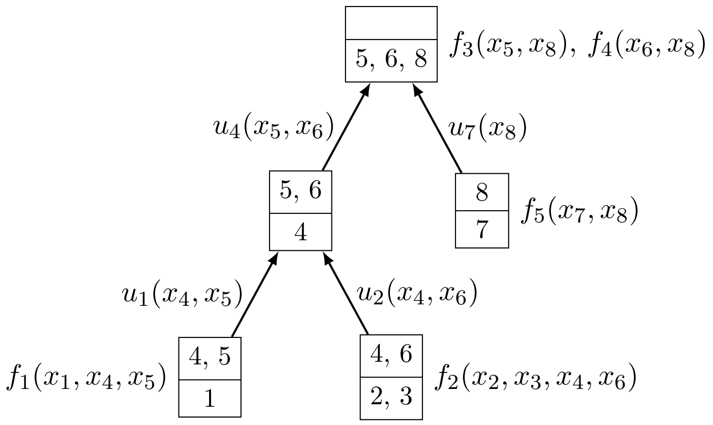


Figure 7.6: Clique tree and intermediate variables for Algorithm 7.2, applied to the maximal supernode partition associated with the clique tree.

7.2 Probabilistic networks

In the second application, we compute the sum of a product of functions

$$\sum_{x\in X}\prod_{k=1}^{l} f_k(x_{\beta_k}). \tag{7.11}$$

The discussion of the previous section carries over almost verbatim to this problem, with summation over $x \in X$ replacing minimization over X, and the product of the functions f_k replacing their sum. In the example of Figure 7.6, we replace (7.10) by

$$\begin{aligned}
&\sum_{x\in X} f_1(x_1,x_4,x_5)f_2(x_2,x_3,x_4,x_6)f_3(x_5,x_8)f_4(x_6,x_8)f_5(x_7,x_8)\\
&\quad= \sum_{x_5,x_6,x_8}\Bigg(f_3(x_5,x_8)f_4(x_6,x_8)(\sum_{x_7} f_5(x_7,x_8))\\
&\qquad\times\Bigg(\sum_{x_4}(\sum_{x_2,x_3} f_2(x_2,x_3,x_4,x_6))(\sum_{x_1} f_1(x_1,x_4,x_5))\Bigg)\Bigg).
\end{aligned}$$

To modify Algorithms 7.1 and 7.2 we only need to change (7.8) to

$$u_i(x_{\alpha_i}) = \sum_{x_i\in X_i}\left(\prod_{k\in F_i} f_k(x_{\beta_k})\prod_{j\in \mathrm{ch}(i)} u_j(x_{\alpha_j})\right)$$

and (7.9) to

$$u_i(x_{\alpha_i}) = \sum_{x_{\nu_i}\in X_{\nu_i}}\left(\prod_{k\in F_i} f_k(x_{\beta_k})\prod_{j\in \mathrm{ch}(i)} u_j(x_{\alpha_j})\right). \tag{7.12}$$

Graphical methods for computing sums of the form (7.11) have been studied since the 1980s in the literature on probabilistic graphical models. In these applications the functions $f_k(x_{\beta_k})$ are nonnegative and $f(x) = \prod_k f_k(x_{\beta_k})$ is a factored probability distribution. In a *Markov network* a joint probability of n random variables $x_1, \ldots, x_n$ is expressed as a product

$$\Pr(x) = \prod_{k=1}^{l} f_k(x_{\beta_k}) \tag{7.13}$$

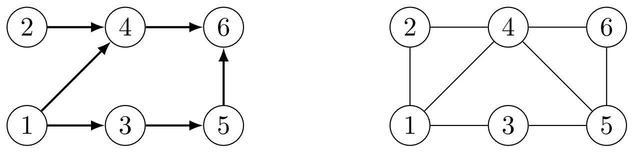


Figure 7.7: Bayesian network and its interaction graph.

of nonnegative functions $f_k(x_{\beta_k})$. The functions f_k are called *potential functions* or *factors*. The interaction graph of $\Pr(x)$ is an undirected *graphical model* of the distribution. In a *Bayesian network* [184, 64] a joint probability distribution of a random variable $(x_1, x_2, \ldots, x_n)$ is represented by a directed acyclic graph with vertices $V = \{1, 2, \ldots, n\}$. The probability of $x = (x_1, x_2, \ldots, x_n)$ is expressed as a product of conditional probabilities

$$\Pr(x) = \prod_{k=1,\ldots,n} \Pr\left(x_k \mid x_{\mathrm{par}(k)}\right), \tag{7.14}$$

where $\mathrm{par}(i)$ is an index set containing the *parents* of vertex i, *i.e.*, the vertices j that are connected to vertex i by a directed edge (j, i). If vertex i has no parents then $\mathrm{par}(i)$ is empty, and $\Pr(x_i \mid x_{\mathrm{par}}(i))$ is replaced with $\Pr(x_i)$ in (7.14). The joint probability distribution (7.14) is of the same factored form as (7.13), with $l = n$ and $f_k(x_{\beta_k}) = \Pr(x_k \mid x_{\mathrm{par}(k)})$. The index set β_k therefore contains k and the elements of $\mathrm{par}(k)$. To construct the interaction graph from the directed acyclic graph that defines (7.14) we replace the directed edges by undirected edges and add an edge between every two non-adjacent vertices that are parents of the same vertex. Figure 7.7 illustrates this for the distribution

$$\Pr(x_1)\Pr(x_2)\Pr(x_3 \mid x_1)\Pr(x_4 \mid x_1, x_2)\Pr(x_5 \mid x_3)\Pr(x_6 \mid x_4, x_5).$$

Detailed surveys of probabilistic networks are available in several books, including [184, 146, 58, 227, 64, 133].

Two common tasks in applications of probabilistic networks are *inferencing* and *marginalization*. In the inferencing problem we are given observations of a subset of the variables and are asked to compute the posterior probability for the other variables. Consider a set of discrete random variables $x = (x_1, x_2, \ldots, x_n)$ with a (prior) joint probability of the form (7.13), where x takes values in the finite set $X = X_1 \times \cdots \times X_n$. Suppose we observe values a_i for some the variables x_i. If for the observed variables we replace the set X_i by the singleton $\{a_i\}$, then the posterior probability of x is given by

$$\Pr(x \mid \text{observations}) = \begin{cases} (1/S) \prod\limits_{k=1}^{l} f_k(x_{\beta_k}) & x \in X \\ 0 & \text{otherwise,} \end{cases}$$

where

$$S = \sum_{x \in X} \prod_{k=1}^{l} f_k(x_{\beta_k}). \tag{7.15}$$

The normalization constant S is a sum of the form (7.11) and can be computed using Algorithm 7.1 or 7.2 (with the max-sum update formulas replaced with sum-product updates).

The second problem is *marginalizing* a product function $f(x) = \prod_k f_k(x_{\beta_k})$ over a subset of the variables. If η is an index set in $\{1, 2, \ldots, n\}$, then we will use the notation $\bar{\eta}$ to denote a complementary index set, with the elements of $\{1, 2, \ldots, n\} \setminus \eta$. The *$\eta$-marginal* of f is defined as the function

$$g_\eta(x_\eta) = \sum_{x_{\bar{\eta}} \in X_{\bar{\eta}}} \prod_{k=1}^{l} f_k(x_{\beta_k}).$$

Suppose we compute S in (7.15) using the sum-product version of Algorithm 7.2, applied to a supernodal elimination tree with root r. The last step of the algorithm is to compute $u_r = \sum_{x_{\nu_r}} g_{\nu_r}(x_{\nu_r})$ where

$$g_{\nu_r}(x_{\nu_r}) = \prod_{k \in F_r} f_k(x_{\beta_k}) \prod_{j \in \mathrm{ch}(r)} u_j(x_{\alpha_j}) = \sum_{x_{\bar{\nu}_r} \in X_{\bar{\nu}_r}} f(x).$$

Hence, we also obtain the ν_r-marginal of f. If a maximal supernode partition is used (the supernodal elimination tree corresponds to a clique tree), then the elements of ν_r form the clique in the filled interaction graph that was chosen as the root of the clique tree.

An extension of the algorithm computes the marginals for all the cliques in the filled interaction graph, by adding a recursion in inverse topological ordering. This is summarized in Algorithm 7.3. Here the index sets ν_i, α_i, γ_i, and μ_i are defined as follows. As in Algorithm 7.2, ν_i contains the elements of $\mathrm{snd}(i)$ and α_i contains the elements of $\mathrm{col}_*(i) \setminus \mathrm{snd}(i)$. In addition, we define an index set γ_i with the elements of $\mathrm{col}_*(i)$, and an index set μ_i with the elements of $\mathrm{col}_*(q(i)) \setminus \mathrm{col}_*(i)$, where $q(i)$ is the parent of i in the supernodal elimination tree. Hence α_i contains the elements in the intersection of $\mathrm{col}_*(i) \cap \mathrm{col}_*(q(i))$, ν_i the difference $\mathrm{col}_*(i) \setminus \mathrm{col}_*(q(i))$ and μ_i the difference $\mathrm{col}_*(q(i)) \setminus \mathrm{col}_*(i)$.

Algorithm 7.3 (Marginals of a discrete product function).

Input. A product function $f(x) = \prod_{k=1,\ldots,l} f_k(x_{\beta_k})$ of variables $(x_1, \ldots, x_n) \in X = X_1 \times X_2 \times \cdots \times X_n$, and a supernodal elimination tree T^{s} of a filled interaction graph.

Output. The γ_i-marginals $g_i(\gamma_i) = \sum\limits_{x_{\bar{\gamma}_i} \in X_{\bar{\gamma}_i}} f(x)$ for $i \in V^{\mathrm{s}}$.

Algorithm.

- Enumerate the vertices $i \in V^{\mathrm{s}}$ of T^{s} in topological order. For each $i \in V^{\mathrm{s}}$, compute

$$u_i(x_{\alpha_i}) = \sum_{x_{\nu_i} \in X_{\nu_i}} \left(\prod_{k \in F_i} f_k(x_{\beta_k}) \prod_{j \in \mathrm{ch}(i)} u_j(x_{\alpha_j}) \right).$$

  The second product is understood to be one if i is a leaf vertex.

- Define $v_r = 1$ for the root r of T^{s}. Enumerate the vertices $i \in V^{\mathrm{s}}$ of T^{s} in inverse topological order. For each $i \in V^{\mathrm{s}}$ and all $j \in \mathrm{ch}(i)$, compute

$$v_j(x_{\alpha_j}) = \sum_{x_{\mu_j} \in X_{\mu_j}} \left(v_i(x_{\alpha_i}) \prod_{k \in F_i} f_k(x_{\beta_k}) \prod_{k \in \mathrm{ch}(i),\, k \neq j} u_k(x_{\alpha_k}) \right).$$

On completion of the algorithm, the γ_i-marginals of $f(x)$ are available as products

$$g_{\gamma_i}(x_{\gamma_i}) = v_i(x_{\alpha_i}) \prod_{j \in \mathrm{ch}(i)} u_j(x_{\alpha_j}) \prod_{k \in F_i} f_k(x_{\beta_k}).$$

If a maximal supernode partition is used, the sets γ_i are exactly the cliques of the filled interaction graph and we obtain all the clique marginals. If a non-maximal supernode partition is used the sets γ_i include all the cliques, but also some subsets of cliques.

Applied to the example of Figure 7.5, the first (bottom-up) half of the algorithm, illustrated in Figure 7.6, computes

$$\begin{aligned}
u_1(x_4, x_5) &= \sum_{x_1} f_1(x_1, x_4, x_5), \\
u_2(x_4, x_6) &= \sum_{x_2, x_3} f_2(x_2, x_3, x_4, x_6), \\
u_4(x_5, x_6) &= \sum_{x_4} u_1(x_4, x_5) u_2(x_4, x_6), \\
u_7(x_8) &= \sum_{x_7} f_5(x_7, x_8), \\
u_5 &= \sum_{x_5, x_6, x_8} f_3(x_5, x_8) f_4(x_6, x_8) u_4(x_5, x_6) u_7(x_8).
\end{aligned}$$

In the second part of the algorithm (Figure 7.8) we take $v_5 = 1$ and compute

$$\begin{aligned}
v_4(x_5, x_6) &= \sum_{x_8} f_3(x_5, x_8) f_4(x_6, x_8) u_7(x_8), \\
v_7(x_8) &= \sum_{x_5, x_6} f_3(x_5, x_8) f_4(x_6, x_8) u_4(x_5, x_6), \\
v_1(x_4, x_5) &= \sum_{x_6} v_4(x_5, x_6) u_2(x_4, x_6), \\
v_2(x_4, x_6) &= \sum_{x_5} v_4(x_5, x_6) u_1(x_4, x_5).
\end{aligned}$$

From the functions u_i and v_i we find the clique marginals

$$\begin{aligned}
g_{\gamma_1}(x_1, x_4, x_5) &= f_1(x_1, x_4, x_5) v_1(x_4, x_5), \\
g_{\gamma_2}(x_2, x_3, x_4, x_6) &= f_2(x_2, x_3, x_4, x_6) v_2(x_4, x_6), \\
g_{\gamma_4}(x_4, x_5, x_6) &= u_1(x_4, x_5) u_2(x_4, x_6) v_4(x_5, x_6), \\
g_{\gamma_7}(x_7, x_8) &= f_5(x_7, x_8) v_7(x_8), \\
g_{\gamma_5}(x_5, x_6, x_8) &= f_3(x_5, x_8) f_4(x_6, x_8) u_4(x_5, x_6) u_7(x_8).
\end{aligned}$$

Algorithm 7.3 is an example of a *message-passing* algorithm; the functions (tables) $u_i(x_{\alpha_i})$ and $v_i(x_{\alpha_i})$ are the 'messages' passed between adjacent cliques in the clique tree. For a deeper discussion of

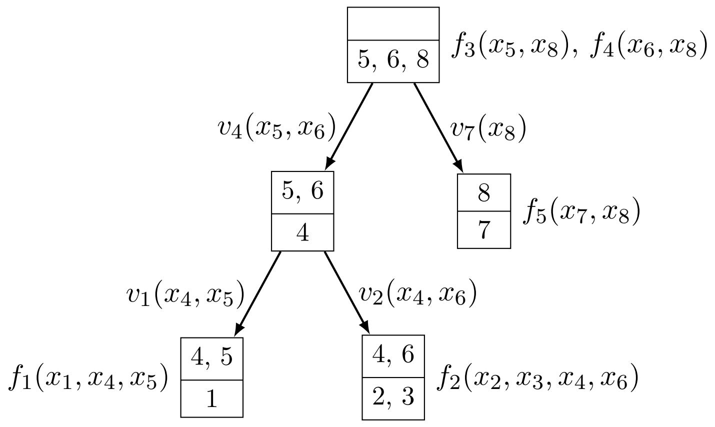


Figure 7.8: Clique tree and intermediate variables for the top-down recursion in Algorithm 7.3.

the theory, applications, and implementation of message passing algorithms in probabilistic networks, and historical notes on their origins, we refer the interested reader to the reference texts [162, chapter 26], [58, section 6.3], [64, chapter 7], [133, chapter 10].

7.3 Generalized marginalization

We have discussed variable elimination methods for calculating the minimum of the sum and the sum of the product of functions $f_k(x_{\beta_k})$. The algorithms for the two problems cases are essentially the same. The techniques for the min-sum and max-product problems can be extended to other pairs of operations [201, 147, 2], [58, section 6.4]. Aji and McEliece in [2] refer to the general problem as *marginalizing a product function* in a commutative semiring and give several examples from communications, information theory, and signal processing. They describe a general message-passing algorithm, the *generalized distributive law*, and show that it includes as special cases a large family of fast algorithms, with a history going back to decoding algorithms developed in the 1960s. The connections between the machine learning and coding theory applications are further discussed in [162, 164].

8

Sparse Matrices

The second part of the paper (Chapters 8–11) covers applications of chordal graphs to sparse symmetric matrices. Graph theory has been an important tool in sparse matrix analysis since the 1960s [183] and much of the development of chordal graph theory was motivated by its relevance for sparse Cholesky factorization [193]. In this section we set out the notation and definitions that will be used in our discussion of sparse symmetric matrices.

8.1 Symmetric sparsity pattern

We define a symmetric *sparsity pattern* of order n as a set

$$E \subseteq \{\{i, j\} \mid i, j \in \{1, 2, \ldots, n\}\}.$$

A symmetric matrix A is said to have sparsity pattern E if $A_{ij} = A_{ji} = 0$ whenever $i \neq j$ and $\{i, j\} \notin E$. The diagonal entries A_{ii} and the off-diagonal entries A_{ij} with $\{i, j\} \in E$ may or may not be zero. The graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ is called the *sparsity graph* associated with the sparsity pattern. Figure 8.1 shows a sparsity

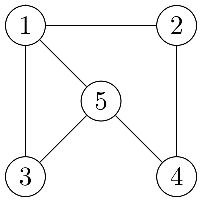


Figure 8.1: Sparsity graph for the matrix (8.1).

graph for the matrix

$$A = \begin{bmatrix} A_{11} & A_{21} & A_{31} & 0 & A_{51} \\ A_{21} & A_{22} & 0 & A_{42} & 0 \\ A_{31} & 0 & A_{33} & 0 & A_{53} \\ 0 & A_{42} & 0 & A_{44} & A_{54} \\ A_{51} & 0 & A_{53} & A_{54} & A_{55} \end{bmatrix}. \tag{8.1}$$

Note that the sparsity pattern and sparsity graph of a matrix are not unique (except for a matrix with no zero off-diagonal entries). If A has sparsity pattern E and $E \subset E'$, then A also has sparsity pattern E'. The sparsity pattern E' is called an *extension* or *embedding* of the sparsity pattern E.

The notation $\mathbf{S}^n$ will be used for the set of symmetric matrices of order n and $\mathbf{S}^n_E$ for the set of symmetric matrices of order n with sparsity pattern E. When we refer to 'nonzeros' or 'dense submatrices' of a sparsity pattern, it should be understood that this refers to the entire set of matrices $\mathbf{S}^n_E$, *i.e.*, it means that there exist matrices in $\mathbf{S}^n_E$ with nonzero entries or dense submatrices in these positions. In particular, if $W \subseteq \{1, 2, \ldots, n\}$ induces a complete subgraph of G, then the principal submatrix with rows and columns indexed by W is referred to as a dense principal submatrix of the sparsity pattern. The clique $W = \{1, 3, 5\}$ in Figure 8.1, for example, defines a 'dense' submatrix

$$\begin{bmatrix} A_{11} & A_{31} & A_{51} \\ A_{31} & A_{33} & A_{53} \\ A_{51} & A_{53} & A_{55} \end{bmatrix},$$

because there exist matrices in $\mathbf{S}^n_E$ with nonzero entries in these nine positions.

We use the term *index set* for an ordered sequence of distinct integers from $\{1, 2, \ldots, n\}$. If $\beta = (\beta(1), \ldots, \beta(r))$ is an index set of length $r \leq n$, then P_β denotes the $r \times n$ matrix with entries

$$(P_\beta)_{ij} = \begin{cases} 1 & j = \beta(i) \\ 0 & \text{otherwise.} \end{cases}$$

If $r = n$ this is a permutation matrix. Multiplying an n-vector x with P_β forms an r-vector with the entries indexed by β, *i.e.*, a permuted subvector of x:

$$P_\beta x = x_\beta = (x_{\beta(1)}, \ldots, x_{\beta(r)}).$$

Multiplying an $n \times n$ matrix with P_β on the left and with P_β^T on the right extracts the $r \times r$ principal submatrix with rows and columns indexed by β, and applies a symmetric reordering to it:

$$P_\beta X P_\beta^T = X_{\beta\beta} = \begin{bmatrix} X_{\beta(1)\beta(1)} & X_{\beta(1)\beta(2)} & \cdots & X_{\beta(1)\beta(r)} \\ X_{\beta(2)\beta(1)} & X_{\beta(2)\beta(2)} & \cdots & X_{\beta(2)\beta(r)} \\ \vdots & \vdots & & \vdots \\ X_{\beta(r)\beta(1)} & X_{\beta(r)\beta(2)} & \cdots & X_{\beta(r)\beta(r)} \end{bmatrix}.$$

The product of an r-vector y with P_β^T is an n-vector x with $x_\beta = y$ and $x_i = 0$ in the positions outside β: for $i = 1, \ldots, r$,

$$(P_\beta^T y)_i = \begin{cases} y_j & i = \beta(j) \\ 0 & i \notin \{\beta(1), \ldots, \beta(r)\}. \end{cases}$$

Multiplying an $r \times r$ matrix Y with P_β^T on the left and P_β on the right forms an $n \times n$ matrix X with $X_{\beta\beta} = Y$ and other entries zero: for $k, l = 1, \ldots, n$,

$$(P_\beta^T Y P_\beta)_{kl} = \begin{cases} Y_{ij} & (k, l) = (\beta(i), \beta(j)) \\ 0 & (k, l) \notin \{\beta(1), \ldots, \beta(r)\} \times \{\beta(1), \ldots, \beta(r)\}. \end{cases}$$

8.2 Chordal sparsity pattern

A sparsity pattern E is said to be *chordal* if the corresponding sparsity graph is chordal. In our discussion of matrices with chordal sparsity patterns we will often make the following assumptions. They can be made without loss of generality and simplify the analysis and interpretation of sparse matrix algorithms considerably.

First, we will assume that the sparsity graph $G = (V, E)$ is connected. If not, one can apply a reordering of the rows and columns to convert the matrices in $\mathbf{S}^n_E$ to block-diagonal form. The sparsity patterns of the diagonal blocks are the connected components of G and can be analyzed separately.

Second, again by applying a suitable reordering, we can assume that the numerical ordering $\sigma = (1, 2, \ldots, n)$ is a perfect elimination ordering for G. Moreover, for any given supernodal partition $\{\mathrm{snd}(i) \mid i \in V^{\mathrm{s}}\}$ with $V^{\mathrm{s}} \subseteq \{1, 2, \ldots, n\}$, we can select a perfect elimination ordering that satisfies the two additional properties in §4.6: the vertices in each supernode are numbered consecutively ($\mathrm{snd}(i) = \{i, i{+}1, \ldots, i{+}n_i\}$ if $\mathrm{snd}(i)$ has $n_i + 1$ elements), and the numerical ordering is a topological ordering of the supernodal elimination tree ($i < q(i)$ if supernode $\mathrm{snd}(q(i))$ is the parent of supernode $\mathrm{snd}(i)$ in the supernodal elimination tree T^{s}).

For convenience, we will use the term *postordered supernodal elimination tree* to describe all of these assumptions. Given a postordered supernodal elimination tree, we can define the following index sets.

- For $i = 1, \ldots, n$, the index set γ_i contains the elements of $\mathrm{col}(i)$, sorted using the numerical ordering σ. The index set γ_i contains the row indices of the lower-triangular nonzeros in column i.

- For $i = 1, \ldots, n$, the index set η_i contains the elements of $\{j \mid j \geq i\} \setminus \mathrm{col}(j)$, sorted using the numerical ordering σ. The index set η_i contains the row indices of the lower-triangular zeros in column i.

- For $i \in V^{\mathrm{s}}$, the index set ν_i contains the elements of $\mathrm{snd}(i)$, sorted using the numerical ordering: $\nu_i = (i, i+1, \ldots, i+n_i)$ if $n_i = |\mathrm{snd}(i)| - 1$.

- For $i \in V^{\mathrm{s}}$, the index set α_i contains the elements of $\mathrm{col}(i) \setminus \mathrm{snd}(i)$, sorted using the numerical ordering σ. The index set α_i is empty if i is the root of the supernodal elimination tree.

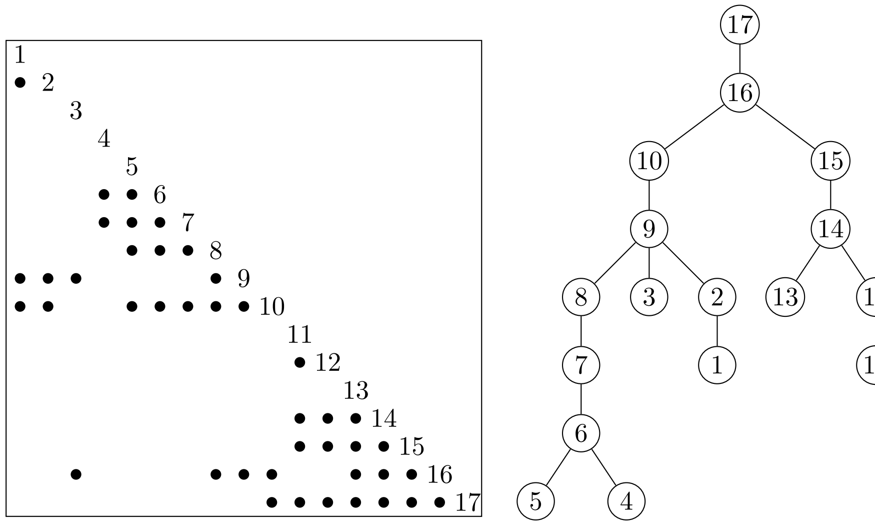


Figure 8.2: A chordal sparsity pattern of order 17 with perfect elimination ordering $(1, 2, \ldots, 17)$ and the corresponding elimination tree.

Hence, for a representative vertex $i \in V^{\mathrm{s}}$, the leading part of γ_i is the index set ν_i and the remaining part is α_i. By Theorem 4.3, the indices in α_i are included in the index set $\gamma_{q(i)}$, where $q(i) > i$ is the parent of the supernode represented by i in the supernodal elimination tree T^{s}. The first element of α_i is the first ancestor $a(i)$ of i.

This notation can be used for any of the supernodal partitions described in §4.5. When applied to a standard elimination tree, any topological ordering of the elimination tree satisfies these assumptions. In this case, the representative vertex set is $V^{\mathrm{s}} = V = \{1, 2, \ldots, n\}$, every supernode ν_i consists of a single element i, and α_i contains the sorted entries of $\mathrm{adj}^+(i)$.

An example is shown in Figure 8.2. In this example,

$$\gamma_5 = (5, 6, 7, 8, 10), \qquad \nu_5 = 5, \qquad \alpha_5 = (6, 7, 8, 10),$$

and

$$\eta_5 = (9, 11, 12, 13, 14, 15, 16, 17).$$

For the same sparsity pattern, Figure 8.3 shows a maximal supernode partition with an ordering that satisfies our assumptions. The elements

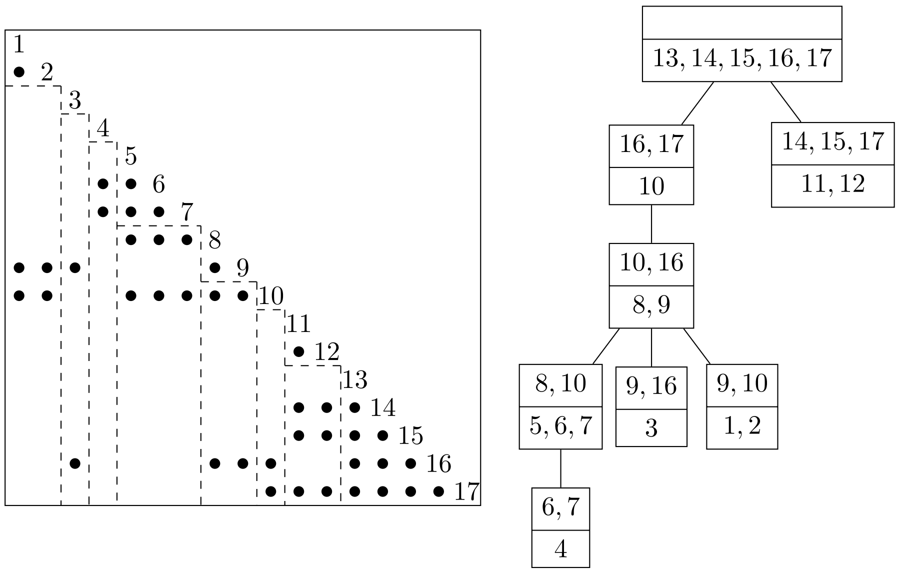


Figure 8.3: *Left.* The sparsity pattern of Figure 8.2. The dashed lines separate the supernodes in a maximal supernode partition. *Right.* The corresponding clique tree.

of the supernodes are numbered consecutively, and the ordering is a topological ordering of the representative vertices. In this case,

$$\gamma_5 = (5, 6, 7, 8, 10), \qquad \nu_5 = (5, 6, 7), \qquad \alpha_5 = (8, 10),$$

and

$$\eta_5 = (9, 11, 12, 13, 14, 15, 16, 17).$$

The parent of ν_5 in the supernodal elimination tree is the supernode $\nu_8 = (8, 9)$.

When deriving or interpreting algorithms for chordal sparse matrices, it is often useful to consider a few common basic patterns. The simplest non-complete chordal pattern has two cliques. Matrices with this pattern consist of two overlapping diagonal blocks, as shown in Figure 8.4. Two other simple chordal sparsity patterns are band patterns (Figure 8.5) and arrow patterns (Figures 8.6 and 8.7). The sparsity graphs for band and arrow patterns are k-trees with $k = w$ (see §3.2).

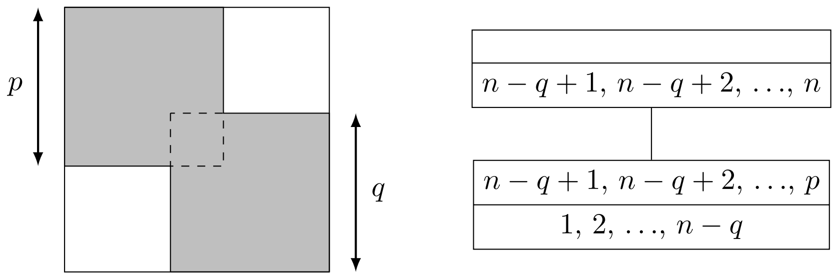

p
q
$n-q+1,\, n-q+2,\, \ldots,\, n$
$n-q+1,\, n-q+2,\, \ldots,\, p$
$1,\, 2,\, \ldots,\, n-q$

Figure 8.4: Chordal sparsity pattern with two overlapping diagonal blocks of order p and q and associated clique tree.

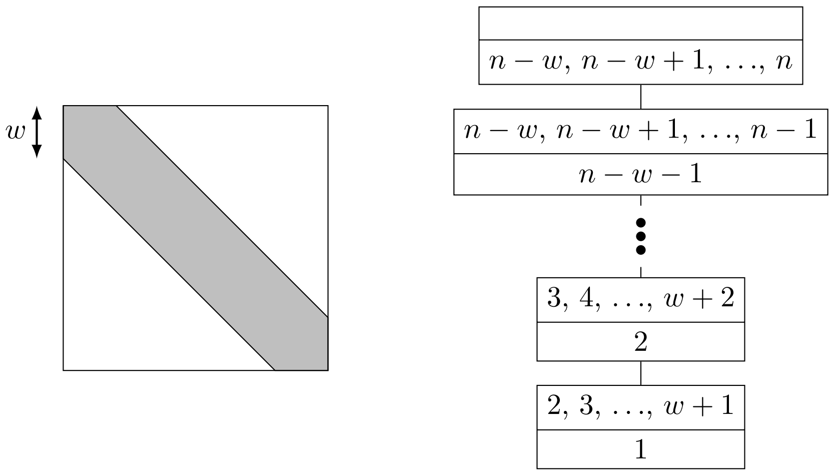

w
$n-w,\, n-w+1,\, \ldots,\, n$
$n-w,\, n-w+1,\, \ldots,\, n-1$
$n-w-1$
$3,\, 4,\, \ldots,\, w+2$
2
$2,\, 3,\, \ldots,\, w+1$
1

Figure 8.5: Band pattern with bandwidth $2w+1$ and a clique tree.

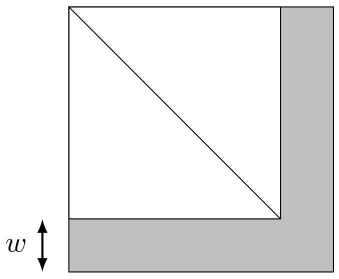


Figure 8.6: Block arrow pattern with block width w.

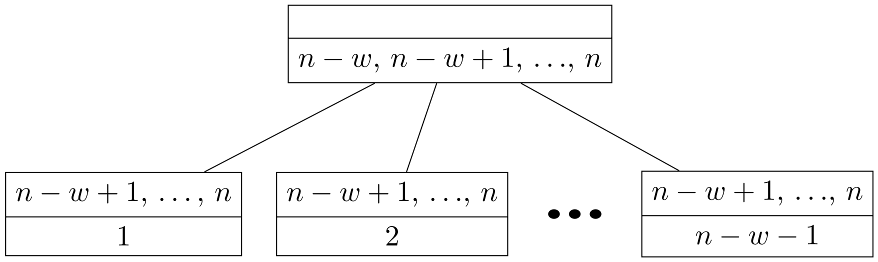


Figure 8.7: Clique tree for the block arrow pattern in Figure 8.6.

8.3 Chordal extension

If E' is chordal and $E \subset E'$ then we say E' is a *chordal extension* or *chordal embedding* of the sparsity pattern E. Chordal extensions of non-chordal sparsity patterns can be constructed by choosing an ordering σ and computing the elimination graph G^*_σ of $G_\sigma = (V, E, \sigma)$. A common ordering choice is a fill-reducing heuristic such as the minimum degree or approximate minimum degree ordering, but depending on the application other choices may be appropriate (see §6.6).

9

Positive Semidefinite Matrices

In this chapter we discuss positive semidefinite matrices with chordal sparsity patterns.

9.1 Cholesky factorization

We define the Cholesky factorization of a symmetric positive definite matrix A as a decomposition

$$P_\sigma A P_\sigma^T = LDL^T \tag{9.1}$$

with L unit lower-triangular, D positive diagonal, and $\sigma = (\sigma(1), \ldots, \sigma(n))$ a permutation of $(1, 2, \ldots, n)$. Equivalently, A is factored as $A = P_\sigma^T LDL^T P_\sigma$. It is a basic result in linear algebra that if A is positive definite, then for every permutation σ there exist unique L and D that satisfy (9.1). If A is positive semidefinite but singular, a decomposition (9.1) with nonnegative diagonal entries in D exists for every permutation σ, but it is not unique (and difficult to compute in finite precision); see [117]. To make the semidefinite factorization unique, we will define the entries of L below the jth diagonal entry to be zero if D_{jj} is zero.

The graph elimination process of Chapter 6 describes the relation between the sparsity patterns of A and $L + L^T$. We will show that if $A \in \mathbf{S}^n_E$ then

$$P_\sigma^T(L + L^T)P_\sigma \in \mathbf{S}^n_{E'} \tag{9.2}$$

where $E' = E^*_\sigma$ is the edge set of the filled ordered sparsity graph $G_\sigma = (V, E, \sigma)$. To simplify the notation we will assume that σ is the numerical ordering $(1, 2, \ldots, n)$ and derive the sparsity pattern of $L + L^T$ in the factorization $A = LDL^T$. In the 'outer product' formulation of the Cholesky factorization algorithm [100, section 4.2.5], one starts from the partitioned matrix

$$A = \begin{bmatrix} d_1 & b_1^T \\ b_1 & C_1 \end{bmatrix}$$

and factors it as $A = L_1 D_1 L_1^T$ where

$$L_1 = \begin{bmatrix} 1 & 0 \\ (1/d_1)b_1 & I \end{bmatrix}, \qquad D_1 = \begin{bmatrix} d_1 & 0 \\ 0 & C_1 - (1/d_1)b_1 b_1^T \end{bmatrix}.$$

(If singular positive semidefinite matrices A are allowed, the leading element d_1 may be zero, but then b_1 must be zero, and one replaces $1/d_1$ by zero in the definition of L_1 and D_1.) After this first step, $L_1 + D_1 + L_1^T \in \mathbf{S}^n_{E_1}$ where

$$E_1 = E \cup \{\{j, k\} \mid 1 < j < k,\ \{1, j\} \in E,\ \{1, k\} \in E\}.$$

This is the edge set after one step of the graph elimination process (defined in (6.1)). The added edges in E_1 account for possible new nonzero entries introduced by the outer product term $b_1 b_1^T$. The matrix $C_1 - (1/d_1)b_1 b_1^T$ is positive (semi-)definite if A is, so D_1 can be further factored as $D_1 = \tilde{L}_2 D_2 \tilde{L}_2^T$ with

$$\tilde{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & (1/d_2)b_2 & I \end{bmatrix}, \qquad D_2 = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & C_2 - (1/d_2)b_2 b_2^T \end{bmatrix}.$$

(Again, we substitute zero for $1/d_2$ if $d_2 = 0$.) Here d_2, b_2, and C_2 refer to the partitioned matrix

$$C_1 - \frac{1}{d_1} b_1 b_1^T = \begin{bmatrix} d_2 & b_2^T \\ b_2 & C_2 \end{bmatrix}.$$

We now have a factorization $A = L_2 D_2 L_2^T$ where $L_2 = L_1 \tilde{L}_2$ and $L_2 + D_2 + L_2^T \in \mathbf{S}^n_{E_2}$ where

$$E_2 = E_1 \cup \{\{j, k\} \mid 2 < j < k,\ \{2, j\} \in E_1,\ \{2, k\} \in E_1\}.$$

In the next step we factor the $(n-2) \times (n-2)$ matrix $C_2 - (1/d_2) b_2 b_2^T$, et cetera. After i steps of this process, we have a factorization $A = L_i D_i L_i^T$ with $L_i + D_i + L_i^T \in \mathbf{S}^n_{E_i}$, where

$$E_i = E_{i-1} \cup \{\{j, k\} \mid i < j < k,\ \{i, j\} \in E_{i-1},\ \{i, k\} \in E_{i-1}\}.$$

After $n-1$ steps, we obtain the complete factorization with sparsity pattern $E_{n-1} = E^*_\sigma$.

If E is a chordal pattern and σ is a perfect elimination ordering for it, then $E^*_\sigma = E$ and we obtain a 'zero-fill' Cholesky factorization, since (9.2) gives $L + L^T \in \mathbf{S}^n_E$. This explains the term 'perfect elimination ordering'. The connection between chordal graphs and matrix elimination was made by Rose [193].

Theorem 9.1. A sparsity pattern E is chordal if and only if every positive definite matrix in $\mathbf{S}^n_E$ has a Cholesky factorization with

$$P_\sigma^T (L + L^T) P_\sigma \in \mathbf{S}^n_E. \tag{9.3}$$

Proof. If E is chordal, then there exists a perfect elimination ordering σ (Theorem 4.1). We have $E^*_\sigma = E$ and (9.3) follows from (9.2). If E is not chordal, then for every ordering σ one can find positive definite $A \in \mathbf{S}^n_E$ for which the Cholesky factors do not satisfy (9.3), for example, using the following construction. Let σ be an arbitrary ordering of $V = \{1, 2, \ldots, n\}$. Since E is not chordal, the sparsity graph has a chordless cycle of length greater than three. Let $\sigma(i) \in V$ be the vertex in the cycle with the lowest index i, and let $\sigma(j)$ and $\sigma(k)$ with $k > j$ be its two neighbors in the chordless cycle. Define $A \in \mathbf{S}^n_E$ as the matrix with unit diagonal,

$$\begin{bmatrix} A_{\sigma(i)\sigma(i)} & A_{\sigma(i)\sigma(j)} & A_{\sigma(i)\sigma(k)} \\ A_{\sigma(j)\sigma(i)} & A_{\sigma(j)\sigma(j)} & A_{\sigma(j)\sigma(k)} \\ A_{\sigma(k)\sigma(i)} & A_{\sigma(k)\sigma(j)} & A_{\sigma(k)\sigma(k)} \end{bmatrix} = \begin{bmatrix} 1 & 1/2 & 1/2 \\ 1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{bmatrix},$$

and all other off-diagonal elements equal to zero. It can be verified that A is positive definite, with a factorization (9.3) that has entries

$$\begin{bmatrix} L_{ii} & 0 & 0 \\ L_{ji} & L_{jj} & 0 \\ L_{ki} & L_{kj} & L_{kk} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/2 & -1/3 & 1 \end{bmatrix},$$

$$\begin{bmatrix} D_{ii} & 0 & 0 \\ 0 & D_{jj} & 0 \\ 0 & 0 & D_{kk} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/4 & 0 \\ 0 & 0 & 2/3 \end{bmatrix}.$$

The other strictly lower-triangular entries of L are zero and the other diagonal entries of D are one. Since $L_{kj} \neq 0$, the matrix $P_\sigma^T(L+L^T)P_\sigma$ is not in $\mathbf{S}_E^n$. □

9.2 Positive semidefinite matrix cone

The positive semidefinite matrices with sparsity pattern E form a convex cone

$$\mathbf{S}_+^n \cap \mathbf{S}_E^n = \{A \in \mathbf{S}_E^n \mid A \succeq 0\}.$$

This is a proper cone in $\mathbf{S}_E^n$, *i.e.*, closed, pointed, and with nonempty interior. It is clearly a closed convex cone, since it is the intersection of a subspace and a closed convex cone. It is pointed because A and $-A$ are positive semidefinite only if $A = 0$. Its interior is $\mathbf{S}_{++}^n \cap \mathbf{S}_E^n$, where $\mathbf{S}_{++}^n = \{A \in \mathbf{S}^n \mid A \succ 0\}$ is the set of positive definite matrices.

When the sparsity pattern E is chordal, the cone $\mathbf{S}_+^n \cap \mathbf{S}_E^n$ can be decomposed as a sum of simple convex cones, as stated in the following theorem [106, theorem 4] [1, theorem 2.3] [122, theorem 1].

Theorem 9.2. Let E be a chordal sparsity pattern of order n. Then $A \in \mathbf{S}_+^n \cap \mathbf{S}_E^n$ if and only if it can be expressed as

$$A = \sum_{\beta \in C} P_\beta^T H_\beta P_\beta \tag{9.4}$$

where C is the set of cliques of the sparsity graph and the matrices H_β are symmetric and positive semidefinite.

Proof. Clearly, any decomposition (9.4) with $H_\beta \succeq 0$ implies that A is positive semidefinite and has sparsity pattern E. The other direction

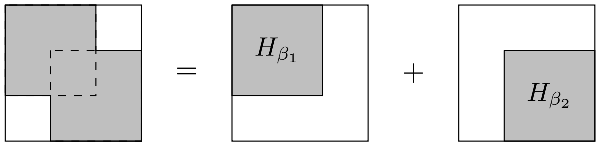


Figure 9.1: A chordal sparsity pattern with two overlapping dense principal submatrices. A matrix with this pattern is positive semidefinite if and only if it can be written as a sum of two positive semidefinite matrices, each with only one dense principal submatrix $H_{\beta_i} \succeq 0$.

(existence of a decomposition (9.4) if E is a chordal pattern) follows from the zero-fill property of the Cholesky factorization. Without loss of generality we assume that the numerical ordering $\sigma = (1, 2, \ldots, n)$ is a perfect elimination ordering for E. Let $V^{\mathrm{c}} \subseteq \{1, 2, \ldots, n\}$ be the set of representative vertices of the cliques in the sparsity graph, $\{\mathrm{snd}(i) \mid i \in V^{\mathrm{c}}\}$ a maximal supernode partition, and γ_j, $j = 1, \ldots, n$, the index sets with the elements of $\mathrm{col}(j)$.

If $A \in \mathbf{S}^n_+ \cap \mathbf{S}^n_E$, then it has a Cholesky factorization $A = LDL^T$ and the matrix product can be expressed as a sum of outer products

$$A = \sum_{j=1}^{n} D_{jj} P_{\gamma_j}^T L_{\gamma_j j} L_{\gamma_j j}^T P_{\gamma_j} = \sum_{i \in V^{\mathrm{c}}} \sum_{j \in \mathrm{snd}(i)} D_{jj} P_{\gamma_j}^T L_{\gamma_j j} L_{\gamma_j j}^T P_{\gamma_j}. \qquad (9.5)$$

We observe that if $j \in \mathrm{snd}(i)$, then the elements of γ_j are included in γ_i and therefore $P_{\gamma_j}^T L_{\gamma_j j} = P_{\gamma_i}^T P_{\gamma_i} P_{\gamma_j}^T L_{\gamma_j j}$. This allows us to write the sum in (9.5) as $A = \sum_{i \in V^{\mathrm{c}}} P_{\gamma_i}^T B_i P_{\gamma_i}$ where

$$B_i = P_{\gamma_i} \left(\sum_{j \in \mathrm{snd}(i)} P_{\gamma_j}^T L_{\gamma_j j} L_{\gamma_j j}^T P_{\gamma_j} \right) P_{\gamma_i}^T \succeq 0.$$

Since $C = \{\gamma_i \mid i \in V^{\mathrm{c}}\}$ is the set of cliques of the sparsity graph, we have decomposed A as in (9.4). □

Figures 9.1 and 9.2 illustrate the theorem and its proof with a simple chordal pattern consisting of two overlapping diagonal blocks.

The result of Theorem 9.2 can be written concisely as a decomposition of the cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ as a sum of cones:

$$\mathbf{S}^n_+ \cap \mathbf{S}^n_E = \sum_{\beta \in C} \mathcal{K}_\beta, \qquad \mathcal{K}_\beta = \{P_\beta^T H_\beta P_\beta \mid H_\beta \succeq 0\}. \qquad (9.6)$$

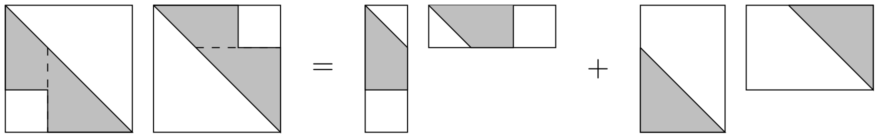


Figure 9.2: Construction of the decomposition in Figure 9.1 from a zero-fill Cholesky factorization.

Each cone $\mathcal{K}_\beta$ has a very simple structure. Its elements consist of one positive semidefinite principal submatrix, in the rows and columns indexed by β, and are zero outside this block. The cones $\mathcal{K}_\beta$ are closed, convex, and pointed, but not proper because they have empty interiors (unless the sparsity graph is complete and there is only one clique).

Theorem 9.2 implies that the sum $\sum_\beta \mathcal{K}_\beta$ is closed, since $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ is closed. This can also be seen from [192, corollary 9.1.3] where the following sufficient condition is given for the sum of closed convex cones to be closed: if $\mathcal{C}_i$, $i = 1, \ldots, l$, are closed convex cones that satisfy the condition

$$\sum_{i=1}^{l} x_i = 0, \quad x_i \in \mathcal{C}_i, \;\; i = 1, \ldots, l \qquad \Longrightarrow \qquad x_i = 0, \;\; i = 1, \ldots, l,$$

then $\sum_{i=1}^{l} \mathcal{C}_i$ is closed. This sufficient condition holds for the cones $\mathcal{K}_\beta$ defined in (9.6), because $\sum_\beta P_\beta^T H_\beta P_\beta = 0$ with $H_\beta \succeq 0$ implies that the diagonal of each H_β is zero, hence $H_\beta = 0$.

If A is a positive semidefinite matrix with a non-chordal sparsity pattern, then a decomposition (9.4) may still exist. However, one can show that for every non-chordal pattern there exist positive semidefinite matrices that cannot be decomposed as in (9.4). Instances can be constructed by the following method from [106, page 41]. Suppose the sparsity graph (V, E) has a chordless cycle of length greater than three. After a suitable reordering we can assume that the cycle is $(1, 2, \ldots, k, 1)$ with $k > 3$. Define A as the matrix with

$$A_{ii} = \lambda, \quad i = 1, \ldots, k, \qquad A_{i,i+1} = A_{i+1,i} = 1, \quad i = 1, \ldots, k-1,$$

$A_{k1} = A_{1k} = (-1)^{k-1}$, and other entries zero. Hence, A has a leading

$k \times k$ diagonal block

$$\tilde{A} = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 & 0 & (-1)^{k-1} \\ 1 & \lambda & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \lambda & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & \lambda & 1 \\ (-1)^{k-1} & 0 & 0 & \cdots & 0 & 1 & \lambda \end{bmatrix}$$

and is otherwise zero. For odd k, $\tilde{A}$ is a circulant Toeplitz matrix with eigenvalues $\lambda + 2\cos(2i\pi/k)$, $i = 0, \ldots, k-1$. For even k, $\tilde{A}$ is an anticirculant Toeplitz matrix with eigenvalues $\lambda + 2\cos((2i+1)\pi/k)$, $i = 0, \ldots, k-1$. In both cases, $\tilde{A}$ is positive semidefinite for $\lambda \geq 2\cos(\pi/k)$.

Each clique of the sparsity pattern contains at most one edge of the cycle, since a second edge would create a chord in the cycle. Since A is zero outside of the block $\tilde{A}$, a decomposition (9.4) exists if and only $\tilde{A}$ can be decomposed as $\tilde{A} = \sum_i P_{\beta_i}^T H_i P_{\beta_i}$ with $\beta_i = (i, i+1)$ for $i = 1, \ldots, k-1$, $\beta_k = (1, k)$, and positive semidefinite 2×2-matrices H_i. This is equivalent to existence of scalars $u_1, \ldots, u_k$ such that

$$H_i = \begin{bmatrix} u_i & 1 \\ 1 & \lambda - u_{i+1} \end{bmatrix} \succeq 0, \quad i = 1, \ldots, k-1,$$

and

$$H_k = \begin{bmatrix} \lambda - u_1 & (-1)^{k-1} \\ (-1)^{k-1} & u_k \end{bmatrix} \succeq 0.$$

The conditions on the coefficients u_i are that $0 \leq u_i \leq \lambda$ for $i = 1, \ldots, k$ and

$$1 \leq u_i(\lambda - u_{i+1}), \quad i = 1, \ldots, k-1, \qquad 1 \leq u_k(\lambda - u_1).$$

We now show that these inequalities have no solution if $\lambda < 2$. Combining the quadratic inequalities and using the arithmetic-geometric mean inequality, we obtain that the coefficients u_i must satisfy

$$1 \leq \left(\prod_{i=1}^{k} u_i(\lambda - u_i) \right)^{\frac{1}{2k}} \leq \frac{\lambda}{2}.$$

Therefore a decomposition (9.4) does not exist if $\lambda < 2$. However the matrix A is positive semidefinite if $\lambda \geq 2\cos(\pi/k)$ and this lower bound is less than 2. We conclude that for λ in the interval $[2\cos(\pi/k), 2)$ the matrix A is positive semidefinite but does not have a decomposition (9.4).

9.3 Multifrontal factorization

Elimination trees and the theory from Chapter 4 play a key role in the the multifrontal Cholesky factorization algorithm [75, 153, 154].

We use the notation and conventions of §8.2. In particular, $\sigma = (1, 2, \ldots, n)$ is the numerical ordering ($P_\sigma = I$) and we examine the Cholesky factorization $A = LDL^T$ with diagonal D. We first examine how the product in $A = LDL^T$ determines A. Assume D is a nonnegative diagonal matrix, L is unit lower-triangular with $L + L^T \in \mathbf{S}^n_E$, and E is a chordal sparsity pattern with perfect elimination ordering σ. The i, j entry of A, for $i \geq j$, is

$$A_{ij} = \sum_{k<j} D_{kk} L_{ik} L_{jk} + D_{jj} L_{ij}. \tag{9.7}$$

This shows that $A_{ij} = 0$ when $\{i, j\} \notin E$ (by monotone transitivity, $i > j > k$ and $\{i, k\} \in E$, $\{j, k\} \in E$ imply $\{i, j\} \in E$). Hence, A has sparsity pattern E and it is sufficient to examine A_{ij} for $i = j$ or $\{i, j\} \in E$. We denote the elimination tree for E by T and use the index set notation of §8.2: α_j, γ_j are the index sets containing the sorted elements of $\mathrm{adj}^+(j)$ and $\mathrm{col}(j)$, respectively. From (9.7) the subvector $A_{\gamma_j j}$ can be expressed as

$$\begin{aligned}
\begin{bmatrix} A_{jj} \\ A_{\alpha_j j} \end{bmatrix} &= \sum_{k<j} D_{kk} L_{jk} \begin{bmatrix} L_{jk} \\ L_{\alpha_j k} \end{bmatrix} + D_{jj} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix} \\
&= \sum_{k \in T_j \setminus \{j\}} D_{kk} L_{jk} \begin{bmatrix} L_{jk} \\ L_{\alpha_j k} \end{bmatrix} + D_{jj} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix} \\
&= \sum_{i \in \mathrm{ch}(j)} \sum_{k \in T_i} D_{kk} L_{jk} \begin{bmatrix} L_{jk} \\ L_{\alpha_j k} \end{bmatrix} + D_{jj} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix},
\end{aligned} \tag{9.8}$$

where T_j denotes the subtree of the elimination tree formed by the descendants of vertex j, and $\mathrm{ch}(j)$ is the set of children of j in the elimination tree. The second line follows because if $k < j$ then $\{j,k\} \in E$ only if k is a descendant of j in the elimination tree. (More precisely, k must be in the jth row subtree.) Now suppose that with each vertex in the elimination tree we associate an $|\alpha_j| \times |\alpha_j|$ matrix

$$U_j = \sum_{k \in T_j} D_{kk} L_{\alpha_j k} L_{\alpha_j k}^T.$$

We call this the jth *update matrix.* The update matrices satisfy a recursive equation

$$\begin{aligned} U_j &= D_{jj} L_{\alpha_j j} L_{\alpha_j j}^T + \sum_{i \in \mathrm{ch}(j)} \sum_{k \in T_i} D_{kk} L_{\alpha_j k} L_{\alpha_j k}^T \\ &= D_{jj} L_{\alpha_j j} L_{\alpha_j j}^T + P_{\alpha_j} \left(\sum_{i \in \mathrm{ch}(j)} P_{\alpha_i}^T \left(\sum_{k \in T_i} D_{kk} L_{\alpha_i k} L_{\alpha_i k}^T \right) P_{\alpha_i} \right) P_{\alpha_j}^T \\ &= D_j L_{\alpha_j j} L_{\alpha_j j}^T + P_{\alpha_j} \left(\sum_{i \in \mathrm{ch}(j)} P_{\alpha_i}^T U_i P_{\alpha_i} \right) P_{\alpha_j}^T. \end{aligned}$$

On line 2 we use $L_{\alpha_j k} = P_{\alpha_j} P_{\alpha_i}^T L_{\alpha_i k}$ for $k \in T_i$ and $i \in \mathrm{ch}(j)$. (This follows from (4.4).) Combining the last expression with (9.8) we get

$$\begin{aligned} &\begin{bmatrix} A_{jj} & A_{\alpha_j j}^T \\ A_{\alpha_j j} & U_j \end{bmatrix} = \\ &\qquad D_{jj} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix}^T + P_{\gamma_j} \left(\sum_{i \in \mathrm{ch}(j)} P_{\alpha_i}^T U_i P_{\alpha_i} \right) P_{\gamma_j}^T. \end{aligned} \tag{9.9}$$

This formula allows us to compute A, column by column, by enumerating the vertices of the elimination tree in topological order. We start at the leaves of the elimination tree (with $\mathrm{ch}(j) = \emptyset$). At vertex j we compute A_{jj}, $A_{\alpha_j j}$, and the update matrix U_j from D_{jj}, $L_{\alpha_j j}$, and the update matrices of the children of j.

The equation (9.9) also shows us how to compute the Cholesky factors from A, column by column in a topological ordering. This becomes clear if we rearrange (9.9) as

$$\begin{aligned}
&\begin{bmatrix} A_{jj} & A_{\alpha_j j}^T \\ A_{\alpha_j j} & 0 \end{bmatrix} - P_{\gamma_j} \left(\sum_{i \in \mathrm{ch}(j)} P_{\alpha_i}^T U_i P_{\alpha_i} \right) P_{\gamma_j}^T \\
&\quad = D_{jj} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix}^T - \begin{bmatrix} 0 & 0 \\ 0 & U_j \end{bmatrix} \\
&\quad = \begin{bmatrix} 1 & 0 \\ L_{\alpha_j j} & I \end{bmatrix} \begin{bmatrix} D_{jj} & 0 \\ 0 & -U_j \end{bmatrix} \begin{bmatrix} 1 & L_{\alpha_j j}^T \\ 0 & I \end{bmatrix}.
\end{aligned}$$

The left-hand side matrix is called the *frontal matrix* in step j. The jth frontal matrix can be assembled from column j of A and the update matrices of the children of j in the elimination tree. From the frontal matrix, one easily compute D_{jj}, $L_{\alpha_j j}$, and the update matrix U_j. The update matrices of the children of vertex j can be discarded once the frontal matrix for vertex j has been formed. This is summarized in the following algorithm.

Algorithm 9.1 (Multifrontal Cholesky factorization).

Input. A positive definite matrix $A \in \mathbf{S}_E^n$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered elimination tree for G_σ.

Output. The Cholesky factorization $A = LDL^T$.

Algorithm. Enumerate the vertices $j \in V = \{1, 2, \ldots, n\}$ of the elimination tree in topological order. For each j execute the following two steps.

- Form the frontal matrix

$$\begin{bmatrix} F_{11} & F_{21}^T \\ F_{21} & F_{22} \end{bmatrix} = \begin{bmatrix} A_{jj} & A_{\alpha_j j}^T \\ A_{\alpha_j j} & 0 \end{bmatrix} - P_{\gamma_j} \left(\sum_{i \in \mathrm{ch}(j)} P_{\alpha_i}^T U_i P_{\alpha_i} \right) P_{\gamma_j}^T.$$

- Calculate D_{jj}, L_{α_j}, and the update matrix U_j from

$$D_{jj} = F_{11}, \quad L_{\alpha_j j} = \frac{1}{D_{jj}} F_{21}, \quad U_j = -F_{22} + D_{jj} L_{\alpha_j j} L_{\alpha_j j}^T.$$

The algorithm can be generalized to handle singular positive semidefinite matrices. In the extended version, if the leading entry F_{11} of the frontal matrix is zero, we take $D_{jj} = 0$, $L_{\alpha_j j} = 0$, and $U_j = -F_{22}$.

The main calculations in Algorithm 9.1 are the sums in the construction of the frontal matrix and the outer product in the definition of U_j.

9.4 Supernodal factorization

The efficiency of the multifrontal factorization algorithm can be improved by applying block elimination to groups of columns with the same structure. The theory of supernodal partitions, discussed in §4.4 and §4.5, was developed for this purpose (see [154] for a survey and historical notes).

We will use the notation and assumptions of §8.2 for a supernodal elimination tree T^{s} with vertex set V^{s}. We define a blocked or supernodal Cholesky factorization

$$A = LDL^T,$$

where D is block-diagonal with dense symmetric positive definite diagonal blocks $D_{\nu_i\nu_i}$, and L is block unit lower-triangular with $L_{\nu_i\nu_i} = I$ and a nonzero lower-diagonal block $L_{\alpha_i\nu_i}$ in block-column ν_i. This block Cholesky factorization is easily transformed into the standard one (with diagonal D) by taking dense Cholesky factorizations of the diagonal blocks $D_{\nu_i\nu_i}$ and combining their Cholesky factors with L.

The extension of equation (9.9) to the supernodal Cholesky factorization reads

$$\begin{bmatrix} A_{\nu_j\nu_j} & A^T_{\alpha_j\nu_j} \\ A_{\alpha_j\nu_j} & U_j \end{bmatrix} = \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix} D_{\nu_j\nu_j} \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix}^T + P_{\gamma_j}\left(\sum_{i\in\mathrm{ch}(j)} P^T_{\alpha_i} U_i P_{\alpha_i}\right) P^T_{\gamma_j} \qquad (9.10)$$

for $j \in V^{\mathrm{s}}$ where $\mathrm{ch}(j)$ now refers to the children of vertex j in the supernodal elimination tree T^{s}. The update matrices are defined as

$$U_j = \sum_{k\in T^{\mathrm{s}}_j} L_{\alpha_j\nu_k} D_{\nu_k\nu_k} L^T_{\alpha_j\nu_k}$$

where T^{s}_j is the subtree of T^{s} formed by vertex j and its descendants.

The equation (9.10) shows us how to compute the supernodal Cholesky decomposition using a generalization of Algorithm 9.1.

Algorithm 9.2 (Supernodal multifrontal Cholesky factorization).

Input. A positive definite matrix $A \in \mathbf{S}^n_E$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered supernodal elimination tree for G_σ.

Output. The supernodal Cholesky factorization $A = LDL^T$.

Algorithm. Enumerate the supernode representatives $j \in V^{\mathrm{s}}$ in topological order. For each j execute the following two steps.

- Form the frontal matrix

$$\begin{bmatrix} F_{11} & F_{21}^T \\ F_{21} & F_{22} \end{bmatrix} = \begin{bmatrix} A_{\nu_j\nu_j} & A_{\alpha_j\nu_j}^T \\ A_{\alpha_j\nu_j} & 0 \end{bmatrix} - P_{\gamma_j}\left(\sum_{i\in\mathrm{ch}(j)} P_{\alpha_i}^T U_i P_{\alpha_i}\right) P_{\gamma_j}^T. \qquad (9.11)$$

- Calculate the matrices $D_{\nu_j\nu_j}$, $L_{\alpha_j\nu_j}$, from

$$D_{\nu_j\nu_j} = F_{11}, \qquad L_{\alpha_j\nu_j} = F_{21} D_{\nu_j\nu_j}^{-1} \qquad (9.12)$$

  and the update matrix U_j from

$$U_j = -F_{22} + L_{\alpha_j\nu_j} D_{\nu_j\nu_j} L_{\alpha_j\nu_j}^T.$$

This algorithm includes Algorithm 9.1 as a special case, obtained by taking $V^{\mathrm{s}} = V$, $T^{\mathrm{s}} = T$. As for the (nodal) multifrontal Algorithm 9.1, we can generalize Algorithm 9.2 to handle singular positive semidefinite matrices, by allowing singular positive semidefinite blocks $D_{\nu_j\nu_j}$. If the $1,1$-block F_{11} of the frontal matrix is singular, we must have $F_{21} = F_{21}F_{11}^+F_{11}$ if A is positive semidefinite and we can replace the expression for $L_{\alpha_j\nu_j}$ in (9.12) with $L_{\alpha_j\nu_j} = F_{21}D_{\nu_j\nu_j}^+$ where $D_{\nu_j\nu_j}^+$ is the pseudoinverse of $D_{\nu_j\nu_j}$.

The main steps in the algorithm are the construction of the frontal matrices, the factorization of $D_{\nu_j\nu_j}$ needed to compute $L_{\alpha_j\nu_j}$, and the matrix outer-product in the formula for U_j.

The algorithm can also be interpreted as computing a decomposition as described in Theorem 9.2. If we use a maximal supernode

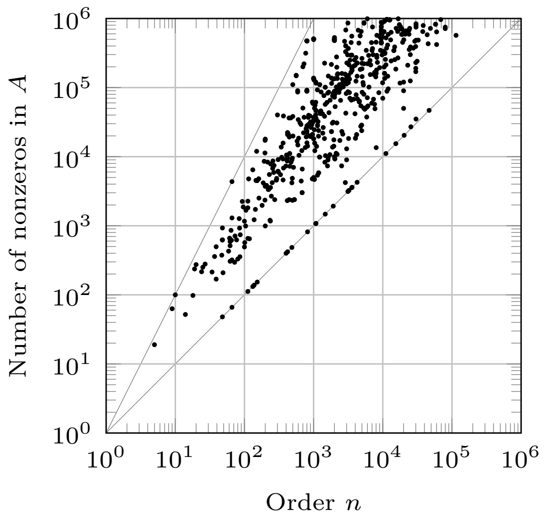


Figure 9.3: Scatter plot of test problem dimensions.

partition ($V^{\mathrm{s}} = V^{\mathrm{c}}$ and γ_j are the cliques) and define

$$H_j = F_j + \begin{bmatrix} 0 & 0 \\ 0 & U_j \end{bmatrix}$$

where F_j is the frontal matrix (9.11), then H_j is positive semidefinite (from (9.10)) and

$$\sum_{j \in V^{\mathrm{c}}} P_{\gamma_j}^T H_j P_{\gamma_j} = \sum_{j \in V^{\mathrm{c}}} P_{\gamma_j}^T \begin{bmatrix} A_{\nu_j \nu_j} & A_{\alpha_j \nu_j}^T \\ A_{\alpha_j \nu_j} & 0 \end{bmatrix} P_{\gamma_j} = A.$$

The first equality follows because in the sum over j the contributions of the update matrices cancel out.

To illustrate the advantage of the supernodal factorization, we include in Figure 9.4 a scatter plot that shows the computation times for the nodal and supernodal multifrontal Cholesky factorizations based on 667 test problems with sparsity patterns from the University of Florida Sparse Matrix Collection [66]. The order of the test matrices range from 5 to approximately 130,000; see Figure 9.3. Notice that the computation times are very similar for small problems, but for many

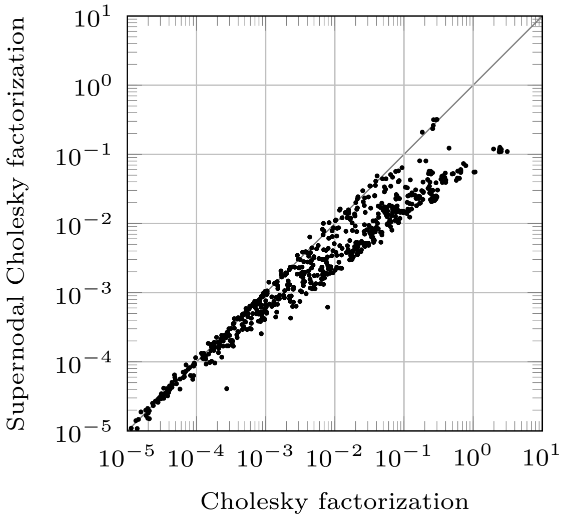


Figure 9.4: Scatter plot of computation times (in seconds) for nodal and supernodal multifrontal Cholesky factorizations.

large problems, the supernodal factorization is up to an order of magnitude faster than the nodal factorization. Both factorizations were computed using Chompack [13].

9.5 Projected inverse

The inverse of a large sparse matrix is usually dense, expensive to compute, and rarely needed in practice. However the problem of computing a *partial inverse*, *i.e.*, a small subset of the entries of the inverse, arises in several applications and can be solved quite efficiently [78, 99, 49, 61, 9, 6]. In this section, we consider the problem of computing $\Pi_E(A^{-1})$ for a positive definite matrix $A \in \mathbf{S}^n_E$, where Π_E denotes projection on the subspace $\mathbf{S}^n_E$: for $Y \in \mathbf{S}^n$,

$$\Pi_E(Y)_{ij} = \begin{cases} Y_{ij} & \{i,j\} \in E \text{ or } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{9.13}$$

We will show that if E is chordal, then $\Pi_E(A^{-1})$ can be efficiently computed from the sparse Cholesky factorization of A [61, 9].

As in the previous section we assume the numerical ordering σ is a perfect elimination ordering for E. To compute the projected inverse $\Pi_E(A^{-1})$ we first factor A as $A = LDL^T$ with L unit lower-triangular and D positive diagonal. The equation

$$A^{-1}L = L^{-T}D^{-1} \tag{9.14}$$

relates A^{-1} and the Cholesky factors L and D. Define $B = \Pi_E(A^{-1})$. For $j = 1, \ldots, n$, the submatrix $B_{\gamma_j\gamma_j}$ of B satisfies

$$\begin{bmatrix} B_{jj} & B_{\alpha_j j}^T \\ B_{\alpha_j j} & B_{\alpha_j\alpha_j} \end{bmatrix} \begin{bmatrix} 1 \\ L_{\alpha_j j} \end{bmatrix} = \begin{bmatrix} 1/D_{jj} \\ 0 \end{bmatrix}. \tag{9.15}$$

This follows from (9.14), the fact that the jth column of L is zero in the positions not included in γ_j, and that L^{-T} is upper triangular with unit diagonal. The equation (9.15) shows that if $B_{\alpha_j\alpha_j}$ is given, then B_{jj} and $B_{\alpha_j j}$ are easily computed as follows:

$$B_{\alpha_j j} = -B_{\alpha_j\alpha_j}L_{\alpha_j j}, \qquad B_{jj} = \frac{1}{D_{jj}} - B_{\alpha_j j}^T L_{\alpha_j j}.$$

Since the indices in α_j are included in the index set γ_i for the parent $i = p(j)$, we can compute the columns of B via a recursion on the elimination tree in inverse topological order, starting at the root $j = n$ (where α_n is empty). The algorithm closely resembles the multifrontal Cholesky factorization, with $U_j = B_{\alpha_j\alpha_j}$ in the role of update matrix, the matrix

$$\begin{bmatrix} B_{jj} & B_{\alpha_j j}^T \\ B_{\alpha_j j} & U_j \end{bmatrix}$$

replacing the frontal matrix, and the direction of the recursion reversed. This formulation is especially useful when B is stored in a column-compressed storage (CCS) scheme [65]. In the CCS scheme it is straightforward to locate the subvector $B_{\alpha_j j}$ with the lower-triangular nonzeros in each column j, but it is not as easy to extract a submatrix $B_{\alpha_j\alpha_j}$. Passing the matrices $B_{\alpha_j\alpha_j}$ as dense update matrices between vertices of the elimination tree and their children avoids the need to retrieve $B_{\alpha_j\alpha_j}$.

As for the multifrontal Cholesky factorization, there is a supernodal version of this algorithm, based on the supernodal Cholesky factorization $A = LDL^T$. In the notation of §9.4 and Algorithm 9.2, the blocked equivalent of equation (9.15) is

$$\begin{bmatrix} B_{\nu_j\nu_j} & B_{\alpha_j\nu_j}^T \\ B_{\alpha_j\nu_j} & B_{\alpha_j\alpha_j} \end{bmatrix} \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix} = \begin{bmatrix} D_{\nu_j\nu_j}^{-1} \\ 0 \end{bmatrix}, \tag{9.16}$$

for $j \in V^{\mathrm{s}}$. This equation is the $\gamma_j \times \nu_j$ subblock of the matrix equation $A^{-1}L = L^{-T}D^{-1}$.

The following outline summarizes the supernodal algorithm for computing the projected inverse. The nodal algorithm (based on equation (9.15)) is a special case for $T^{\mathrm{s}} = T$, $V^{\mathrm{s}} = \{1, 2, \ldots, n\}$, $\nu_j = j$.

Algorithm 9.3 (Projected inverse).

Input. The supernodal Cholesky factors L, D of a positive definite matrix $A \in \mathbf{S}_E^n$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered supernodal elimination tree for G_σ.

Output. The projected inverse $B = \Pi_E(A^{-1})$.

Algorithm. Enumerate the representative vertices $j \in V^{\mathrm{s}}$ in inverse topological order. For each j execute the following two steps.

- Calculate $B_{\alpha_j\nu_j}$ and $B_{\nu_j\nu_j}$ from

$$B_{\alpha_j\nu_j} = -U_j L_{\alpha_j\nu_j}, \qquad B_{\nu_j\nu_j} = D_{\nu_j\nu_j}^{-1} - B_{\alpha_j\nu_j}^T L_{\alpha_j\nu_j}. \tag{9.17}$$

- For $i \in \mathrm{ch}(j)$ compute the update matrix

$$U_i = P_{\alpha_i} P_{\gamma_j}^T \begin{bmatrix} B_{\nu_j\nu_j} & B_{\alpha_j\nu_j}^T \\ B_{\alpha_j\nu_j} & U_j \end{bmatrix} P_{\gamma_j} P_{\alpha_i}^T. \tag{9.18}$$

The cost of computing the projected inverse is similar to that of computing the Cholesky factorization. This is illustrated in Figure 9.5 which shows a scatter plot of the computation times for the projected inverse and the Cholesky factorization. Indeed, the computation times are roughly the same for large problems. The results are based on the test problems that were described in §9.4.

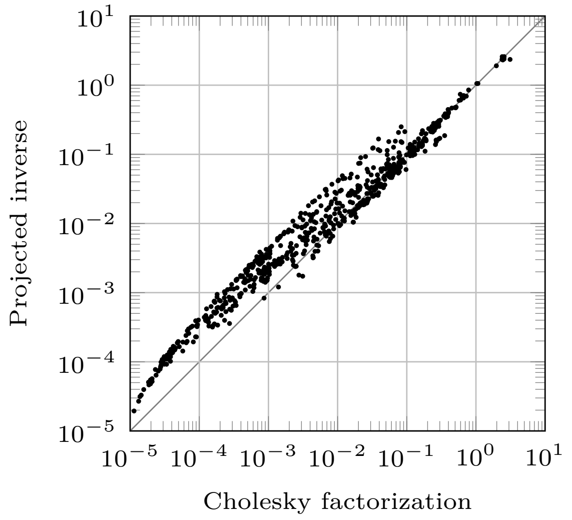


Figure 9.5: Scatter plot of computation times (in seconds) for multifrontal projected inverse and Cholesky factorization.

9.6 Logarithmic barrier

The function $\phi : \mathbf{S}^n_E \to \mathbf{R}$, with domain $\mathbf{dom}\,\phi = \mathbf{S}^n_{++} \cap \mathbf{S}^n_E$ (the positive definite matrices in $\mathbf{S}^n_E$) and function value

$$\phi(S) = -\log\det S, \tag{9.19}$$

is important in semidefinite optimization as a logarithmic barrier for the cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$. It is readily evaluated from a Cholesky factorization of S. For example, if a factorization $P_\sigma S P_\sigma^T = LDL^T$ is available, with D positive diagonal and L unit lower-triangular, then $\phi(S) = -\sum_i \log D_{ii}$.

The gradient of ϕ (for the trace inner product $\langle A, B\rangle = \mathbf{tr}(AB)$) is defined as

$$\nabla\phi(S) = -\Pi_E(S^{-1}),$$

i.e., the negative of the projected inverse, and can be efficiently computed using Algorithm 9.3. Similar algorithms exist for higher derivatives. The Hessian of ϕ at S is a linear operator that maps a matrix

$Y \in \mathbf{S}^n_E$ to the matrix

$$\nabla^2\phi(S)[Y] = \frac{d}{dt}\nabla\phi(S+tY)\bigg|_{t=0} = \Pi_E(S^{-1}YS^{-1}).$$

This expression can be evaluated without explicitly computing the inverse S^{-1} or the (generally dense) matrix product $S^{-1}YS^{-1}$, via two recursions on an elimination tree, one in topological order and the second in an inverse topological order [61, 9]. The two recursions in the algorithm can be interpreted as linearizations of the calculations needed to compute the gradient at S. The first recursion, in topological order, computes the directional derivatives of the Cholesky factors of S in the direction Y. The second recursion, in inverse topological order, computes the directional derivative of the projected inverse. It is also interesting to note that the linear operators evaluated by the two recursions are adjoints. This provides a useful factorization of the Hessian $\nabla^2\phi(S)$ as a composition $\nabla^2\phi(S)[Y] = \mathcal{R}^*_S(\mathcal{R}_S(Y))$ of a linear operator $\mathcal{R}_S$ and its adjoint $\mathcal{R}^*_S$. Moreover, the inverse operator $\mathcal{R}_S^{-1}$ and its adjoint can also be evaluated using similar algorithms, leading to efficient methods for evaluating the inverse of $\nabla^2\phi(S)$. These algorithms and their applications are discussed in greater detail in [61, 9].

10

Positive Semidefinite Matrix Completion

In a positive semidefinite completion problem, one is given a sparse matrix $A \in \mathbf{S}^n_E$, representing a partially specified dense matrix. The diagonal entries A_{ii} and the off-diagonal entries A_{ij} with $\{i, j\} \in E$ are fixed; the off-diagonal entries for $\{i, j\} \notin E$ are free. The problem is to find values for the free entries that make the matrix positive semidefinite. Geometrically, we are expressing the matrix $A \in \mathbf{S}^n_E$ as the projection $\Pi_E(X)$ of a positive semidefinite matrix X on the subspace $\mathbf{S}^n_E$. Positive semidefinite completion problems have been studied extensively since the 1980s [76, 108, 20]. In this section we survey the most important results and present algorithms for solving positive semidefinite completion problems with chordal sparsity patterns E. Surveys of the vast literature on this subject can be found in [121, 141, 142] and the book [19].

10.1 Positive semidefinite completable matrix cone

We denote by $\Pi_E(\mathbf{S}^n_+)$ the set of matrices in $\mathbf{S}^n_E$ that have a positive semidefinite completion:

$$\Pi_E(\mathbf{S}^n_+) = \{\Pi_E(X) \mid X \in \mathbf{S}^n_+\}.$$

This is a proper convex cone. It is the projection of the positive semidefinite cone in $\mathbf{S}^n$ on the subspace $\mathbf{S}^n_E$ and therefore itself a convex cone, with nonempty interior in $\mathbf{S}^n_E$. It is pointed because $A \in \Pi_E(\mathbf{S}^n_+)$ and $-A \in \Pi_E(\mathbf{S}^n_+)$ implies that $A = \Pi_E(X) = -\Pi_E(Y)$ for some $X \succeq 0$ and $Y \succeq 0$ and, hence, $\Pi_E(X+Y) = 0$, $X + Y \succeq 0$, which is only possible if $X = Y = 0$ (since $\Pi_E(X+Y) = 0$ implies that the diagonal of $X + Y$ is zero). Closedness of $\Pi_E(\mathbf{S}^n_+)$ follows from [192, theorem 9.1], which states that if $\mathcal{C}$ is a closed convex cone and $\mathcal{A}$ a linear mapping that satisfies

$$\text{nullspace}(\mathcal{A}) \cap \mathcal{C} = \{0\},$$

then $\mathcal{A}(\mathcal{C})$ is closed. Applying this to $\mathcal{C} = \mathbf{S}^n_+$ and $\mathcal{A} = \Pi_E$ shows that $\Pi_E(\mathbf{S}^n_+)$ is closed. (In this case, $\Pi_E(X) = 0$ implies that the diagonal of X is zero, which for a positive semidefinite matrix implies $X = 0$.)

The positive semidefinite completable cone $\Pi_E(\mathbf{S}^n_+)$ and the positive semidefinite cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ form a pair of dual cones in $\mathbf{S}^n_E$. To see this, we first verify that the dual of $\Pi_E(\mathbf{S}^n_+)$ is $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$:

$$\begin{aligned}
(\Pi_E(\mathbf{S}^n_+))^* &= \{B \in \mathbf{S}^n_E \mid \mathbf{tr}(AB) \geq 0 \ \ \forall A \in \Pi_E(\mathbf{S}^n_+)\} \\
&= \{B \in \mathbf{S}^n_E \mid \mathbf{tr}(\Pi_E(X)B) \geq 0 \ \ \forall X \succeq 0\} \\
&= \{B \in \mathbf{S}^n_E \mid \mathbf{tr}(XB) \geq 0 \ \ \forall X \succeq 0\} \\
&= \mathbf{S}^n_+ \cap \mathbf{S}^n_E.
\end{aligned}$$

Since the dual of the dual of a convex cone is the closure of the cone [192, page 121], we also have

$$(\mathbf{S}^n_+ \cap \mathbf{S}^n_E)^* = (\Pi_E(\mathbf{S}^n_+))^{**} = \mathbf{cl}(\Pi_E(\mathbf{S}^n_+)) = \Pi_E(\mathbf{S}^n_+).$$

These properties hold for any sparsity pattern E, chordal or not. For a chordal pattern, the decomposition result for the cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ in Theorem 9.2 leads to the following characterization of the positive semidefinite completable cone $\Pi_E(\mathbf{S}^n_+)$ [108, theorem 7].

Theorem 10.1. Let E be a chordal sparsity pattern of order n. Then $A \in \Pi_E(\mathbf{S}^n_+)$ if and only if

$$A_{\beta\beta} \succeq 0 \tag{10.1}$$

for all cliques β of the sparsity graph. $A \in \mathbf{int}\,\Pi_E(\mathbf{S}^n_+) = \Pi_E(\mathbf{S}^n_{++})$ if and only if $A_{\beta\beta} \succ 0$ for all cliques β.

Proof. $A \in \Pi_E(\mathbf{S}^n_+)$ if and only if $\mathbf{tr}(AB) \geq 0$ for all $B \in \mathbf{S}^n_+ \cap \mathbf{S}^n_E$. Using the characterization of $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ of Theorem 9.2, this holds if and only if

$$0 \leq \mathbf{tr}\left(\sum_{\beta\in C} P_\beta^T H_\beta P_\beta A\right) = \sum_{\beta\in C} \mathbf{tr}(H_\beta P_\beta A P_\beta^T) \qquad (10.2)$$

for all matrices $H_\beta \succeq 0$. This is equivalent to (10.1). Similarly, $A \in \mathbf{int}\,\Pi_E(\mathbf{S}^n_+)$ if and only if strict inequality holds in (10.2) for all positive semidefinite H_β that are not all zero. This is equivalent to $A_{\beta\beta} \succ 0$ for all cliques β. □

If E is not chordal, the condition (10.1) is necessary but not sufficient. A counterexample can be constructed as follows. Suppose the sparsity graph contains a chordless cycle of length greater than three. After a suitable reordering we can assume that the cycle is $(1, 2, \ldots, k, 1)$ with $k > 3$. Every clique of the sparsity graph contains at most one edge of this cycle, since a second edge would define a chord. Now take $A \in \mathbf{S}^n_E$ to have diagonal one,

$$A_{i,i+1} = A_{i+1,i} = 1, \quad i = 1, \ldots, k-1, \qquad A_{k1} = A_{1k} = -1$$

and $A_{ij} = 0$ for all other edges $\{i, j\} \in E$. This matrix satisfies $A_{\beta\beta} \succeq 0$ for every clique β, because each matrix $A_{\beta\beta}$ is either an identity matrix (if the clique does not contain an edge of the cycle), or a matrix with unit diagonal and one symmetric pair of off-diagonal entries equal to one (if the clique contains an edge $\{i, i+1\}$ of the cycle) or negative one (if the clique contains the edge $\{1, k\}$). However A has no positive semidefinite completion. If X were such a completion, then necessarily $X_{\eta\eta} = \mathbf{1}\mathbf{1}^T$ for the two principal submatrices indexed by $\eta = (1, 2, \ldots, k-1)$ and $\eta = (2, \ldots, k)$. This gives a contradiction because then the submatrix

$$\begin{bmatrix} X_{11} & X_{21} & X_{k1} \\ X_{21} & X_{22} & X_{k2} \\ X_{k1} & X_{k2} & X_{kk} \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

is indefinite.

10.2 Maximum determinant completion

The positive semidefinite completion of a matrix in $\Pi_E(\mathbf{S}^n_+)$ is not unique in general. For matrices in the interior of $\Pi_E(\mathbf{S}^n_+)$, the completion with maximum determinant is of special interest. The maximum determinant positive definite completion of a matrix $A \in \Pi_E(\mathbf{S}^n_{++})$ is defined as the solution of the optimization problem

$$\begin{array}{ll} \text{maximize} & \log\det W \\ \text{subject to} & \Pi_E(W) = A, \end{array} \tag{10.3}$$

with variable $W \in \mathbf{S}^n$, where we define the domain of $\log\det W$ to be the set of positive definite matrices $\mathbf{S}^n_{++}$. The solution W is also called the *maximum entropy* completion of A, since it maximizes the entropy of the normal distribution $N(0, W)$, which is given by

$$\frac{1}{2}(\log\det W + n\log(2\pi) + n),$$

subject to the constraint $\Pi_E(W) = A$; see [68].

The Lagrange dual of (10.3) is the minimization problem

$$\text{minimize} \quad \mathbf{tr}(AY) - \log\det Y - n, \tag{10.4}$$

where the variable $Y \in \mathbf{S}^n_E$ is a Lagrange multiplier for the equality constraint in (10.3). (As elsewhere, we define the domain of $-\log\det Y$ to be $\mathbf{S}^n_{++} \cap \mathbf{S}^n_E$ so there is an implicit constraint that Y is positive definite.) The dual problem (10.4) has an important application in statistics and machine learning. The solution Y can be shown to be the inverse of the maximum likelihood estimate of the covariance matrix of a Gaussian random vector variable, subject to a set of conditional independence constraints. In this interpretation A is the projection $\Pi_E(C)$ on $\mathbf{S}^n_E$ of a sample covariance matrix C. Absence of an edge $\{i, j\}$ in E indicates that the components i and j of the random vector are conditionally independent [68] [146, chapter 5]. Another application of problem (10.4) will be discussed in section 10.5.

The optimality conditions for the convex optimization problem (10.3) and its dual are

$$\Pi_E(W) = A, \qquad W \succ 0, \qquad W^{-1} = Y. \tag{10.5}$$

The two equalities show that the inverse of the maximum determinant completion W has sparsity pattern E. Since W is usually dense, its inverse or the Cholesky factors of the inverse therefore provide a convenient and economical representation of the maximum determinant completion W.

We now show that if E is a chordal sparsity pattern, then the Cholesky factors of $W^{-1} = Y$ can be computed directly from A, by exploiting the same equations (9.15) and (9.16) as used in the algorithm for computing the projected inverse of a matrix in $\mathbf{S}^n_{++} \cap \mathbf{S}^n_E$. This also gives a constructive proof that (10.3) is solvable for all $A \in \Pi_E(\mathbf{S}^n_{++})$. As in the previous sections, we consider a chordal sparsity pattern E with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$ and formulate the algorithm in terms of a supernodal elimination tree T^{s}. We make the assumptions and use the index set notation of section 8.2.

The factors L and D in a supernodal Cholesky factorization $W^{-1} = LDL^T$ satisfy $WL = L^{-T}D^{-1}$. The constraint $\Pi_E(W) = A$ specifies the submatrices $W_{\gamma_i\gamma_i}$, so the $\gamma_j \times \nu_j$ subblock of the equation $WL = L^{-T}D^{-1}$ can be written

$$\begin{bmatrix} A_{\nu_j\nu_j} & A^T_{\alpha_j\nu_j} \\ A_{\alpha_j\nu_j} & A_{\alpha_j\alpha_j} \end{bmatrix} \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix} = \begin{bmatrix} D^{-1}_{\nu_j\nu_j} \\ 0 \end{bmatrix}. \tag{10.6}$$

The equation (10.6) determines $L_{\alpha_j\nu_j}$ and $D_{\nu_j\nu_j}$ uniquely if $A_{\gamma_j\gamma_j} \succ 0$:

$$\begin{aligned} L_{\alpha_j\nu_j} &= -A^{-1}_{\alpha_j\alpha_j} A_{\alpha_j\nu_j}, \\ D_{\nu_j\nu_j} &= \left(A_{\nu_j\nu_j} + A^T_{\alpha_j\nu_j} L_{\alpha_j\nu_u}\right)^{-1} \\ &= \left(A_{\nu_j\nu_j} - A^T_{\alpha_j\nu_j} A^{-1}_{\alpha_j\alpha_j} A_{\alpha_j\nu_j}\right)^{-1}. \end{aligned}$$

The last expression also shows that $D_{\nu_j\nu_j}$ is positive definite. The order in which we enumerate the supernodes ν_j to calculate $L_{\alpha_j\nu_j}$ and $D_{\nu_j\nu_j}$ in this method is arbitrary. However, if we use an inverse topological order we obtain an algorithm that is very similar to the multifrontal algorithms of the previous chapter.

Algorithm 10.1 (Factorization of maximum determinant completion).

Input. A matrix $A \in \Pi_E(\mathbf{S}^n_{++})$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered supernodal elimination tree for G_σ.

Output. The supernodal Cholesky factors L, D of the inverse W^{-1} of the maximum determinant positive definite completion of A.

Algorithm. Enumerate the representative vertices $j \in V^{\mathrm{s}}$ in inverse topological order. For each j, execute the following two steps.

- Calculate $L_{\alpha_j \nu_j}$ and $D_{\nu_j \nu_j}$ from

$$L_{\alpha_j \nu_j} = -U_j^{-1} A_{\alpha_j \nu_j}, \quad D_{\nu_j \nu_j} = (A_{\nu_j \nu_j} + A_{\alpha_j \nu_j}^T L_{\alpha_j \nu_j})^{-1}. \quad (10.7)$$

- For $i \in \mathrm{ch}(j)$ compute the update matrix

$$U_i = P_{\alpha_i} P_{\gamma_j}^T \begin{bmatrix} A_{jj} & A_{\alpha_j j}^T \\ A_{\alpha_j j} & U_j \end{bmatrix} P_{\gamma_j} P_{\alpha_i}^T. \quad (10.8)$$

The update matrices are defined as $U_i = A_{\alpha_i \alpha_i}$. If A is stored in a CCS format, propagating these submatrices as temporary dense matrix variables is easier than retrieving them from the CCS data structure. It also allows us to exploit the fact that (10.8) typically involves the addition and deletion of only a few rows and columns of U_j. This makes it possible to efficiently update the factorization of U_j, needed to solve the equation in (10.7). The details are discussed in [9]. Figure 10.1 shows a scatter plot comparing the computation times for the multifrontal Cholesky factorization and the inverse completion factorization for the sparsity patterns in the benchmark set.

It is interesting to examine what happens if we allow matrices A on the boundary of $\Pi_E(\mathbf{S}_+^n)$. From Theorem 10.1 we know that if $A \in \Pi_E(\mathbf{S}_+^n)$, then the submatrices

$$A_{\gamma_j \gamma_j} = \begin{bmatrix} A_{\nu_j \nu_j} & A_{\alpha_j \nu_j}^T \\ A_{\alpha_j \nu_j} & A_{\alpha_j \alpha_j} \end{bmatrix}, \quad j \in V^{\mathrm{s}},$$

are positive semidefinite but possibly singular. Equivalently,

$$A_{\alpha_j \alpha_j} \succeq 0, \qquad A_{\alpha_j \nu_j} = A_{\alpha_j \alpha_j} A_{\alpha_j \alpha_j}^+ A_{\alpha_j \nu_j}, \quad (10.9)$$

and

$$A_{\nu_j \nu_j} - A_{\alpha_j \nu_j}^T A_{\alpha_j \alpha_j}^+ A_{\alpha_j \nu_j} \succeq 0, \quad (10.10)$$

where $A_{\alpha_j \alpha_j}^+$ is the pseudoinverse. Using these properties, one can show that if the inverses in (10.7) are replaced with pseudoinverses, *i.e.*, $L_{\alpha_j \nu_j}$ and $D_{\nu_j \nu_j}$ are computed as

$$L_{\alpha_j \nu_j} = -U_j^+ A_{\alpha_j \nu_j}, \qquad D_{\nu_j \nu_j} = (A_{\nu_j \nu_j} + A_{\alpha_j \nu_j}^T L_{\alpha_j \nu_j})^+, \quad (10.11)$$

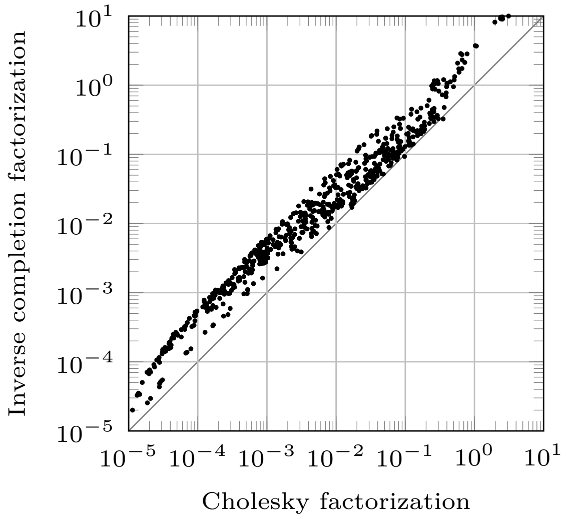


Figure 10.1: Scatter plot of computation times (in seconds) for multifrontal inverse factorization of the maximum determinant completion and Cholesky factorization.

then the matrix $W = L^{-T}D^{+}L^{-1}$ is a positive semidefinite completion of A. To see this, we first note that (10.9) and (10.10) imply that the matrix

$$D_{\nu_j\nu_j} = (A_{\nu_j\nu_j} + A^T_{\alpha_j\nu_j}L_{\alpha_j\nu_j})^{+} = (A_{\nu_j\nu_j} - A^T_{\alpha_j\nu_j}A^{+}_{\alpha_j\alpha_j}A_{\alpha\nu_j})^{+}$$

is positive semidefinite. Therefore the matrices LDL^T and $W = L^{-T}D^{+}L^{-1}$ are positive semidefinite. Next, we use the equation $WL = L^{-T}D^{+}$ and, more specifically, its $\gamma_j \times \nu_j$ blocks

$$\begin{bmatrix} W_{\nu_j\nu_j} & W^T_{\alpha_j\nu_j} \\ W_{\alpha_j\nu_j} & W_{\alpha_j\alpha_j} \end{bmatrix} \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix} = \begin{bmatrix} D^{+}_{\nu_j\nu_j} \\ 0 \end{bmatrix}, \quad j \in V^{\mathrm{s}}, \qquad (10.12)$$

to show that $\Pi_E(W) = A$. First consider the root of T^{s}, *i.e.*, the first representative vertex j in the inverse topological ordering. At the root supernode we have $\nu_j = \gamma_j$ and α_j is empty. Equation (10.12) then reduces to

$$W_{\gamma_j\gamma_j} = W_{\nu_j\nu_j} = D^{+}_{\nu_j\nu_j} = A_{\nu_j\nu_j}.$$

Next suppose that $W_{\gamma_i\gamma_i} = A_{\gamma_i\gamma_i}$ for the proper ancestors of supernode j in the supernodal elimination tree. Therefore $W_{\alpha_j\alpha_j} = A_{\alpha_j\alpha_j}$

in (10.12) and the definition of $L_{\alpha_j\nu_j}$ and $D_{\nu_j\nu_j}$ in (10.11) implies

$$\begin{array}{rcl} W_{\alpha_j\nu_j} & = & -W_{\alpha_j\alpha_j}L_{\alpha_j\nu_j} \\ & = & A_{\alpha_j\alpha_j}A^+_{\alpha_j\alpha_j}A_{\alpha_j\nu_j} \\ & = & A_{\alpha_j\nu_j}, \\ W_{\nu_j\nu_j} & = & D^+_{\nu_j\nu_j} - W^T_{\alpha_j\nu_j}L_{\alpha_j\nu_j} \\ & = & A_{\nu_j\nu_j} - A^T_{\alpha_j\nu_j}A^+_{\alpha_j\alpha_j}A_{\alpha_j\nu_j} + A^T_{\alpha_j\nu_j}A^+_{\alpha_j\alpha_j}A_{\alpha_j\nu_j} \\ & = & A_{\nu_j\nu_j}. \end{array}$$

Hence $W_{\gamma_j\gamma_j} = A_{\gamma_j\gamma_j}$. Continuing the recursion in the inverse topological order we arrive at the conclusion that $W_{\gamma_i\gamma_i} = A_{\gamma_i\gamma_i}$ for all $i \in V^{\mathrm{s}}$.

10.3 Positive semidefinite completion

The Cholesky factors L and D computed in Algorithm 10.1 give a compact, sparse representation of the positive semidefinite completion $W = L^{-T}D^+L^{-1}$ and are sufficient for most applications. If W is needed explicitly, Algorithm 10.1 (and its extension to matrices on the boundary of $\Pi_E(\mathbf{S}^n_+)$) can be modified to compute W directly. Recall from §8.2 that η_j denotes the index set with the row indices of the lower-triangular zeros in column j of the sparsity pattern.

Algorithm 10.2 (Positive semidefinite completion).

Input. A matrix $A \in \Pi_E(\mathbf{S}^n_+)$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered supernodal elimination tree for G_σ.

Output. A positive semidefinite completion W.

Algorithm. Initialize W as $W := A$. Enumerate the supernodes $j \in V^{\mathrm{s}}$ in descending order. For each j, compute

$$W_{\eta_j\nu_j} = W_{\eta_j\alpha_j}W^+_{\alpha_j\alpha_j}W_{\alpha_j\nu_j}, \qquad W_{\nu_j\eta_j} = W^T_{\eta_j\nu_j}. \qquad (10.13)$$

Since this algorithm computes a dense matrix W, we do not use a multifrontal formulation as in the previous algorithms. (The multifrontal formulation is mainly of interest for matrices stored in a sparse matrix format such as the CCS format.)

Algorithm 10.2 completes the matrix W starting in the bottom right corner. After step j the entire submatrix $W_{\beta_j\beta_j}$ with $\beta_j = (j, j+1, \ldots, n)$ has been computed. The expression for $W_{\eta_j\nu_j}$ follows from the $\eta_j \times \nu_j$ block of the equation $WL = L^{-T}D^{+}$, *i.e.*,

$$\begin{bmatrix} W_{\eta_j\nu_j} & W_{\eta_j\alpha_j} \end{bmatrix} \begin{bmatrix} I \\ L_{\alpha_j\nu_j} \end{bmatrix} = 0,$$

with $L_{\alpha_j\nu_j} = -W^{+}_{\alpha_j\alpha_j} W_{\alpha_j\nu_j}$. Note that the indices in α_j do not necessarily precede those in η_j and therefore $W_{\eta_j\alpha_j}$ may contain upper triangular entries of $W_{\beta_j\beta_j}$. This is the reason why a descending enumeration order is used in Algorithm 10.2 and not just any inverse topological order.

Algorithm 10.2 is related to several existing methods for positive semidefinite completion [108, 206]. To show the connection we now give a direct argument (*i.e.*, independent of the results of the previous section and Algorithm 10.1) why the algorithm returns a positive semidefinite completion W for any matrix $A \in \Pi_E(\mathbf{S}^n_+)$.

First, it is clear that the result W is a completion of A because the submatrices $W_{\gamma_i\gamma_i}$, initialized as $A_{\gamma_i\gamma_i}$, are not modified during the algorithm. It is therefore sufficient to show that W is positive semidefinite. In the first cycle of the algorithm j is the root of T^{s}. For this supernode, α_j and η_j are empty and $\beta_j = \nu_j = \gamma_j$, so step (10.13) is vacuous and $W_{\beta_j\beta_j} = A_{\gamma_j\gamma_j} \succeq 0$ holds by the assumption that $A \in \Pi_E(\mathbf{S}^n_+)$. Next suppose that $W_{\beta_i\beta_i} \succeq 0$ for all $i \in V^{\mathrm{s}}$ with $i > j$. The matrix $W_{\beta_j\beta_j}$ is equal to

$$\begin{bmatrix} W_{\nu_j\nu_j} & W^T_{\alpha_j\nu_j} & W^T_{\eta_j\nu_j} \\ W_{\alpha_j\nu_j} & W_{\alpha_j\alpha_j} & W^T_{\alpha_j\eta_j} \\ W_{\eta_j\nu_j} & W_{\eta_j\alpha_j} & W_{\eta_j\eta_j} \end{bmatrix} \tag{10.14}$$

up to a symmetric reordering (because elements in α_j do not necessarily precede those in η_j). We need to find a value for $W_{\eta_j\nu_j}$ that makes this matrix positive semidefinite. The two 2×2 submatrices

$$\begin{bmatrix} W_{\nu_j\nu_j} & W^T_{\alpha_j\nu_j} \\ W_{\alpha_j\nu_j} & W_{\alpha_j\alpha_j} \end{bmatrix}, \qquad \begin{bmatrix} W_{\alpha_j\alpha_j} & W^T_{\alpha_j\eta_j} \\ W_{\eta_j\alpha_j} & W_{\eta_j\eta_j} \end{bmatrix}$$

are positive semidefinite. The first matrix is equal to $A_{\gamma_j\gamma_j}$ and is positive semidefinite if $A \in \Pi_E(\mathbf{S}^n_+)$. The second matrix is a reordering of $W_{\beta_k\beta_k}$, where k is the smallest element in V^{s} greater than j, and is positive semidefinite by the induction hypothesis. Equivalently, using Schur complements, we see that the blocks in these two matrices satisfy

$$W_{\alpha_j\alpha_j} \succeq 0, \qquad \begin{bmatrix} W_{\alpha_j\nu_j}^T \\ W_{\eta_j\alpha_j} \end{bmatrix} = \begin{bmatrix} W_{\alpha_j\nu_j}^T \\ W_{\eta_j\alpha_j} \end{bmatrix} W_{\alpha_j\alpha_j} W_{\alpha_j\alpha_j}^+$$

and

$$\begin{aligned} S_1 &= W_{\nu_j\nu_j} - W_{\alpha_j\nu_j}^T W_{\alpha_j\alpha_j}^+ W_{\alpha_j\nu_j} \succeq 0, \\ S_2 &= W_{\eta_j\eta_j} - W_{\eta_j\alpha_j} W_{\alpha_j\alpha_j}^+ W_{\eta_j\alpha_j}^T \succeq 0. \end{aligned}$$

From these conditions, one can show that the matrix $W_{\eta_j\nu_j} = W_{\eta_j\alpha_j} W_{\alpha_j\alpha_j}^+ W_{\alpha_j\nu_j}$ computed in (10.13) makes (10.14) positive semidefinite. This can be seen by verifying the correctness of the factorization

$$\begin{bmatrix} W_{\nu_j\nu_j} & W_{\alpha_j\nu_j}^T & W_{\eta_j\nu_j}^T \\ W_{\alpha_j\nu_j} & W_{\alpha_j\alpha_j} & W_{\eta_j\nu_j}^T \\ W_{\eta_j\nu_j} & W_{\eta_j\alpha_j} & W_{\eta_j\eta_j} \end{bmatrix} =$$

$$\begin{bmatrix} I & W_{\alpha_j\nu_j}^T & 0 \\ 0 & W_{\alpha_j\alpha_j} & 0 \\ 0 & W_{\eta_j\alpha_j} & I \end{bmatrix} \begin{bmatrix} S_1 & 0 & 0 \\ 0 & W_{\alpha_j\alpha_j}^+ & 0 \\ 0 & 0 & S_2 \end{bmatrix} \begin{bmatrix} I & W_{\alpha_j\nu_j}^T & 0 \\ 0 & W_{\alpha_j\alpha_j} & 0 \\ 0 & W_{\eta_j\alpha_j} & I \end{bmatrix}^T.$$

Hence $W_{\beta_j\beta_j} \succeq 0$. Continuing the enumeration of V^{s} in descending order, we establish that W is positive semidefinite.

Most algorithms for positive semidefinite completion are based on a similar reduction to a sequence of 3×3 block completion problems with unknown $3,1$ and $1,3$ blocks; see, for example, [108, 85, 206]. Frakt, Lev-Ari, and Willsky [85, 234] also present an algorithm for a generalization of the maximum entropy completion problem, in which the projection $\Pi_{E'}(W)$ of the optimal completion W in (10.3) is computed, where E' is a chordal extension of the chordal pattern E. They refer to the solution as a maximum entropy *extension* of A.

In some applications, it is of particular importance to find a positive semidefinite completion with the lowest possible rank. The rank of such a minimum rank positive semidefinite completion is upper bounded by the so-called Gram dimension of its sparsity graph which is the smallest integer k such that

$$\Pi_E(\mathbf{S}_+^n) \subseteq \{\Pi_E(BB^T) \mid B \in \mathbf{R}^{n \times k}\}.$$

Laurent and Varvitsiotis [145] have shown that the Gram dimension of a graph is upper bounded by its treewidth plus one. A direct consequence of this is that if $G = (V, E)$ is chordal, then every matrix in $\Pi_E(\mathbf{S}^n_+)$ has a positive semidefinite completion of rank at most $\omega(G)$, the clique number of G.

10.4 Logarithmic barrier

The Legendre transform or conjugate of the logarithmic barrier $\phi(S) = -\log\det S$ for the cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ is a logarithmic barrier for its dual cone $\Pi_E(\mathbf{S}^n_+)$ [175, section 2.4]. We define the *dual barrier* as

$$\begin{aligned} \phi_*(X) &= \sup_{S\in\mathbf{dom}\,\phi} (-\mathbf{tr}(XS) - \phi(S)) \\ &= \sup_{S\in\mathbf{S}^n_{++}\cap\mathbf{S}^n_E} (-\mathbf{tr}(XS) + \log\det S). \end{aligned} \tag{10.15}$$

($\phi_*(X)$ is the conjugate of ϕ evaluated at $-X$.) The optimization problem in the definition of ϕ_* is a dual maximum determinant positive definite completion problem (10.4), with $A = X$. Therefore the maximizer $\hat{S}$ of the problem in the definition of $\phi_*(X)$ is the inverse of the maximum determinant positive definite completion of X. It is also the unique positive definite solution of the nonlinear equation

$$\Pi_E(S^{-1}) = X$$

with variable $S \in \mathbf{S}^n_E$.

From this it follows that for chordal sparsity patterns E, the value, gradient, and Hessian of the dual barrier $\phi_*(X)$ can be evaluated by the algorithms we have discussed earlier. To evaluate $\phi_*(X)$, we apply Algorithm 10.1 to find a factorization $\hat{S} = LDL^T$ of the optimal $\hat{S}$. The value of the barrier function,

$$\phi_*(X) = \log\det\hat{S} - n$$

is readily evaluated from D (we have $\log\det\hat{S} = \sum_i \log D_{ii}$ for a standard Cholesky factorization and $\log\det\hat{S} = \sum_i \log\det D_{\nu_i\nu_i}$ for a supernodal factorization). The gradient and Hessian of ϕ_* follow from properties that relate the derivatives of convex functions and their conjugates [175, section 2.4]:

$$\nabla\phi_*(X) = -\hat{S}, \qquad \nabla^2\phi_*(X) = \nabla^2\phi(\hat{S})^{-1}.$$

The algorithmic aspects of the dual barrier computations are discussed in more detail in [61, 9].

10.5 Sparse Bregman projection

Let h be a strictly convex and differentiable function. The *generalized distance*, or *D-function*, or *Bregman divergence* associated with h is defined as

$$d(u, v) = h(u) - h(v) - \langle \nabla h(v), u - v\rangle$$

with $\mathbf{dom}\, d = \mathbf{dom}\, h \times \mathbf{dom}\, h$. The function h is called the *kernel function* for d [50, chapter 2]. Two properties follow immediately from convexity of the kernel function. First, $d(u, v)$ is convex in u for fixed v. Second, $d(u, v)$ is nonnegative for all u, v, and zero only if $u = v$. However, in general, $d(u, v) \neq d(v, u)$. For this reason, d is sometimes called a *directed* distance.

A well-known example is the generalized distance for the log-det kernel $h(X) = -\log\det X$ with $\mathbf{dom}\, h = \mathbf{S}^n_{++}$:

$$\begin{aligned} d_{\mathrm{kl}}(U, V) &= -\log\det U + \log\det V + \mathbf{tr}(V^{-1}(U - V)) \\ &= -\log\det(V^{-1}U) + \mathbf{tr}(V^{-1}U) - n, \end{aligned} \qquad (10.16)$$

defined for positive definite U, V. In information theory, $d_{\mathrm{kl}}(U, V)/2$ is known as the *Kullback-Leibler (KL) divergence* or *relative entropy* between zero-mean normal distributions with covariance matrices U and V [137, page 189]. Although d_{kl} is not symmetric, one can note that $d_{\mathrm{kl}}(U, V) = d_{\mathrm{kl}}(V^{-1}, U^{-1})$.

In §10.2 we discussed the maximum determinant positive definite completion of a matrix $A \in \Pi_E(\mathbf{S}^n_{++})$. The corresponding dual problem (10.4) can be written in terms of the KL divergence as

$$\text{minimize} \quad d_{\mathrm{kl}}(Y, C^{-1}) = d_{\mathrm{kl}}(C, Y^{-1}) \qquad (10.17)$$

with variable $Y \in \mathbf{S}^n_E$, where C is any positive definite matrix that satisfies $\Pi_E(C) = A$. The solution Y of this problem is the inverse of the maximum determinant positive definite completion of $\Pi_E(C)$. The first expression for the objective of (10.17) shows that the solution Y is the sparse positive definite matrix 'closest' to C^{-1} in the generalized distance. This is an example of a matrix nearness problem based on *Bregman projection* [71]: the solution Y is called the Bregman projection of C^{-1} on the set $\mathbf{S}^n_E$. Bregman projections with KL divergence are useful when positive definite approximations with certain properties (in our case, sparsity) are needed. The second expression for the objective in (10.17) shows that the maximum determinant positive definite completion Y^{-1} of $\Pi_E(C)$ can be thought of as an approximation of C, in the sense that it minimizes $d_{\mathrm{kl}}(C, Y^{-1})$.

If the sparsity pattern E is chordal, the optimal Y (more precisely, a Cholesky factorization of Y) can be found via Algorithm 10.1, roughly at the same cost as a sparse Cholesky factorization of a positive definite matrix with the same sparsity pattern. Moreover, the algorithm only requires knowledge of $\Pi_E(C)$, not the entire matrix C or C^{-1}.

Bregman projection on sets of chordal sparse matrices is useful in numerical linear algebra and nonlinear optimization, since it provides an inexpensive method for approximating the inverse C^{-1} of a dense positive definite matrix C by a sparse positive definite matrix Y. An application to band patterns is discussed in [170]. In [12] the idea was used in a fast approximate interior-point solver for large quadratic programs with dense Hessian matrices, with applications in machine learning.

The sparse Bregman projection has an interesting tie to matrix approximation via partial Cholesky factorization. Suppose we compute the first k columns of the Cholesky factorization of a positive definite matrix C of order n, *i.e.*, we factor C as

$$C = \begin{bmatrix} C_{11} & C_{21}^T \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ L_2 & I \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} L_1^T & L_2^T \\ 0 & I \end{bmatrix} \qquad (10.18)$$

where C_{11}, L_1, and D_1 are $k \times k$ with L_1 unit lower triangular and D positive diagonal, and

$$S = C_{22} - L_2 D L_2^T = C_{22} - C_{21} C_{11}^{-1} C_{21}^T$$

is the Schur complement of C_{11} in C. Then, if we define D_2 as the diagonal matrix with the diagonal entries of S (*i.e.*, $D_2 = \mathbf{diag}(\mathbf{diag}(S))$), we obtain an approximation of C by substituting D_2 for S in (10.18):

$$\begin{aligned} X &= \begin{bmatrix} X_{11} & X_{21}^T \\ X_{21} & X_{22} \end{bmatrix} \\ &= \begin{bmatrix} L_1 & 0 \\ L_2 & I \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} L_1^T & L_2^T \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} C_{11} & C_{21}^T \\ C_{21} & D_2 + L_2 D_1 L_2^T \end{bmatrix}. \end{aligned}$$

The approximation X clearly satisfies $\Pi_E(X) = \Pi_E(C)$ if we define E to be the block-arrow sparsity pattern (see §8) with dense 11-, 12-, and 21-blocks, and diagonal 22-block, *i.e.*, cliques $\{1, \ldots, k, k+i\}$ for $i = 1, \ldots, n-k$. In other words, X satisfies

$$X_{11} = C_{11}, \qquad X_{21} = C_{21}, \qquad \mathbf{diag}(X_{22}) = \mathbf{diag}(C_{22}).$$

In fact, X is the maximum determinant positive definite completion of $\Pi_E(C)$. This follows from the optimality conditions (10.5) by noting that the inverse of X, given by

$$Y = X^{-1} = \begin{bmatrix} L_1^{-T}(D_1^{-1} + L_2^T D_2^{-1} L_2) L_1^{-1} & -L_1^{-T} L_2^T D_2^{-1} \\ -D_2^{-1} L_2 L_1^{-1} & D_2^{-1} \end{bmatrix},$$

is sparse with sparsity pattern E. Matrix approximation by means of partial Cholesky factorization can therefore be viewed as a special case of sparse matrix approximation via KL divergence minimization.

The partial Cholesky factorization technique can be combined with symmetric pivoting to adaptively choose which columns of the matrix C to use for the approximation. One such approach is to greedily select the next column to include in the partial factorization based on the largest diagonal element of $\mathbf{diag}(S)$. This approach has been used to compute preconditioners for large, dense positive definite systems of equations [102, 23]. It has also been used in the context of nonlinear support vector machine training to compute an approximate factorization of a large, dense kernel matrix [81].

10.6 Sparse quasi-Newton updates

Variable metric methods for unconstrained minimization of smooth functions f take the form

$$x^{k+1} = x^k - t_k (B^k)^{-1} \nabla f(x^k) \tag{10.19}$$

where $t_k > 0$ is a step size and B^k is a positive definite approximation of the Hessian $\nabla^2 f(x^k)$. Quasi-Newton methods are variable metric methods that update B^k or its inverse H^k at each iteration, based on changes in the gradient, by imposing the *secant condition*

$$B^{k+1} s^k = y^k,$$

where $s^k = x^{k+1} - x^k$ and $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$ [70][177, chapter 6]. The most widely used quasi-Newton method uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

$$\begin{aligned} B^+ &= B + \frac{1}{y^T s} y y^T - \frac{1}{s^T B s} B s s^T B & (10.20) \\ H^+ &= (I - \frac{1}{s^T y} s y^T) H (I - \frac{1}{s^T y} y s^T) + \frac{1}{s^T y} s s^T, & (10.21) \end{aligned}$$

where we omit the superscripts in B^k, H^k, y^k, s^k, and write $B^+ = B^{k+1}$, $H^+ = H^{k+1}$. Switching y and s, and B and H in these formulas gives the Davidon-Fletcher-Powell (DFP) quasi-Newton update

$$\begin{aligned} B^+ &= (I - \frac{1}{s^T y} y s^T) B (I - \frac{1}{s^T y} s y^T) + \frac{1}{s^T y} y y^T & (10.22) \\ H^+ &= H + \frac{1}{y^T s} s s^T - \frac{1}{y^T H y} H y y^T H. & (10.23) \end{aligned}$$

Fletcher [83] has observed that the BFGS update (10.20) is the solution of the optimization problem

$$\begin{array}{ll} \text{minimize} & d_{\text{kl}}(X, B) \\ \text{subject to} & Xs = y \end{array} \tag{10.24}$$

with variable $X \in \mathbf{S}^n$, *i.e.*, the Bregman projection on the affine set defined by the secant equation. Similarly, the DFP update H^+ in (10.23) is the Bregman projection of H on the affine set $\{X \in \mathbf{S}^n \mid Xy = s\}$.

Several authors have proposed modified quasi-Newton updates that incorporate sparsity constraints on B^k with the goal, for example, to impose the sparsity pattern of the true Hessian $\nabla^2 f(x)$ on the approximate Hessian. Early references on this subject include [200, 219, 202]. Fletcher's variational characterization of the BFGS update suggests two approaches based on Bregman projections. A first approach, proposed in [84], is to start with a sparse positive definite matrix B^0 and, at each iteration, define $B^+ = B^{k+1}$ as the projection of $B = B^k$ on the set $\{X \in \mathbf{S}^n_E \mid Xs = y\}$, the matrices that satisfy the secant condition and have the desired sparsity pattern. This corresponds to restricting the optimization variable in (10.24) to the set $\mathbf{S}^n_E$, and requires solving the convex optimization problem

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(B^{-1}X) - \log\det X \\ \text{subject to} & Xs = y \end{array} \tag{10.25}$$

with variable $X \in \mathbf{S}^n_E$ (and the implicit constraint $X \succ 0$). Unfortunately, there is no simple formula for the sparse BFGS update, analogous to the dense BFGS update formula (10.20), and problem (10.25) must be solved via numerical optimization. A second difficulty is that feasibility of the sparse problem is more difficult to guarantee than in the dense problem (where it is sufficient that $s^T y > 0$, a condition that is automatically satisfied when f is strictly convex). A dual method for (10.25) that can exploit the chordal matrix techniques of §10.2 would work as follows. The Lagrange dual function of (10.25) is

$$\begin{aligned} g(\nu) &= -y^T\nu + \inf_{X \in \mathbf{S}^n_{++} \cap \mathbf{S}^n_E} \left(\mathbf{tr}(B^{-1}X) - \log\det X + \nu^T X s\right) \\ &= -y^T\nu - \phi_*\left(\Pi_E(B^{-1}) + \frac{1}{2}\Pi_E(\nu s^T + s\nu^T)\right), \end{aligned}$$

where ϕ_* is the logarithmic barrier function for the cone $\Pi_E(\mathbf{S}^n_+)$ defined in (10.15). The variable ν is the multiplier for the equality constraint in (10.25). The dual problem is to maximize $g(\nu)$:

$$\text{maximize} \quad -y^T\nu - \phi_*\left(\Pi_E(B^{-1}) + \frac{1}{2}\Pi_E(\nu s^T + s\nu^T)\right). \tag{10.26}$$

This is an unconstrained differentiable problem that can be solved by a first-order algorithm. As we saw in §10.4, evaluating ϕ_* and its gradient involves a maximum determinant matrix completion problem. The relevant expressions are

$$g(\nu) = -y^T\nu - \log\det \hat{X} + n, \qquad \nabla g(\nu) = -y + \hat{X}s$$

where $\hat{X}$ is the inverse of the maximum determinant positive definite completion of the matrix

$$\Pi_E(B^{-1}) + \frac{1}{2}\Pi_E(s\nu^T + \nu s^T). \tag{10.27}$$

The matrix $\Pi_E(B^{-1})$ is the projection of the inverse of a sparse positive definite matrix. If the sparsity pattern E is chordal, the matrix $\Pi_E(B^{-1})$ can be computed by Algorithm 9.3. A factorization of the inverse $\hat{X}$ of the maximum determinant completion of the matrix (10.27) can be computed by Algorithm 10.1, and from this one readily obtains $g(\nu)$ and $\nabla g(\nu)$. Hence, the complexity of one iteration of a first-order method for the dual problem (10.26) is roughly equal to the cost of a sparse Cholesky factorization of a matrix with sparsity pattern E. When evaluated at the dual optimal ν, the matrix $\hat{X}$ gives the solution of (10.25).

A more recent Bregman projection method that overcomes some of the limitations of the first method is the matrix completion quasi-Newton method of Yamashita [239, 62]. Here the update B^+ is computed by Bregman projection of the standard dense BFGS update of B, given by the right-hand side of (10.20), on $\mathbf{S}^n_E$. In other words, $B^+ = B^{k+1}$ is the solution of

$$\text{minimize} \quad \mathbf{tr}(\bar{B}^{-1}X) - \log\det X \tag{10.28}$$

with variable $X \in \mathbf{S}^n_E$, where

$$\begin{aligned} \bar{B}^{-1} &= (I - \frac{1}{s^Ty}sy^T)B^{-1}(I - \frac{1}{s^Ty}ys^T) + \frac{1}{s^Ty}ss^T \\ &= B^{-1} - \frac{1}{s^Ty}(sy^TB^{-1} + B^{-1}ys^T) + \frac{1}{s^Ty}(1 + \frac{y^Ty}{s^Ty})ss^T \end{aligned} \tag{10.29}$$

and $B = B^k$. Problem (10.28) can be solved by Algorithm 10.1, which returns a sparse Cholesky factorization of the solution X. The algorithm takes $\Pi_E(\bar{B}^{-1})$ as input. From (10.29) we see that this requires

$\Pi_E(B^{-1})$, which can be computed from the Cholesky factors of B via Algorithm 9.3, and the projection of three rank-one matrices. Note that $B^{-1}y$ is also easily computed using the Cholesky factorization of B. Kanamori and Ohara [123, 124] have used a similar approach to generalize BFGS and DFP quasi-Newton updates by allowing other Bregman divergences than the KL-divergence.

11

Correlation and Euclidean Distance Matrices

11.1 Correlation matrices

A *correlation matrix* is a positive semidefinite matrix with diagonal entries equal to one. As the name suggests, correlation matrices arise in statistics. If x is a random vector variable with mean $a = \mathbf{E}\, x$ and covariance $\Sigma = \mathbf{E}(x-a)(x-a)^T$, then the matrix of *correlation coefficients*

$$A_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}$$

is positive semidefinite with unit diagonal. Correlation matrices can also be interpreted geometrically, as Gram matrices of sets of unit-norm vectors y_i, $i = 1, \ldots, n$. The component-wise cosine of A gives the matrix of pairwise angles $\cos A_{ij} = \angle(y_i, y_j)$ between the vectors. Conversely, if A is a correlation matrix then a factorization $A = Y^T Y$ provides a set of unit-norm vectors with $A_{ij} = y_i^T y_j$. Correlation matrices are also important in combinatorial optimization [98, 143] and much is known about their geometry [109, 144].

In a *correlation matrix completion problem* one is given a sparse matrix with unit diagonal A and is asked to find a correlation matrix X with $\Pi_E(X) = A$. Geometrically, we are given a set of angles θ_{ij},

$\{i,j\} \in E$, and are asked to find a set of n vectors with $\angle(y_i, y_j) = \theta_{ij}$ for $\{i,j\} \in E$. This is a special case of the positive semidefinite completion problem studied in section 10. (Recall that in the positive semidefinite completion problem, the diagonal entries are among the specified entries, *i.e.*, the constraint $\Pi_E(X) = A$ always includes $X_{ii} = A_{ii}$; see definition (9.13).) In particular, if E is a chordal pattern, then a matrix $A \in \mathbf{S}^n_E$ has a correlation matrix completion if and only if for every clique β in the sparsity graph the submatrix $A_{\beta\beta}$ is a correlation matrix.

11.2 Euclidean distance matrices

A *Euclidean distance matrix* (EDM) is a symmetric matrix whose entries can be expressed as squared pairwise Euclidean distances of a set of points: $A \in \mathbf{S}^n$ is an EDM if there exist vectors $y_1, \ldots, y_n$ such that

$$A_{ij} = \|y_i - y_j\|_2^2, \quad i,j = 1, \ldots, n. \tag{11.1}$$

The dimension of the vectors y_i is arbitrary. We refer to the set of points $\{y_1, \ldots, y_n\}$ as a *realization* of the Euclidean distance matrix. Euclidean distance matrices arise in several important applications and have been studied extensively; see [136] for a recent survey.

Expanding the square in (11.1) we see that A is an EDM if and only if its coefficients can be expressed as

$$A_{ij} = y_i^T y_i + y_j^T y_j - 2 y_i^T y_j,$$

for a set of points $y_1, \ldots, y_n$. The right-hand side only involves the inner products $y_i^T y_j$ and is a linear function of these inner products. In matrix notation, A is an EDM if and only if

$$A = \mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\,\mathbf{diag}(Y)^T - 2Y \tag{11.2}$$

for some positive semidefinite Y, *i.e.*, the Gram matrix

$$Y = \left[\begin{array}{cccc} y_1 & y_2 & \cdots & y_n \end{array}\right]^T \left[\begin{array}{cccc} y_1 & y_2 & \cdots & y_n \end{array}\right].$$

This shows that the set of Euclidean distance matrices is the image of a convex cone under a linear transformation, and therefore itself a convex cone. We will refer to a positive semidefinite matrix Y that satisfies (11.2) as a *Gram matrix for* A.

From the definition (11.1) it is clear that realizations and Gram matrices are not unique. In particular, an EDM is invariant with respect to translations of the points y_i, so the origin in the realization space can be chosen arbitrarily. Without loss of generality, we can therefore impose the condition that the origin is at a given affine combination of the points in the realization, *i.e.*, $\sum_i \mu_i y_i = 0$ (hence, $Y\mu = 0$) for some given μ with $\mathbf{1}^T\mu = 1$. Given any realization $\tilde{y}_1, \ldots, \tilde{y}_n$, we find a realization $y_1, \ldots, y_n$ with $\sum_i \mu_i y_i = 0$, by taking $y_i = \tilde{y}_i - \sum_{j=1}^n \mu_j \tilde{y}_j$ for $i = 1, \ldots, n$. The Gram matrix Y of the points y_i is related to the Gram matrix $\tilde{Y}$ for $\tilde{y}_i$ via

$$Y = (I - \mathbf{1}\mu^T)\tilde{Y}(I - \mu\mathbf{1}).$$

With the extra condition $Y\mu = 0$, the Gram matrix is uniquely determined from (11.2): if Y satisfies (11.2), and $Y\mu = 0$, $\mathbf{1}^T\mu = 1$, then

$$\begin{aligned} Y &= (I - \mathbf{1}\mu^T)Y(I - \mu\mathbf{1}^T) \\ &= -\frac{1}{2}(I - \mathbf{1}\mu^T)\left(\mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\,\mathbf{diag}(Y)^T - 2Y\right)(I - \mu\mathbf{1}^T) \\ &= -\frac{1}{2}(I - \mathbf{1}\mu^T)A(I - \mu\mathbf{1}^T). \end{aligned}$$

Several classical characterizations of Euclidean distance matrices are due to Schoenberg [197, 198].

Theorem 11.1. A symmetric matrix A of order n is a Euclidean distance matrix if and only if

$$\mathbf{diag}(A) = 0, \qquad Q^T A Q \preceq 0 \tag{11.3}$$

where Q is a matrix whose columns span the orthogonal complement of $\mathbf{1}$ (*i.e.*, Q satisfies $\mathbf{1}^T Q = 0$ and $\mathbf{rank}(Q) = n - 1$).

Proof. Suppose A is an EDM and let Y be a Gram matrix for A, *i.e.*, a positive semidefinite matrix that satisfies (11.2). Clearly, $\mathbf{diag}(A) = 0$. Suppose Q satisfies $\mathbf{1}^T Q = 0$. Then

$$\begin{aligned} Q^T A Q &= Q^T\left(\mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\,\mathbf{diag}(Y)^T - 2Y\right)Q \\ &= -2Q^T Y Q. \\ &\preceq 0. \end{aligned}$$

To show that the conditions (11.3) are sufficient we make a specific choice $Q = I - \mu\mathbf{1}^T$, where μ is a vector that satisfies $\mathbf{1}^T\mu = 1$. (Any other matrix Q with the same column space as $I - \mu\mathbf{1}^T$ gives an equivalent condition $Q^TAQ \preceq 0$. The inequality $Q^TAQ \preceq 0$ means that A is negative semidefinite on the complement of $\mathbf{1}$, *i.e.*, $x^TAx \leq 0$ if $\mathbf{1}^Tx = 0$.) Suppose A is a symmetric matrix with $\mathbf{diag}(A) = 0$ and

$$(I - \mathbf{1}\mu^T)A(I - \mu\mathbf{1}) \preceq 0.$$

We show that the positive semidefinite matrix

$$\begin{aligned} Y &= -\frac{1}{2}(I - \mathbf{1}\mu^T)A(I - \mu\mathbf{1}^T) \\ &= -\frac{1}{2}\left(A - \mathbf{1}\mu^TA - A\mu\mathbf{1}^T + (\mu^TA\mu)\mathbf{1}\mathbf{1}^T\right) \end{aligned}$$

satisfies (11.2) and therefore is a Gram matrix for A:

$$\begin{aligned} &\mathbf{diag}(Y)\mathbf{1}^T + \mathbf{1}\,\mathbf{diag}(Y)^T - 2Y \\ &= (A\mu - \frac{\mu^TA\mu}{2}\mathbf{1})\mathbf{1}^T + \mathbf{1}(A\mu - \frac{\mu^TA\mu}{2}\mathbf{1})^T + A \\ &\quad - \mathbf{1}\mu^TA - A\mu\mathbf{1}^T + (\mu^TA\mu)\mathbf{1}\mathbf{1}^T \\ &= A. \end{aligned}$$

□

The condition $(I - \mathbf{1}\mu^T)A(I - \mu\mathbf{1}^T) \preceq 0$ with $\mu = e_1$ appears in [197]. Choosing $\mu = (1/n)\mathbf{1}$ gives the criterion in [198].

11.3 Euclidean distance matrix completion

There is a well-studied Euclidean distance matrix completion problem, analogous to the positive semidefinite completion problem [141, 3]. In the EDM completion problem we express a sparse matrix as the projection of a Euclidean distance matrix on the sparsity pattern: given $A \in \mathbf{S}^n_E$, the problem is to find a Euclidean distance matrix X such that

$$\Pi_E(X) = A.$$

The following fundamental result was proved by Bakonyi and Johnson [18, theorem 3.3].

Theorem 11.2. Let E be a chordal sparsity pattern of order n. A matrix $A \in \mathbf{S}^n_E$ has an EDM completion if and only if for every clique β of the sparsity graph, the matrix $A_{\beta\beta}$ is a Euclidean distance matrix.

We show the result by adapting Algorithm 10.2 to compute an EDM completion for a matrix A that satisfies the condition in Theorem 11.2. We make the assumptions and use the notation of §8.2.

Algorithm 11.1 (Euclidean distance matrix completion).

Input. An EDM-completable matrix $A \in \mathbf{S}^n_E$, where $G = (V, E)$ is a chordal sparsity pattern with perfect elimination ordering $\sigma = (1, 2, \ldots, n)$, and a postordered supernodal elimination tree for G_σ.

Output. A Euclidean distance matrix completion X.

Algorithm. Initialize X as $X := A$. Enumerate the vertices $j \in V^{\mathrm{s}}$ in descending order. For each non-root j we compute $X_{\eta_j\nu_j}$ by the following two steps (η_j is empty at the root vertex j, so there is nothing to compute).

1. Define $\mu = e_{a(j)}$ where $a(j)$ is the first ancestor of supernode j (the first element of α_j). Compute the submatrices $Y_{\alpha_j\nu_j}$, $Y_{\alpha_j\alpha_j}$, $Y_{\eta_j\alpha_j}$ of the matrix on the left-hand side of the equation

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & Y_{\nu_j\nu_j} & Y_{\nu_j\alpha_j} & \times \\ \times & Y_{\alpha_j\nu_j} & Y_{\alpha_j\alpha_j} & Y_{\alpha_j\eta_j} \\ \times & \times & Y_{\eta_j\alpha_j} & Y_{\eta_j\eta_j} \end{bmatrix} =$$
$$-\frac{1}{2}(I - \mathbf{1}\mu^T) \begin{bmatrix} \times & \times & \times & \times \\ \times & X_{\nu_j\nu_j} & X_{\nu_j\alpha_j} & \times \\ \times & X_{\alpha_j\nu_j} & X_{\alpha_j\alpha_j} & X_{\alpha_j\eta_j} \\ \times & \times & X_{\eta_j\alpha_j} & X_{\eta_j\eta_j} \end{bmatrix} (I - \mu\mathbf{1}^T), \quad (11.4)$$

as well as the diagonals of the matrices $Y_{\nu_j\nu_j}$, $Y_{\eta_j\eta_j}$. (The '$\times$' entries on the right-hand side indicate blocks that have not yet been computed at this point; the $\times$ entries on the left are irrelevant.)

2. Complete the $\eta_j \times \nu_j$ block of X by taking

$$\begin{aligned} X_{\eta_j\nu_j} &= -2Y_{\eta_j\alpha_j}Y^+_{\alpha_j\alpha_j}Y_{\alpha_j\nu_j} \\ &\qquad + \mathbf{1}\,\mathbf{diag}(Y_{\nu_j\nu_j})^T + \mathbf{diag}(Y_{\eta_j\eta_j})\mathbf{1}^T. \quad (11.5) \end{aligned}$$

Note that after the initialization the algorithm does not modify the submatrices $X_{\gamma_j\gamma_j}$, so $\Pi_E(X) = A$ holds on exit and it is sufficient to show that the result X is an EDM if the matrix A is EDM-completable, *i.e.*, if $A_{\gamma_j\gamma_j}$ is an EDM for all $j \in V^{\mathrm{s}}$. We verify by induction that after processing the supernode represented by $j \in V^{\mathrm{s}}$, the matrix $X_{\beta_j\beta_j}$, with $\beta_j = \{j, j{+}1, \ldots, n\}$, is an EDM. This property holds trivially if j is the root of the elimination tree, since then $\beta_j = \nu_j = \gamma_j$ and $X_{\beta_j\beta_j} = A_{\gamma_j\gamma_j}$.

Suppose $X_{\beta_i\beta_i}$ is an EDM for all $i \in V^{\mathrm{s}}$ with $i > j$. Since $a(j)$ is the first element of α_j, we have from (11.4) and the results of the previous section that

$$\begin{aligned}\begin{bmatrix} Y_{\alpha_j\alpha_j} & Y_{\alpha_j\eta_j} \\ Y_{\eta_j\alpha_j} & Y_{\eta_j\eta_j} \end{bmatrix} &= -\frac{1}{2}(I - \mathbf{1}e_1^T)\begin{bmatrix} X_{\alpha_j\alpha_j} & X_{\alpha_j\eta_j} \\ X_{\eta_j\alpha_j} & X_{\eta_j\eta_j} \end{bmatrix}(I - e_1\mathbf{1}^T) \\ &\succeq 0. \end{aligned} \tag{11.6}$$

Similarly, if $k = |\nu_j| + 1$,

$$\begin{aligned}\begin{bmatrix} Y_{\nu_j\nu_j} & Y_{\nu_j\alpha_j} \\ Y_{\alpha_j\nu_j} & Y_{\alpha_j\alpha_j} \end{bmatrix} &= -\frac{1}{2}(I - \mathbf{1}e_k^T)\begin{bmatrix} X_{\nu_j\nu_j} & X_{\nu_j\alpha_j} \\ X_{\alpha_j\nu_j} & X_{\alpha_j\alpha_j} \end{bmatrix}(I - e_k\mathbf{1}^T) \\ &= -\frac{1}{2}(I - \mathbf{1}e_k^T)\begin{bmatrix} A_{\nu_j\nu_j} & A_{\nu_j\alpha_j} \\ A_{\alpha_j\nu_j} & A_{\alpha_j\alpha_j} \end{bmatrix}(I - e_k\mathbf{1}^T) \\ &\succeq 0. \end{aligned} \tag{11.7}$$

As we have seen in §10.3, the inequalities (11.6) and (11.7) guarantee that the matrix

$$W = \begin{bmatrix} Y_{\nu_j\nu_j} & Y_{\nu_j\alpha_j} & Y_{\nu_j\eta_j} \\ Y_{\alpha_j\nu_j} & Y_{\alpha_j\alpha_j} & Y_{\alpha_j\eta_j} \\ Y_{\eta_j\nu_j} & Y_{\eta_j\alpha_j} & Y_{\eta_j\eta_j} \end{bmatrix}$$

is positive semidefinite if we define $Y_{\eta_j\nu_j} = Y_{\eta_j\alpha_j}Y_{\alpha_j\alpha_j}^{+}Y_{\alpha_j\nu_j}$. The EDM for which W is a Gram matrix is given by the formula

$$\begin{aligned}\begin{bmatrix} X_{\nu_j\nu_j} & X_{\nu_j\alpha_j} & X_{\nu_j\eta_j} \\ X_{\alpha_j\nu_j} & X_{\alpha_j\alpha_j} & X_{\alpha_j\eta_j} \\ X_{\eta_j\nu_j} & X_{\eta_j\alpha_j} & X_{\eta_j\eta_j} \end{bmatrix} &= \\ -2W + \mathbf{1}\,\mathbf{diag}(W)^T + \mathbf{diag}(W)\mathbf{1}^T&. \end{aligned} \tag{11.8}$$

The $X_{\eta_j\nu_j}$ block of this equation gives the formula (11.5) in step 2 of the algorithm. (The other blocks are the same as at the start of cycle j.)

Up to a symmetric reordering, the matrix (11.8) is equal to $X_{\beta_j\beta_j}$. This completes the proof that $X_{\beta_j\beta_j}$ is an EDM.

If E is not chordal, the condition in Theorem 11.2 is necessary but not sufficient. A counterexample can be constructed as for positive semidefinite completions in §10.1; see [18, page 651]. Suppose the sparsity graph has a chordless cycle $(1, 2, \ldots, k, 1)$ with $k > 3$. Take $A \in \mathbf{S}^n_E$ to have zero diagonal,

$$A_{i,i+1} = A_{i+1,i} = 0, \quad i = 1, \ldots, k-1, \qquad A_{1k} = A_{k1} = 1, \tag{11.9}$$

$A_{ij} = A_{ji} = 1$ for all edges $\{i, j\} \in E$ with $i \leq k$ and $j > k$, and $A_{ij} = A_{ji} = 0$ for edges $\{i, j\}$ with $i, j > k$. This matrix A does not have an EDM completion, because (11.9) would impose the conditions

$$\|y_1 - y_2\|^2 = \|y_2 - y_3\|_2^2 = \cdots = \|y_{k-1} - y_k\|^2 = 0, \quad \|y_1 - y_k\|_2^2 = 1$$

on any realization of an EDM completion. However, one can verify that $A_{\beta\beta}$ is an EDM for every clique β. Every clique β contains at most one edge of the chordless cycle $(1, 2, \ldots, k, 1)$ because a second edge would create a chord. Therefore the matrix $A_{\beta\beta}$ is either zero (if β contains no vertices from $\{1, 2 \ldots, k\}$), or it has one of the following three forms, which are all EDMs:

$$A_{\beta\beta} = \begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & 0 \end{bmatrix}, \quad A_{\beta\beta} = \begin{bmatrix} 0 & 0 & \mathbf{1}^T \\ 0 & 0 & \mathbf{1}^T \\ \mathbf{1} & \mathbf{1} & 0 \end{bmatrix}, \quad A_{\beta\beta} = \begin{bmatrix} 0 & 1 & \mathbf{1}^T \\ 1 & 0 & \mathbf{1}^T \\ \mathbf{1} & \mathbf{1} & 0 \end{bmatrix}.$$

The first case occurs when β contains one vertex from $\{1, \ldots, k\}$, the second case when β contains two vertices i, $i+1$, with $i \in \{1, \ldots, k-1\}$, and the third case when β contains the vertices 1 and k.

12

Partial Separability in Convex Optimization

12.1 Partial separability

Griewank and Toint, in work on structured quasi-Newton algorithms, considered functions that can be decomposed as sums of elementary functions, where each elementary function only depends on a small subset of the variables; see [104, 105, 106] and the more recent discussion in [127]. They defined a function $f : \mathbf{R}^n \to \mathbf{R}$ as *partially separable* if it can be expressed as

$$f(x) = \sum_{k=1}^{l} f_k(A_k x) \tag{12.1}$$

where each matrix A_k has a high-dimensional nullspace. A special case is the class of functions of the form

$$f(x) = \sum_{k=1}^{l} f_k(P_{\beta_k} x) \tag{12.2}$$

where the sets β_k are (small) index sets in $\{1, 2, \ldots, n\}$. We will use the term partial separability in this second, more restricted, sense. Recall that $P_{\beta_k} x = (x_{\beta_k(1)}, \ldots, x_{\beta_k(r_k)})$ if $\beta_k = (\beta_k(1), \ldots, \beta_k(r_k))$, so the kth term in (12.2) depends on a subset of $|\beta_k|$ of the variables.

In Chapter 7 we have already discussed partially separable functions of discrete variables. Here we describe some applications in continuous optimization that have a connection to chordal graphs and graph elimination. As in Chapter 7 we associate with the partially separable structure an undirected *interaction graph* $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ and edge set $E = \{\{i, j\} \mid \{i, j\} \in \beta_k$ for some $\beta_k\}$. The vertices in this graph represent the variables of f; two vertices i and j are adjacent if the variables x_i and x_j appear as arguments in the same elementary function f_k. Absence of an edge $\{i, j\}$ therefore indicates that the function f is separable in the variables x_i and x_j, if the other variables are held fixed. More precisely, if i and j are not adjacent, then the equality

$$f(x + se_i + te_j) = f(x + se_i) + f(x + te_j) - f(x) \tag{12.3}$$

holds for all $x \in \mathbf{R}^n$, $s, t \in \mathbf{R}$ for which x, $x + se_i$, $x + te_j \in \mathbf{dom}\, f$, where e_i and e_j are the ith and jth unit vectors in $\mathbf{R}^n$. To see this, we expand f using (12.2):

$$\begin{aligned}
f(x + se_i + te_j) &= \sum_{i\in\beta_k} f_k(x + se_i) + \sum_{j\in\beta_k} f_k(x + te_j) + \sum_{i,j\notin\beta_k} f_k(x) \\
&= \sum_{i\in\beta_k} f_k(x + se_i) + \sum_{j\in\beta_k} f_k(x) + \sum_{i,j\notin\beta_k} f_k(x) \\
&\quad + \sum_{i\in\beta_k} f_k(x) + \sum_{j\in\beta_k} f_k(x + te_j) + \sum_{i,j\notin\beta_k} f_k(x) \\
&\quad - \sum_{i\in\beta_k} f_k(x) - \sum_{j\in\beta_k} f_k(x) - \sum_{i,j\notin\beta_k} f_k(x)
\end{aligned}$$

and this is equal to (12.3). On the first line we partition the index sets in three groups, using the fact that no β_k contains i and j.

Note that the interaction graph G does not capture the complete partially separable structure (12.2). For example, if $n = 3$, the interaction graph for $n = 3$, $l = 3$, $\beta_1 = (1, 2)$, $\beta_2 = (2, 3)$, $\beta_3 = (1, 3)$, is the complete graph with three vertices, and is identical to the interaction graph for $l = 1$, $\beta_1 = (1, 2, 3)$. More generally, a function can have a complete interaction graph and be partially separable with elementary functions that only depend on pairs of variables. A quadratic function with a dense Hessian is an example. When analyzing partial separability via the interaction graph, one can therefore assume that the sets β_k

are cliques of the interaction graph, since any finer partial separability structure is ignored in a graphical analysis.

If the elementary functions f_k in (12.2) are convex and twice differentiable, then the Hessian of f has the form

$$\nabla^2 f(x) = \sum_{k=1}^{l} P_{\beta_k}^T H_k P_{\beta_k}, \tag{12.4}$$

where $H_k = \nabla^2 f_k(P_{\beta_k} x)$, so $\nabla^2 f(x)$ is positive semidefinite with sparsity pattern G. The decomposition theorem for positive semidefinite matrices (Theorem 9.2) shows that for chordal sparsity patterns the converse holds: if $\nabla^2 f(x)$ is positive semidefinite with a chordal sparsity graph G, then it can be decomposed in the form (12.4) with positive semidefinite matrices H_k, where the sets β_k are the cliques of G. This was the motivation that led to the theorem in [106]. The construction of the counterexample on page 342 shows the importance of chordality. Consider, for example, the quadratic function

$$f(x_1, x_2, x_3, x_4) = \lambda(x_1^2 + x_2^2 + x_3^2 + x_4^2) + 2(x_1x_2 + x_2x_3 + x_3x_4 - x_1x_4).$$

The function is partially separable with index sets $\beta_1 = (1,2)$, $\beta_2 = (2,3)$, $\beta_3 = (3,4)$, $\beta_4 = (1,4)$ and its interaction graph is a chordless cycle of length 4. The general form of a decomposition of f as a partially separable sum with quadratic elementary functions is

$$\begin{aligned} f(x) = {} & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} u_1 & 1 \\ 1 & \lambda - u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ & + \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}^T \begin{bmatrix} u_2 & 1 \\ 1 & \lambda - u_3 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} \\ & + \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}^T \begin{bmatrix} u_3 & 1 \\ 1 & \lambda - u_4 \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \\ & + \begin{bmatrix} x_1 \\ x_4 \end{bmatrix}^T \begin{bmatrix} \lambda - u_1 & -1 \\ -1 & u_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_4 \end{bmatrix}, \end{aligned}$$

where the parameters u_1, u_2, u_3, u_4 can be chosen arbitrarily. As shown in §9.2, if $\sqrt{2} \le \lambda < 2$, then the function f is convex but there exist no values of u_1, u_2, u_3, u_4 that make all four terms in the decomposition convex.

The convex conjugate of a partially separable function possesses an interesting type of dual partial separability structure. It can be shown that if the functions f_k in (12.2) are convex and there exists an element $\bar{x}$ with $P_{\beta_k}\bar{x}$ in the relative interior of $\mathbf{dom}\, f_k$ for $k = 1, \ldots, l$, then the conjugate of f is

$$f^*(s) = \inf\{\sum_{k=1}^{l} f_k^*(\tilde{s}_k) \mid \sum_{k=1}^{l} P_{\beta_k}^T \tilde{s}_k = s\}.$$

Moreover, for each s the minimum in the expression for $f^*(s)$ is attained if it is finite [192, theorem 16.3].

Partial separability, and an extension called *group partial separability*, are exploited in nonlinear optimization modeling software [56], quasi-Newton and trust-region algorithms [104, 105], [177, section 7.4] [57, page 360], and automatic differentiation algorithms [107, section 11.2]. It also plays an important role in sparse semidefinite relaxations of polynomial optimization problems [131, 128, 228, 140, 229, 125]. The authors of [228, 229, 125] consider polynomial optimization problems

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{m} f_k(P_{\beta_k} x) \\ \text{subject to} & f_k(P_{\beta_k} x) \leq 0, \quad k = m+1, \ldots, l, \end{array}$$

where $f_1, \ldots, f_m$ are monomials and $f_{m+1}, \ldots, f_l$ are polynomials. They refer to the interaction graph defined by the index sets $\beta_1, \ldots, \beta_l$ as the *correlative sparsity graph*. They use a chordal extension of the correlative sparsity graph to define a sparse semidefinite relaxation that involves sums of squares of polynomials, where each polynomial depends only on the variables indexed by the vertices in a clique of the chordal extension.

12.2 Partially separable matrix cones

The definition of partial separability can be extended to sets, by defining a partially separable set as a set with a partially separable indicator function. Hence, a set $\mathcal{C}$ in $\mathbf{R}^n$ is partially separable if it can be

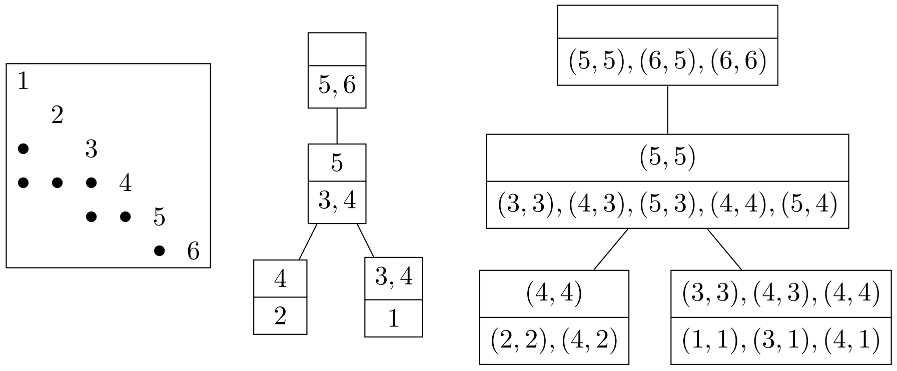


Figure 12.1: *Left.* A chordal sparsity pattern $G' = (V', E')$ of order 6. *Center.* Clique tree with the four cliques $\gamma_1, \ldots, \gamma_4$ of G'. *Right.* Clique tree for the interaction graph $G = (V, E)$ where V are the positions of the lower-triangular nonzeros in the sparsity pattern (denoted by their row and column indices). Two elements (i_1, j_1) and (i_2, j_2) of V are adjacent in G if $i_1, j_1, i_2, j_2 \in \gamma_k$ for some k.

expressed as

$$\mathcal{C} = \{x \in \mathbf{R}^n \mid P_{\beta_k} x \in \mathcal{C}_k,\ k = 1, \ldots, l\}, \tag{12.5}$$

where each $\mathcal{C}_k$ is a set in $\mathbf{R}^{|\beta_k|}$. The indicator function of $\mathcal{C}$ is the partially separable function

$$\delta_{\mathcal{C}}(x) = \sum_{k=1}^{l} \delta_{\mathcal{C}_k}(P_{\beta_k} x).$$

(We use the notation $\delta_{\mathcal{S}}(x)$ for the 0-∞ indicator function of a set $\mathcal{S}$, *i.e.*, the function with domain $\mathcal{S}$ and value $\delta_{\mathcal{S}}(x) = 0$ for $x \in \mathcal{S}$.)

In §10.1 and §11.3 we have encountered two important examples of partially separable convex cones. Theorem 10.1 states that a sparse symmetric matrix with a chordal sparsity pattern has a positive semidefinite completion if and only if all its dense principal submatrices are positive semidefinite. Theorem 11.2 gives a similar decomposition result for sparse matrices with a Euclidean distance matrix completion. Figure 12.1 illustrates how this defines a partially separable structure with chordal interaction graph. It shows a sparsity pattern $G' = (V', E')$ of order 6. The graph G' is chordal and the numerical ordering $\sigma = (1, 2, 3, 4, 5, 6)$ is a perfect elimination ordering. There are four cliques $\gamma_1 = (1,3,4)$, $\gamma_2 = (2,4)$, $\gamma_3 = (3,4,5)$, $\gamma_4 = (5,6)$. Each of these cliques defines a maximal 'dense' principal submatrix in the sparsity pattern. The tree in the center of the figure is a clique tree with the induced subtree property. We can associate with the sparsity pattern G' a second graph $G = (V, E)$, that has as its vertices the positions of the 13 lower-triangular nonzero entries as its vertices. Two vertices (i_1, j_1) and (i_2, j_2) are adjacent in G if there is a clique γ_k of G' that contains all of the indices i_1, j_1, i_2, j_2. In other words, $\{(i_1, j_1), (i_2, j_2)\} \in E$ if (i_1, j_1) and (i_2, j_2) are nonzero positions in some dense principal submatrix of the sparsity pattern. The cliques in G are the sets $\{(i,j) \in \gamma_k \times \gamma_k \mid i \geq j\}$ for $k = 1, \ldots, l$. The kth clique of G contains the lower-triangular positions in the maximal dense principal submatrix indexed by $\gamma_k \times \gamma_k$. The graph G is chordal and there is a direct correspondence between clique trees for G and G'.

To be more specific about how the two matrix cones are special cases of (12.5), we introduce some vector notation for sparse symmetric matrices. For $A \in \mathbf{S}^p_{E'}$, we define $\mathrm{vec}_{E'}(A)$ as the vector of length $n = |E'| + p$ containing the diagonal entries and the lower-triangular entries A_{ij} for $\{i,j\} \in E$, in some arbitrary but fixed ordering, and with the off-diagonal elements scaled by $\sqrt{2}$, so that $\mathbf{tr}(AB) = \mathrm{vec}_{E'}(A)^T \mathrm{vec}_{E'}(B)$ for all $A, B \in \mathbf{S}^p_E$. A similar notation $\mathrm{vec}(A)$ (without subscript) is used for a dense symmetric matrix $A \in \mathbf{S}^r$ to denote the $r(r+1)/2$-vector with the lower-triangular elements A_{ij} in some ordering, and with the off-diagonal elements scaled by $\sqrt{2}$.

Now consider the cone $\Pi_{E'}(\mathbf{S}^p_+)$ of sparse matrices that have a positive semidefinite completion, where E' is a chordal sparsity pattern. Suppose the sparsity graph $G' = (\{1,2,\ldots,p\}, E')$ has l cliques. Let γ_k, $k = 1, \ldots, l$, be an index set in $\{1,2,\ldots,p\}$ with the elements of the kth clique, ordered numerically. For each clique γ_k we define an index set β_k in $\{1,2,\ldots,n\}$ with the indices in the n-vector $\mathrm{vec}_{E'}(A)$ of the entries of the corresponding principal submatrix $A_{\gamma_k\gamma_k}$. In other words, β_k is defined by requiring that the identity

$$P_{\beta_k} \mathrm{vec}_{E'}(A) = \mathrm{vec}(A_{\gamma_k\gamma_k}), \tag{12.6}$$

holds for all $A \in \mathbf{S}^p_{E'}$. With this vector notation, we can express the decomposition result of Theorem 9.2 as (12.5) with

$$\mathcal{C} = \{\mathrm{vec}_{E'}(X) \mid X \in \Pi_{E'}(\mathbf{S}^p_+)\}, \qquad \mathcal{C}_k = \{\mathrm{vec}(A) \mid A \in \mathbf{S}^{|\gamma_k|}_+\}.$$

Here $\mathcal{C}$ is the sparse matrix cone $\Pi_{E'}(\mathbf{S}^p_+)$ in vector form; the cones $\mathcal{C}_k$ are dense positive semidefinite cones in vector form. The cone of EDM-completable matrices in $\mathbf{S}^p_{E'}$ can be represented in a similar way, with

$$\mathcal{C} = \{\mathrm{vec}_{E'}(X) \mid X \in \Pi_E(\mathbf{S}^p_{\mathrm{d}})\}, \qquad \mathcal{C}_k = \{\mathrm{vec}(A) \mid A \in \mathbf{S}^{|\gamma_k|}_{\mathrm{d}}\}.$$

where $\mathbf{S}^r_{\mathrm{d}}$ denotes the set of Euclidean distance matrices of order r. Since we start from a chordal sparsity graph G', the interaction graph G defined by the index sets β_k in (12.6) is also chordal, with cliques β_k.

12.3 Decomposition

In this section we discuss some applications of partial separability in decomposition and splitting algorithms for convex optimization. We use the same notation as in the previous sections and define P as the stacked matrix

$$P = \begin{bmatrix} P_{\beta_1} \\ P_{\beta_2} \\ \vdots \\ P_{\beta_l} \end{bmatrix}. \tag{12.7}$$

Note that the matrix P^TP is diagonal with diagonal elements

$$(P^TP)_{ii} = |\{k \mid i \in \beta_k\}|, \tag{12.8}$$

the number of index sets β_k that contain the index i. We assume that each variable index appears in at least one index set, so $\mathbf{rank}(P) = n$.

We consider the problem of minimizing the partially separable function f given in (12.2), with convex elementary functions f_k. If the index sets β_k are disjoint, the problem is separable and reduces to l independent minimization problems that can be solved in parallel. To achieve a similar decomposition for the general problem one starts with the reformulation

$$\begin{array}{ll} \text{minimize} & \sum\limits_{k=1}^{l} f_k(\tilde{x}_k) \\ \text{subject to} & \tilde{x} = Px \end{array} \tag{12.9}$$

where $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_l) \in \mathbf{R}^{|\beta_1|} \times \mathbf{R}^{|\beta_2|} \times \cdots \times \mathbf{R}^{|\beta_l|}$ is a new variable. A variety of primal, dual, and primal-dual splitting algorithms can be used to exploit the separability of the objective function in (12.9). We discuss a primal application of the *Douglas-Rachford splitting algorithm* [151, 77] [21, section 25.2]. Problem (12.9) can be viewed as a minimization of the sum of two convex functions:

$$\text{minimize} \quad \tilde{f}(\tilde{x}) + \delta_{\mathcal{V}}(\tilde{x}) \tag{12.10}$$

where $\tilde{f}(\tilde{x}) = \sum_{k=1}^{l} f_k(\tilde{x}_k)$ and $\delta_{\mathcal{V}}$ is the indicator function of the subspace $\mathcal{V} = \operatorname{range}(P)$. The Douglas-Rachford algorithm applied to this problem is also known as the *method of partial inverses* [207, 208, 77, 182]. It alternates between two steps: the evaluation of the *proximal mapping* of the separable objective function $\tilde{f}$ and the Euclidean projection on the subspace $\mathcal{V}$. The proximal mapping of $\tilde{f}$ is the direct product of the proximal mappings of the l functions f_k and is defined as

$$\begin{aligned}\operatorname{prox}_{t\tilde{f}}(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_l) = \\ \Big(\operatorname{prox}_{tf_1}(\tilde{x}_1),\ \operatorname{prox}_{tf_2}(\tilde{x}_2),\ \ldots,\ \operatorname{prox}_{tf_l}(\tilde{x}_l)\Big),\end{aligned} \tag{12.11}$$

where $t > 0$ and

$$\operatorname{prox}_{tf_k}(\tilde{x}_k) = \underset{u}{\operatorname{argmin}} \left(f_k(u) + \frac{1}{2t}\|u - \tilde{x}_k\|_2^2 \right), \quad k = 1, \ldots, l.$$

If f_k is closed and convex, then the minimizer in the definition can be shown to exist and be unique for all values of $\tilde{x}_k$ [167]. For relatively simple functions f_k, the proximal mappings can be evaluated via closed-form expressions; see [55, 54, 181] for surveys. For more complicated functions, they can be computed by numerical optimization algorithms. In either case, it is important that the calculation of $\operatorname{prox}_{t\tilde{f}}$ in (12.11) involves l independent proximal operator evaluations that can be computed in parallel. The second step in each iteration of the Douglas-Rachford algorithm requires the Euclidean projection

$$\Pi_{\mathcal{V}}(\tilde{x}) = P(P^T P)^{-1} P^T \tilde{x} \tag{12.12}$$

of a vector $\tilde{x}$ on $\mathcal{V}$. The projection is a composition of two steps: the evaluation of the n-vector $y = (P^T P)^{-1} P^T \tilde{x}$, followed by the matrix-vector product Py. The first step is a straightforward componentwise averaging. The ith component of y can be expressed as

$$y_i = \frac{1}{|N_i|} \sum_{k \in N_i} (\tilde{x}_k)_{j_k}$$

where $N_i = \{k \mid i \in \beta_k\}$ and for each $k \in N_i$ we define $j_k = \beta_k^{-1}(i)$ (in other words, j_k is the index that satisfies $\beta_k(j_k) = i$). If we interpret the vectors $\tilde{x}_k$ as inaccurate copies of subvectors $P_{\beta_k} x$ of a vector x, then y is the estimate of x obtained by taking the average of all available copies of x_i. From the average vector y, we compute (12.12) as $\Pi_{\mathcal{V}}(\tilde{x}) = Py = (P_{\beta_1} y, P_{\beta_2} y, \ldots, P_{\beta_l} y)$. This simply extracts the l subvectors y_{β_k} of y. For more details on the Douglas-Rachford method for solving (12.9) we refer the reader to [212], where the method was used in a decomposition algorithm for conic optimization with partially separable constraints.

An alternative to the primal Douglas-Rachford method is the *alternating direction method of multipliers* (ADMM) [96, 90, 77, 44]. This method is known to be equivalent to the Douglas-Rachford method applied to the dual of (12.10) [89]. The dual can be written as

$$\begin{array}{ll} \text{maximize} & -\sum\limits_{k=1}^{l} f_k^*(\tilde{s}_k) \\ \text{subject to} & P^T \tilde{s} = 0, \end{array} \qquad (12.13)$$

where f_k^* is the convex conjugate of f_k and $\tilde{s} = (\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_l)$ with $\tilde{s}_k \in \mathbf{R}^{|\beta_k|}$. In the dual problem we maximize a separable concave function over $\mathcal{V}^\perp$. The main computations in the ADMM algorithm are similar to the primal Douglas-Rachford algorithm, so we omit the details.

With additional assumptions on the functions f_k, several other types of first-order splitting algorithms can be applied to exploit the separability of the objective function in (12.9), including the dual proximal gradient method [220, 221] and the dual block coordinate ascent method [222]. Applications to symmetric sparse matrix nearness problems are discussed in [213].

So far we have not made any assumptions of chordal structure in our discussion of problem (12.9). Chordal structure in the interaction graph is useful because it makes it easy to derive a compact representation of the subspaces $\mathcal{V} = \text{range}(P)$ and $\mathcal{V}^\perp = \text{nullspace}(P^T)$. We will use

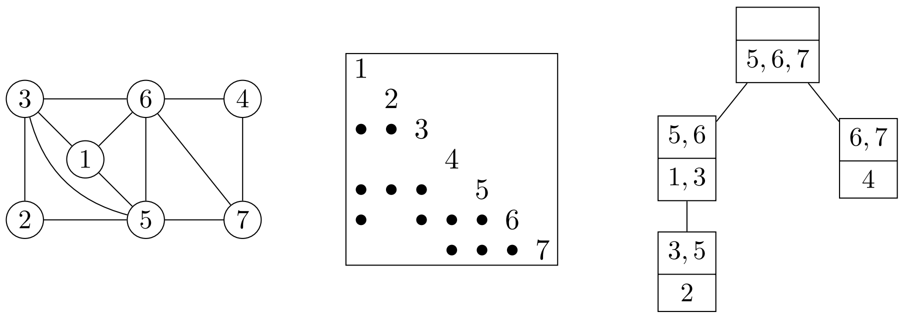


Figure 12.2: *Left and center.* Interaction graph for 7 variables and index sets $\beta_1 = (2,3,5)$, $\beta_2 = (1,3,5,6)$, $\beta_3 = (4,6,7)$, $\beta_4 = (5,6,7)$. *Right.* Clique tree.

the example in Figure 12.2 to explain the idea. The figure shows an interaction graph with 7 variables and index sets

$$\beta_1 = (2,3,5), \quad \beta_2 = (1,3,5,6), \quad \beta_3 = (4,6,7), \quad \beta_4 = (5,6,7).$$

The interaction graph is chordal with cliques β_1, β_2, β_3, β_4. The figure also shows a clique tree with the induced subtree property. The matrix P is given by

$$\begin{aligned} P^T &= \left[\begin{array}{cccc} P_{\beta_1}^T & P_{\beta_2}^T & P_{\beta_3}^T & P_{\beta_4}^T \end{array}\right] \\ &= \left[\begin{array}{ccc|cccc|ccc|ccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]. \end{aligned}$$

First consider the constraint $(\tilde{x}_1, \ldots, \tilde{x}_l) \in \mathcal{V} = \mathrm{range}(P)$. This is satisfied if there exists an x such that $\tilde{x}_k = P_{\beta_k} x$, $k = 1, \ldots, l$, *i.e.*, the l vectors $\tilde{x}_k$ are copies of different subvectors of one n-vector x. By the induced subtree property, the cliques β_k that contain a given index i form a subtree of the clique tree. To enforce the constraint that the different copies of $x_1, \ldots, x_n$ are all equal it is sufficient to enforce the equality between corresponding elements of vectors $\tilde{x}_k$ associated with adjacent cliques β_k in the induced subtree. For the example of Figure 12.2 this gives a set of 6 linear equations:

$$\left[\begin{array}{ccc|cccc|ccc|ccc} 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \end{array}\right] \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \end{bmatrix} = 0.$$

The general expression for these equations is

$$P_{\alpha_j}(P_{\beta_k}^T \tilde{x}_k - P_{\beta_j}^T \tilde{x}_j) = 0 \quad \forall j \in \operatorname{ch}(k), \quad k = 1, \ldots, l, \tag{12.14}$$

where $j \in \operatorname{ch}(k)$ means β_j is a child of β_k in the clique tree and α_j is defined as $\alpha_j = \beta_j \cap \beta_k$ for $j \in \operatorname{ch}(k)$, *i.e.*, the intersection of β_j and its parent clique in the clique tree. Each of the constraints in (12.14) is associated with an edge in the clique tree and involves only the two variables $\tilde{x}_j$ and $\tilde{x}_k$ associated with the adjacent cliques.

By transposition, one obtains a representation for $\mathcal{V}^\perp$. We have $\tilde{s} = (\tilde{s}_1, \ldots, \tilde{s}_l) \in \mathcal{V}^\perp$ if and only if

$$\tilde{s}_k = P_{\beta_k}(P_{\alpha_k}^T u_k - \sum_{j \in \operatorname{ch}(k)} P_{\alpha_j}^T u_j) \tag{12.15}$$

for some $u_j \in \mathbf{R}^{|\alpha_j|}$, $j = 1, \ldots, l$. Each parameter u_i can be associated with the edge in the clique tree that connects clique β_i to its parent. The equation (12.15) expresses $\tilde{s}_k$ as a combination of the parameters u_i associated with the edges that are incident to clique β_k. For the example of the figure, $(\tilde{s}_1, \tilde{s}_2, \tilde{s}_3, \tilde{s}_4) \in \mathcal{V}^\perp$ if and only if there exist $u_1, u_2, u_3 \in \mathbf{R}^2$ such that

$$\tilde{s}_1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_1, \qquad \tilde{s}_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_2 - \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} u_1,$$

and

$$\tilde{s}_3 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_3, \qquad \tilde{s}_4 = -\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} u_2 - \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} u_3.$$

13

Conic Optimization

In this chapter we discuss some applications of chordal sparse matrix techniques to conic linear optimization. Chapter 14 will deal with questions that are specific to semidefinite optimization. We use the notation

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{C} \end{array} \tag{13.1}$$

and

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c \\ & s \in \mathcal{C}^* \end{array} \tag{13.2}$$

for a general primal–dual pair of *conic linear programs* (conic LPs or cone LPs). The cone $\mathcal{C} \subset \mathbf{R}^n$ is a proper convex cone (closed, pointed, and with nonempty interior) and $\mathcal{C}^*$ is the dual cone

$$\mathcal{C}^* = \{z \in \mathbf{R}^n \mid x^T z \geq 0 \; \forall x \in \mathcal{C}\}.$$

The optimization variables in (13.1) and (13.2) are $x, s \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$. Standard linear programming is a special case with $\mathcal{C} = \mathbf{R}^n_+$.

Conic linear optimization with non-polyhedral cones has been popular in convex optimization since Nesterov and Nemirovski used it as a framework to extend interior-point methods for linear programming to nonlinear convex optimization [175]. Much of the research since the early 1990s has focused on cones that can be expressed as direct products of lower-dimensional cones, *i.e.*,

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_l, \qquad \mathcal{C}^* = \mathcal{C}_1^* \times \mathcal{C}_2^* \times \cdots \times \mathcal{C}_l^*, \tag{13.3}$$

where each $\mathcal{C}_k$ belongs to a small set of basic cone types. The most intensely studied cones are the nonnegative orthant, the *second-order* or *quadratic cone*

$$\mathcal{Q}^r = \{(x, y) \in \mathbf{R}^r \times \mathbf{R} \mid \|x\|_2 \leq y\},$$

and the *positive semidefinite cone*,

$$\mathcal{S}^r = \{\mathrm{vec}(X) \mid X \in \mathbf{S}_+^r\}. \tag{13.4}$$

(Recall that $\mathrm{vec}(X)$ denotes the $r(r+1)/2$-vector containing the elements of the symmetric matrix X in a format that preserves the standard inner products of vectors and matrices: $\mathrm{vec}(X)^T\mathrm{vec}(Y) = \mathbf{tr}(XY)$ for all $X, Y \in \mathbf{S}^r$.) These three cones are self-dual, but also *symmetric* and this property can be used to develop interior-point methods with primal-dual symmetry [79, 176, 80, 211, 196]. Conic LPs with the three symmetric cones are surprisingly general and most convex optimization problems that arise in practice can be represented in this form. This observation forms the basis of the convex modeling packages CVX [60, 103], YALMIP [156], and CVXPY [72].

In addition to the vast research on the three symmetric cones, there has been a more limited amount of work on conic optimization with nonsymmetric cones. Efforts in this direction have mainly been concerned with the *exponential cone*, the *power cone* with irrational powers, and ℓ_p-norm cones with irrational p [95, 51, 173, 205]. These cones do not have exact representations by second-order cones or semidefinite cones, and are the most notable exceptions in modeling systems based on the second-order and semidefinite cones. Nonsymmetric conic formulations have also been proposed as an approach to avoid the high complexity of semidefinite formulations, which often require the introduction of a large number of auxiliary variables and constraints. An example of this will be discussed in §14.4.

While the algorithms of (standard) linear programming and conic linear optimization have much in common, they differ greatly when it comes to exploiting sparsity. For standard linear programming, the question of exploiting sparsity in the coefficient matrix A can be addressed with well-developed methods from numerical linear algebra (for example, in interior-point methods, the sparse Cholesky factorization of matrices ADA^T, where D is a positive diagonal matrix). For non-polyhedral conic linear optimization, exploiting sparsity is more difficult. Different types of sparse and non-sparse structure can be distinguished and a variety of techniques has been proposed. The following short overview will be limited to methods that involve chordal graphs and sparse Cholesky factorization.

13.1 Schur complement sparsity

Consider the primal conic LP with the product cone (13.3):

$$\begin{array}{ll} \text{minimize} & \sum\limits_{k=1}^{l} c_k^T x_k \\ \text{subject to} & \sum\limits_{k=1}^{l} A_{jk} x_k = b_j, \quad j = 1, \dots, m \\ & x_k \in \mathcal{C}_k, \quad k = 1, \dots, l. \end{array} \tag{13.5}$$

We partitioned the vectors x, c as

$$x = (x_1, x_2, \dots, x_l), \qquad c = (c_1, c_2, \dots, c_l),$$

with $x_k, c_k \in \mathbf{R}^{n_k}$, assuming $\mathcal{C}_k \subset \mathbf{R}^{n_k}$. The coefficient A_{jk} is a row vector of length n_k. Interior-point methods for solving the conic LP require at each iteration the solution of a linear equation of the form

$$\begin{bmatrix} D^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}, \tag{13.6}$$

where D is block diagonal with positive definite diagonal blocks $D_k \in \mathbf{S}^{n_k}$, $k = 1, \dots, l$. For example, in a primal barrier method D_k is the inverse Hessian of the barrier function of $\mathcal{C}_k$ at the current iterate x_k. In a dual barrier method D_k is the Hessian of the barrier function for the dual cone $\mathcal{C}_k^*$ at the kth block s_k of the dual variable $s = (s_1, s_2, \dots, s_l)$.

The equation (13.6) is usually solved by eliminating Δx and solving an equation $H\Delta y = ADr_1 - r_2$ with coefficient matrix

$$H = ADA^T \tag{13.7}$$

The matrix H is referred to as the *Schur complement matrix*.

The factorization of the Schur complement matrix is often the most expensive step in the interior-point algorithm. To describe its sparsity pattern we define index sets

$$\beta_k = \{j \in \{1, 2, \ldots, m\} \mid A_{jk} \neq 0\}, \quad k = 1, \ldots, l.$$

The set β_k contains the indices of the nonzero rows in the kth block column of A, *i.e.*, the indices of the equality constraints in which the variable x_k appears. The sparsity pattern of ADA^T is the graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, m\}$ and edge set E

$$E = \{\{i, j\} \mid \{i, j\} \in \beta_k \text{ for some } \beta_k\}.$$

An edge $\{i, j\} \in E$ indicates that at least one variable x_k appears in equalities i and j of the constraints of (13.5). Sparsity in the Schur complement system can also be interpreted as partial separability in the dual of problem (13.5),

$$\begin{array}{ll} \text{maximize} & \sum\limits_{j=1}^{m} b_j y_j \\ \text{subject to} & \sum\limits_{j=1}^{m} A_{jk}^T y_j + s_k = c_k, \quad k = 1, \ldots, l \\ & s_k \in \mathcal{C}_k^*, \quad k = 1, \ldots, l. \end{array}$$

The dual can be written as as an unconstrained problem

$$\text{maximize} \quad \sum_{j=1}^{m} b_j y_j + \sum_{k=1}^{n} \delta_{\mathcal{C}_k^*}(c_k - \sum_{j=1}^{m} A_{jk}^T y_j), \tag{13.8}$$

where $\delta_{\mathcal{C}_k^*}$ is the indicator function of $\mathcal{C}_k^*$. The objective function in (13.8) is a partially separable function, since the kth term in the second sum only depends on the variables y_j for $j \in \beta_k$. The term *correlative sparsity* is sometimes used to describe the sparsity pattern of the Schur complement matrix in a conic LP [130, 126].

If the dimensions of the cones $\mathcal{C}_k$ are large, the Schur complement matrix ADA^T is often much denser than AA^T. This is an important limitation in interior-point methods and other types of second-order methods (such as penalty and augmented Lagrangian methods) [134]. For large problems it may be necessary to solve the Schur complement equations or the 2×2-block system (13.6) via iterative algorithms [135, 218, 225], or to solve the conic LP by a first-order method. Several first-order algorithms for convex optimization are well suited to handle general sparse structure in A. These methods typically require projections on the primal or dual cones, and either use the matrix A only in matrix-vector products with A or A^T, or in the solution of linear equations with coefficients of the form $I + tAA^T$; see [120, 232, 233, 139, 178, 182, 178] for examples.

13.2 Conversion and decomposition

Sparsity in the coefficient matrix A is more likely to lead to a sparse Schur complement matrix (13.7) if the conic LP has many low-dimensional cones $\mathcal{C}_k$, *i.e.*, when the dimensions of the dense diagonal blocks D_k are small. Certain types of conic LPs with dense Schur complement equations can be reformulated to enhance the sparsity of the equations and speed up the solution by interior-point methods. Examples of this idea are the conversion methods for semidefinite optimization proposed in [87, 169, 126]. The reformulations can be interpreted as techniques for exploiting partial separability and are also useful in combination with decomposition and first-order algorithms, as described in §12.3.

Although the most important application of a partially separable cone is the cone of symmetric matrices with a positive semidefinite completion (to be discussed in Chapter 14), it is useful to discuss the implications of partial separability for a general conic LP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{C}. \end{array} \tag{13.9}$$

This simplifies the notation but also shows that partial separability is more generally applicable. Suppose the cone $\mathcal{C}$ in (13.9) is a partially separable cone (12.5), so the problem can be written as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & P_{\beta_k} x \in \mathcal{C}_k, \quad k = 1, \ldots, l. \end{array} \tag{13.10}$$

We make the following assumptions about the cones $\mathcal{C}_k$ and index sets β_k. The first assumption is that the union of the index sets β_k is equal to $\{1, 2, \ldots, n\}$. This implies that the stacked matrix P in (12.7) has rank n. The second assumption is that the l cones $\mathcal{C}_k$ are proper convex cones. Third, we assume that there exists an element $\bar{x}$ with $P_{\beta_k}\bar{x} \in \mathbf{int}\, \mathcal{C}_k$, $k = 1, \ldots, l$. These assumptions imply that $\mathcal{C}$ is a proper cone, with dual cone

$$\mathcal{C}^* = \{\sum_{k=1}^{l} P_{\beta_k}^T \tilde{s}_k \mid \tilde{s}_k \in \mathcal{C}_k^*,\ k = 1, \ldots, l\},$$

see [212, 213].

The conic LP (13.1) with the partially separable cone (12.5) can be reformulated as

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T \tilde{x} \\ \text{subject to} & \tilde{A}\tilde{x} = b \\ & \tilde{x} \in \tilde{\mathcal{C}} \\ & \tilde{x} \in \text{range}(P) \end{array} \tag{13.11}$$

where $\tilde{\mathcal{C}}$ is the direct product cone $\tilde{\mathcal{C}} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_l$ and the variable is

$$\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_l) \in \mathbf{R}^{|\beta_1|} \times \mathbf{R}^{|\beta_2|} \times \cdots \times \mathbf{R}^{|\beta_l|}.$$

To make the two problems (13.11) and (13.9) equivalent, the matrix $\tilde{A}$ and vector $\tilde{c}$ must satisfy $\tilde{A}P = A$ and $\tilde{c}^T P = c^T$. We can choose any matrix and vector of the form

$$\tilde{A} = AP^+ + Q, \qquad \tilde{c}^T = c^T P^+ + q^T$$

where $P^+ = (P^T P)^{-1} P^T$ is the pseudoinverse of P, and $QP = 0$, $q^T p = 0$. Although the number of variables and equality constraints in (13.11) is larger than in the original conic LP (13.9), we replaced the n-dimensional cone $\mathcal{C}$ with a product cone $\tilde{\mathcal{C}}$ composed of several lower-dimensional cones.

The components of $\tilde{x}$ can be interpreted as copies of the vector $\tilde{x}_k = P_{\beta_k} x$, and the range constraint in (13.11) guarantees that there exists such an x. If we write the range constraint as a sparse matrix equation $B\tilde{x} = 0$ (for example, the equations (12.14) if the interaction graph is chordal), the problem and its dual can be written as

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T \tilde{x} \\ \text{subject to} & \tilde{A}\tilde{x} = b \\ & B\tilde{x} = 0 \\ & \tilde{x} \in \tilde{\mathcal{C}}. \end{array} \tag{13.12}$$

and

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \tilde{A}^T y + B^T u + \tilde{s} = c \\ & \tilde{s} \in \tilde{\mathcal{C}}^*. \end{array} \tag{13.13}$$

If we now use an interior-point method to solve (13.12) and (13.13) the key step in each iteration is the solution of a linear equation with the Schur complement coefficient matrix

$$\left[\begin{array}{c} \tilde{A} \\ B \end{array} \right] \tilde{D} \left[\begin{array}{c} \tilde{A} \\ B \end{array} \right]^T, \tag{13.14}$$

where $\tilde{D}$ is block-diagonal with l diagonal blocks of size $|\beta_k|$, $k = 1, \ldots, l$. The Schur complement system of the reformulated conic LPs can be much larger than the Schur complement system ADA^T for the original problem (13.9), but it may be easier to solve when ADA^T is dense. The matrix $\tilde{D}$ is a block-diagonal matrix with small diagonal blocks and if $\tilde{A}$ is very sparse, the matrix (13.14) may have a useful sparsity pattern. When applied to semidefinite optimization this technique is known as the *clique-tree conversion method* [87, 169, 126]; see §14.2.

Applications of the reformulation (13.11) are not limited to interior-point methods. Suppose for example that for each row i in the constraint matrix A of (13.10) there exists an index set β_k that contains the column indices of all the nonzero coefficients A_{ij} in the row. Then the equality constraints $Ax = b$ can be written as

$$\tilde{A}_k P_{\beta_k} x = \tilde{b}_k, \quad k = 1, \ldots, l,$$

and the matrix $\tilde{A}$ in (13.11) can be chosen to be block-diagonal with diagonal blocks $\tilde{A}_k$. The reformulated problem (13.11) is therefore separable except for the coupling constraint $\tilde{x} \in \operatorname{range}(P)$. The Douglas-Rachford algorithm, ADMM, or other decomposition methods can be applied to exploit this structure, as we have seen in §12.3. Examples of applications to semidefinite optimization can be found in [161, 138, 63, 243, 212].

14

Sparse Semidefinite Optimization

In the final chapter we review applications of chordal graph techniques to semidefinite optimization and, more generally, conic LPs (13.5) that include one or more positive semidefinite cones (13.4) among the cones $\mathcal{C}_k$. In §13.1 we discussed how the sparsity pattern of A and the dimensions of the cones $\mathcal{C}_k$ determine the sparsity of the Schur complement system (13.7). The kth conic constraint contributes a term $A_{ik}D_kA_{jk}^T$ to the i, j element of the Schur complement matrix, and this term is nonzero if $A_{ik} \neq 0$ and $A_{jk} \neq 0$. If $\mathcal{C}_k$ is a high-dimensional positive semidefinite cone ($\mathcal{C}_k = \mathcal{S}^{n_k}$ with n_k large), then computing $A_{ik}D_kA_{jk}^T$ can be expensive. In this section we discuss how joint sparsity in the vectors c_k, A_{1k}, …, A_{mk} can be exploited to speed up the calculation of the contributions $A_{ik}D_kA_{jk}^T$ of the kth constraint to the Schur complement matrix. To simplify the notation, we assume there is a single positive semidefinite conic inequality ($l = 1$, $\mathcal{C}_1 = \mathcal{S}^n$). If we use matrix notation for the variable x, the problem can be expressed as a semidefinite program (SDP) in standard form notation

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(CX) \\ \text{subject to} & \mathbf{tr}(A_iX) = b_i, \quad i = 1, \ldots, m \\ & X \succeq 0, \end{array} \tag{14.1}$$

with optimization variable $X \in \mathbf{S}^n$. The dual SDP is

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \sum\limits_{i=1}^{m} y_i A_i + S = C \\ & S \succeq 0 \end{array} \tag{14.2}$$

and has variables $y \in \mathbf{R}^m$ and $S \in \mathbf{S}^n$.

We discuss the implications of a joint sparsity pattern of the coefficients C, A_1, ..., A_m. This is sometimes called *aggregate sparsity.* We say the SDP has the aggregate sparsity pattern E if

$$C, A_1, \ldots, A_m \in \mathbf{S}^n_E.$$

A simple but important example is block-diagonal structure. If the aggregate pattern is block-diagonal, the variables X and S can be restricted to be block-diagonal, and the problem is equivalent to an SDP with multiple smaller matrix inequality constraints. In this chapter, we will assume that the aggregate pattern is not block-diagonal and, as a consequence, the Schur complement matrix of the SDP is dense.

As in the previous chapter, we limit the discussion to algorithms in which chordal sparse matrix results are prominent. This excludes several important semidefinite optimization algorithms, for example, the spectral bundle method [115], Burer and Monteiro's low-rank factorization method [48], and augmented Lagrangian algorithms [134, 242].

14.1 Aggregate sparsity

Although aggregate sparsity is not always present in large sparse SDPs, it appears naturally in applications with underlying graph structure. Examples are semidefinite relaxations of combinatorial graph optimization problems [158, 98, 4, 97] and eigenvalue optimization problems for symmetric matrices associated with graphs [236, 42]. Semidefinite optimization has been used extensively for Euclidean distance geometry problems with network structure, with applications to network node localization [34, 35, 36, 129, 73] and machine learning [230, 231, 237]. The graph structure in these applications is often inherited by the SDP. This is also the case for recently developed semidefinite relaxations of the optimal power flow problem in electricity networks

[17, 119, 165, 11, 148, 159, 160]. General-purpose convex modeling systems are another important source of SDPs with aggregate sparsity [60, 103, 156, 72]. These systems rely on semidefinite representations of convex optimization problems that often involve large numbers of auxiliary variables and sparse constraints [175, 24]. For example, SDP formulations of problems involving matrix norms or robust quadratic optimization typically have a 'block-arrow' aggregate sparsity pattern [7].

When the aggregate sparsity pattern is dense or almost dense, it is sometimes possible to introduce or improve sparsity in the aggregate pattern by means of a variable transformation [87, section 6]. Fukuda *et al.* [87] give the example of the graph equipartition problem, which has sparse coefficient matrices in the objective and constraints, except for one dense equality constraint $\mathbf{1}^T X \mathbf{1} = 0$. A similar example is the SDP formulation of the maximum variance unfolding problem in [230]. To see how a variable transformation can be used to introduce aggregate sparsity, consider the primal SDP (14.1) with $A_m = \mathbf{1}\mathbf{1}^T$ and $b_m = 0$. The aggregate sparsity pattern of this SDP is completely dense. However, if $X \succeq 0$ and $\mathbf{1}^T X \mathbf{1} = 0$, then X can be expressed as $X = VYV^T$ where $Y \succeq 0$ and V is any $n \times (n-1)$-matrix with columns that span the orthogonal complement of $\mathbf{1}$. An example of a sparse basis matrix is

$$V = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & -1 \end{bmatrix}.$$

This observation allows us to eliminate the equality constraint $\mathbf{1}^T X \mathbf{1} = 0$ and reformulate the problem as

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}((V^T C V) Y) \\ \text{subject to} & \mathbf{tr}((V^T A_i V) Y) = b_i, \quad i = 1, \ldots, m-1 \\ & Y \succeq 0. \end{array}$$

If the basis matrix V is sparse, then the aggregate pattern of this SDP is not much denser than the aggregate pattern of C, A_1, ..., A_{m-1} in the original problem.

More generally, suppose the coefficient matrices C and A_i contain dense terms that can be expressed as low rank matrices with a common range:

$$C = C_\mathrm{s} + WB_0W^T, \qquad A_i = A_{\mathrm{s},i} + WB_iW^T, \quad i = 1, \dots, m$$

where C_s and $A_{\mathrm{s},i}$ are sparse, and W is an $n \times k$ matrix with orthonormal columns. Define $U = [V \; W]$ where V is a $n \times (n-k)$ matrix with columns that span the nullspace of W^T. Then a change of variables $X = UYU^T$ transforms the SDP into the equivalent problem

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(\tilde{C}Y) \\ \text{subject to} & \mathbf{tr}(\tilde{A}_iY) = b_i, \quad i = 1, \dots, m \\ & Y \succeq 0 \end{array}$$

with

$$\tilde{C} = U^TCU = \left[\begin{array}{cc} V^TC_\mathrm{s}V & V^TC_\mathrm{s}W \\ W^TC_\mathrm{s}V & W^TC_\mathrm{s}W + B_0 \end{array}\right]$$

and

$$\tilde{A}_i = U^TA_iU = \left[\begin{array}{cc} V^TA_{\mathrm{s},i}V & V^TA_{\mathrm{s},i}W \\ W^TA_{\mathrm{s},i}V & W^TA_{\mathrm{s},i}W + B_i \end{array}\right], \quad i = 1, \dots, m.$$

If V is a sparse basis matrix for the nullspace of W^T and $k \ll n$, then the matrices $\tilde{C}$, $\tilde{A}_i$ may be sufficiently sparse to have a useful sparse aggregate pattern. The problem of finding a sparse nullspace basis has been studied in many contexts, including nonlinear optimization where a sparse nullspace basis is of interest when eliminating linear equality constraints [187, 31, 52, 53, 93].

Aggregate sparsity implies that all dual feasible variables S in (14.2) have sparsity pattern E. We can therefore restrict the dual variable to the subspace $\mathbf{S}^n_E$ and write the dual problem as

$$\begin{array}{ll} \text{maximize} & b^Ty \\ \text{subject to} & \sum_{i=1}^m y_iA_i + S = C \\ & S \in \mathbf{S}^n_+ \cap \mathbf{S}^n_E. \end{array} \qquad (14.3)$$

The matrix variable X in the primal problem (14.1), on the other hand, is not necessarily sparse, and usually dense at the optimum. However, the cost function and equality constraints in (14.1) only involve the entries X_{ij} on the diagonal and for $\{i, j\} \in E$. The other entries are important only to ensure the matrix is positive semidefinite. The problem can therefore be viewed as a problem with a sparse matrix $\Pi_E(X)$ as variable, constrained to have a positive semidefinite completion:

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(CX) \\ \text{subject to} & \mathbf{tr}(A_iX) = b_i, \quad i = 1, \ldots, m \\ & X \in \Pi_E(\mathbf{S}^n_+). \end{array} \tag{14.4}$$

The problems (14.4) and (14.3) form a pair of primal and dual conic LPs, since the cones $\Pi_E(\mathbf{S}^n_+)$ and $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ are dual cones (see §10.1). We also note that, without loss of generality, the sparsity pattern E can be assumed to be chordal, since replacing a non-chordal pattern with any chordal extension of it gives an equivalent problem.

Interior-point methods that exploit aggregate sparsity can be roughly divided in two approaches. One can take an interior-point method for the general semidefinite program (14.1) and exploit aggregate sparsity as much as possible in the implementation of the algorithm. Or one can view the equivalent problem (14.4) as a conic LP with respect to the matrix cone $\Pi_E(\mathbf{S}^n_+)$, and solve it with an interior-point method for nonsymmetric conic optimization. These two approaches are discussed in §14.3 and §14.4, respectively. In the next section we first discuss how the partially separable structure of the cone $\Pi_E(\mathbf{S}^n_+)$ can be used to reformulate the problem as an equivalent SDP that may be easier to solve by general-purpose interior-point solvers than the original SDP (14.1).

14.2 Conversion methods

The use of chordal graph techniques in interior-point methods for semidefinite optimization was introduced in [87, 169]. One of the key contributions in these papers is the formulation of *conversion methods* or *clique decomposition methods* to replace an SDP with large matrix inequality constraints by an equivalent problem that may be easier to solve by interior-point methods. The idea is an example of the method described in §13.2 for converting a conic LP with a partially separable cone into an equivalent problem with several smaller-dimensional cones. If we use the vector notation for the matrix cone $\Pi_E(\mathbf{S}^n_+)$ described in §12.2, the reformulated problem can be written as (13.12). It can also be written directly in matrix form as follows. Suppose the aggregate pattern E is chordal with cliques $\gamma_1, \ldots, \gamma_l$. To reformulate (14.4), we introduce a separate variable $\tilde{X}_k$ for each submatrix $X_{\gamma_k\gamma_k}$, and write the problem as

$$\begin{array}{ll} \text{minimize} & \sum\limits_{k=1}^{l} \mathbf{tr}(\tilde{C}_k \tilde{X}_k) \\ \text{subject to} & \sum\limits_{k=1}^{l} \mathbf{tr}(\tilde{A}_{ik}\tilde{X}_k) = b_i, \quad i = 1, \ldots, m \\ & P_{\eta_j}(P_{\gamma_k}^T \tilde{X}_k P_{\gamma_k} - P_{\gamma_j}^T \tilde{X}_j P_{\gamma_j}) P_{\eta_j}^T = 0, \\ & \qquad\qquad\qquad \forall j \in \mathrm{ch}(k), \quad k = 1, \ldots, l \\ & \tilde{X}_k \succeq 0, \quad k = 1, \ldots, l. \end{array} \tag{14.5}$$

Here $\mathrm{ch}(k)$ denotes the set of children of clique γ_k in a clique tree with the induced subtree property, and η_j is the intersection of clique γ_j and its parent clique in the clique tree. The coefficients $\tilde{C}_k$ and $\tilde{A}_{ik}$ are matrices that satisfy

$$\sum_{k=1}^{l} \mathbf{tr}(\tilde{C}_k X_{\gamma_k\gamma_k}) = \mathbf{tr}(CX)$$

and

$$\sum_{k=1}^{l} \mathbf{tr}(\tilde{A}_{ik} X_{\gamma_k\gamma_k}) = \mathbf{tr}(A_i X), \quad i = 1, \ldots, m,$$

for all $X \in \mathbf{S}^n_E$. The second set of equality constraints in (14.5) forces the variables $\tilde{X}_k$ to be consistent with the interpretation that they are overlapping submatrices $X_{\gamma_k\gamma_k}$ of a matrix X. The reformulated problem (14.5) has more variables and constraints than the original SDP, but the single matrix inequality $X \succeq 0$ has been replaced by l smaller ones. If the Schur complement system for (14.5) is sparse the problem may be easier to solve than the original SDP.

As an example, suppose E is the pattern in Figure 12.1. The cliques are $\gamma_1 = (1,3,4)$, $\gamma_2 = (2,4)$, $\gamma_3 = (3,4,5)$, and $\gamma_4 = (5,6)$, so

$$X_{\gamma_1\gamma_1} = \begin{bmatrix} X_{11} & X_{13} & X_{14} \\ X_{31} & X_{33} & X_{34} \\ X_{41} & X_{43} & X_{44} \end{bmatrix}, \qquad X_{\gamma_2\gamma_2} = \begin{bmatrix} X_{22} & X_{24} \\ X_{42} & X_{44} \end{bmatrix},$$

and

$$X_{\gamma_3\gamma_3} = \begin{bmatrix} X_{33} & X_{34} & X_{35} \\ X_{43} & X_{44} & X_{45} \\ X_{53} & X_{54} & X_{55} \end{bmatrix}, \qquad X_{\gamma_4\gamma_4} = \begin{bmatrix} X_{55} & X_{56} \\ X_{65} & X_{66} \end{bmatrix}.$$

If we use the clique tree in the center of the figure, then $\eta_1 = (3,4)$, $\eta_2 = 4$, $\eta_3 = 5$. In the conversion method we introduce four variables $\tilde{X}_1$, $\tilde{X}_2$, $\tilde{X}_3$, $\tilde{X}_4$ for the four submatrices submatrices and impose the constraints

$$\begin{aligned} \begin{bmatrix} \tilde{X}_{1,22} & \tilde{X}_{1,23} \\ \tilde{X}_{1,32} & \tilde{X}_{1,33} \end{bmatrix} &= \begin{bmatrix} \tilde{X}_{3,11} & \tilde{X}_{3,12} \\ \tilde{X}_{3,21} & \tilde{X}_{3,22} \end{bmatrix}, \\ \tilde{X}_{2,22} &= \tilde{X}_{3,22}, \\ \tilde{X}_{3,33} &= \tilde{X}_{4,11} \end{aligned}$$

to ensure that they can be interpreted as $\tilde{X}_k = X_{\gamma_k\gamma_k}$, $k = 1,2,3,4$.

The MATLAB package SparseCoLO [86] provides an implementation of four variants of the conversion method, described in [126]. The conversion method is also supported by the Python library Chompack [14]. An interesting feature of the conversion method is that it can be used in combination with any semidefinite optimization algorithm. The principal drawback is the large number of equality constraints that are added to the m equality constraints in the original problem. If the reformulated problem is solved by an interior-point method, the Schur complement system in each iteration of the algorithm can be much larger than for the original problem, and this can easily offset the benefit of the increased sparsity.

In some applications one has the option of using an inexact conversion and omit some of the additional consistency constraints in the converted problem (14.5). When the SDP is a relaxation of a nonconvex optimization problem, omitting some of the constraints gives a weaker, but less expensive, relaxation. In [11] this idea was applied to semidefinite relaxations of an optimal power flow problem.

The clique decomposition results that underlie the conversion method are also important for first-order algorithms and splitting methods for semidefinite optimization [161, 138, 63, 243, 212].

14.3 Interior-point methods

In this section we discuss techniques for exploiting aggregate sparsity in interior-point methods for semidefinite optimization. The main computation in each iteration of an interior-point method for solving (14.1) and (14.2) is the calculation of primal and dual search directions ΔX, Δy, ΔS by solving a set of linear equations of the form

$$\begin{aligned} \mathbf{tr}(A_i \Delta X) &= r_i, \quad i = 1, \ldots, m, && (14.6) \\ \sum_{j=1}^{m} \Delta y_j A_j + \Delta S &= R_{\mathrm{d}}, && (14.7) \\ \Delta X + \mathcal{D}(\Delta S) &= R_{\mathrm{c}}. && (14.8) \end{aligned}$$

The right-hand sides r, R_{d}, R_{c} are residuals that depend on the values of the current iterates $\hat{X}$, $\hat{y}$, $\hat{S}$ in the algorithm. The mapping $\mathcal{D}$ is a positive definite linear mapping computed from the current value of $\hat{X}$ and/or $\hat{S}$, and is defined differently for different methods. The iterates $\hat{X}$ and $\hat{S}$ are strictly positive definite throughout the algorithm, but not necessarily feasible for the primal and dual SDPs.

A common way to solve (14.6)–(14.7) is to first eliminate ΔX and ΔS and form the *Schur complement equations* $H \Delta y = g$, where H is the positive definite matrix with entries

$$H_{ij} = \mathbf{tr}\left(A_i \mathcal{D}(A_j)\right), \quad i, j = 1, \ldots, m,$$

and $g_i = r_i - \mathbf{tr}\left(A_i(R_{\mathrm{c}} - \mathcal{D}(R_{\mathrm{d}}))\right)$, $i = 1, \ldots, m$. The various definitions of $\mathcal{D}$ all have in common that $\mathcal{D}(U)$ is a dense matrix when U is nonzero (unless the aggregate sparsity pattern is block-diagonal). As a consequence, the Schur complement matrix H is dense. The cost of forming and solving the Schur complement system dominates the per-iteration complexity of the interior-point method. We now review some the most common choices of $\mathcal{D}$ and comment on the effect of aggregate sparsity. For background on these definitions, we refer the reader to the surveys [216, 217, 171].

The earliest interior-point methods for semidefinite optimization used *primal* or *dual scaling*. In a primal scaling method, $\mathcal{D}(U) = \hat{X}U\hat{X}$. This is the inverse Hessian of the logarithmic barrier function $-\log\det X$ at $\hat{X}$. Hence, in a primal scaling method,

$$H_{ij} = \mathbf{tr}\left(A_i\hat{X}A_j\hat{X}\right). \tag{14.9}$$

In a dual scaling method, $\mathcal{D}(U) = \hat{S}^{-1}U\hat{S}^{-1}$, the Hessian of the barrier $-\log\det S$ at the current dual iterate $\hat{S}$. With this choice,

$$H_{ij} = \mathbf{tr}\left(A_i\hat{S}^{-1}A_j\hat{S}^{-1}\right). \tag{14.10}$$

More recent algorithms and most general-purpose semidefinite optimization solvers use search directions defined by a primal-dual scaling, in which $\mathcal{D}$ depends on $\hat{X}$ and $\hat{S}$ [41, 210, 224, 238, 10, 168]. Examples are the AHO direction [5], the HRVW/KSH/M direction [116, 132, 166], and the NT direction [176]. These directions were later unified as special cases of the Monteiro-Zhang family of search directions [166, 241]. Of these directions the NT and HRVW/KSH/M directions are the most popular ones. The NT direction is defined by using $\mathcal{D}(U) = WUW$ where W is the unique positive definite matrix that satisfies $W\hat{S}W = \hat{X}$. Hence, in methods based on the NT direction,

$$H_{ij} = \mathbf{tr}\left(A_iWA_jW\right). \tag{14.11}$$

The HRVW/KSH/M direction uses

$$\mathcal{D}(U) = \frac{1}{2}(\hat{X}U\hat{S}^{-1} + \hat{S}^{-1}U\hat{X}),$$

which gives

$$H_{ij} = \mathbf{tr}\left(A_i\hat{X}A_j\hat{S}^{-1}\right). \tag{14.12}$$

We now turn to the question of exploiting aggregate sparsity. In a problem with aggregate sparsity pattern E, the coefficients A_i all have sparsity pattern E. In many applications they are actually much sparser and each coefficient matrix may only have a few nonzero entries. On the other hand, the matrices $\hat{X}$, $\hat{S}^{-1}$, W in the various definitions of H_{ij} are completely dense. Evaluating these expressions therefore requires matrix-matrix products with dense matrices.

If we start the iteration with a dual feasible $\hat{S}$, then the dual iterate $\hat{S}$ will remain dual feasible throughout the iteration, and be positive definite and sparse with sparsity pattern E. It can therefore be factored with zero-fill, and the sparse Cholesky factorization can be used to compute the products with $\hat{S}^{-1}$. For example, if $\hat{S} = LL^T$, then the elements of H_{ij} in a dual scaling method (14.10) are inner products of scaled matrices $L^{-T}A_jL^{-1}$, which can be computed by forward and backward substitutions with the sparse Cholesky factors. This important advantage of dual scaling methods over the primal or primal-dual scaling methods was pointed out in [26] and applied in the DSDP solver [25]. The cost of computing the Schur complement matrix (14.10) can be further reduced if the coefficients A_i are low rank. This is quite common in sparse problems where each matrix has only a few nonzero entries [26].

In the formula (14.12) for the HRVW/KSH/M method the multiplication with $\hat{S}^{-1}$ can be performed using a sparse factorization of $\hat{S}$, but $\hat{X}$ is a dense matrix with a dense inverse. A variation of the method that avoids multiplications with $\hat{X}$ is the *completion-based primal–dual interior-point method* of Fukuda *et al.* [87]. In the completion-based method the primal iterate $\hat{X}$ in (14.12) is replaced with the maximum determinant positive definite completion of $\Pi_E(\hat{X})$. As explained in Chapter 10 the maximum determinant completion has an inverse with sparsity pattern E. Moreover, if E is chordal, the inverse factorization of the maximum determinant completion can be computed efficiently. When $\hat{X}$ in (14.12) is replaced with the maximum determinant completion, the sparse inverse factorization can be used to evaluate H_{ij} as in the dual scaling method. The method has been implemented in the SDPA-C solver [169].

Deng, Gu, and Overton [69] describe efficient implementations of the AHO and HRVW/KSH/M primal-dual methods for SDPs with band aggregate sparsity, by applying sequentially semiseparable representations of band matrices.

14.4 Nonsymmetric formulation

As an alternative to the approaches of the previous section, one can view a pair of primal and dual SDPs with aggregate sparsity pattern as a pair of conic linear optimization problems (14.4) and (14.3). The primal and dual variables X, S in these problems are sparse matrices in $\mathbf{S}^n_E$. The inequality in the primal problem is with respect to the cone $\Pi_E(\mathbf{S}^n_+)$ of sparse matrices with a positive semidefinite completion. The inequality in the dual problem is with respect to the sparse positive semidefinite matrix cone $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$. The cones are not self-dual so there are no methods with complete primal-dual symmetry, but the problems can still be solved by primal, dual, or nonsymmetric primal-dual interior-point methods [174, 173, 51, 205].

We can assume, without loss of generality, that the sparsity pattern E is chordal. This implies that the barrier functions $\phi : \mathbf{S}^n_E \to \mathbf{R}$ and $\phi_* : \mathbf{S}^n_E \to \mathbf{R}$,

$$\phi(S) = -\log\det S, \qquad \phi_*(X) = \sup_{S \succ 0} \left(-\,\mathbf{tr}(XS) - \phi(S)\right), \tag{14.13}$$

and their first and second derivatives, can be evaluated by the multifrontal algorithms described in §9.6 and §10.4 [9]. The functions (14.13) are *normal barriers* so a host of path-following and potential reduction methods with a well-developed convergence analysis based on self-concordance theory can be applied [175, 172, 190, 174]. These methods follow the central path for the pair of conic LPs, which is defined as the set of $X \in \Pi_E(\mathbf{S}^n_+)$, $y \in \mathbf{R}^m$, $S \in \mathbf{S}^n_+ \cap \mathbf{S}^n_E$ that satisfy the equations

$$\begin{aligned} \mathbf{tr}(A_iX) &= b_i, \quad i = 1, \ldots, m, && (14.14)\\ \sum_{j=1}^m y_jA_j + S &= C, && (14.15)\\ X + \mu\nabla\phi(S) &= 0. && (14.16) \end{aligned}$$

The parameter $\mu > 0$ parameterizes the central path. Equivalently,

$$\begin{aligned} \mathbf{tr}(A_iX) &= b_i, \quad i = 1, \ldots, m, && (14.17)\\ \sum_{j=1}^m y_jA_j + S &= C, && (14.18)\\ \mu\nabla\phi_*(X) + S &= 0. && (14.19) \end{aligned}$$

Burer [47] has developed a primal-dual interior-point method for the nonsymmetric pair of problems (14.4)-(14.3). The search directions in his method are computed by linearizing the central path equations (14.17)–(14.19) after first rewriting the third condition:

$$\begin{aligned} \mathbf{tr}(A_iX) &= b_i, \quad i=1,\dots,m, \\ \sum_{j=1}^{m} y_jA_j + S &= C, \\ -\sqrt{\mu}\mathcal{F}_2(X) + \mathcal{F}_1(S) &= 0. \end{aligned}$$

Here the operator $\mathcal{F}_1$ maps a matrix $S \in \mathbf{S}^n_{++} \cap \mathbf{S}^n_E$ to its sparse Cholesky factor; the operator $\mathcal{F}_2$ maps a matrix $X \in \Pi_E(\mathbf{S}^n_{++})$ to the Cholesky factor of $-\nabla\phi_*(X)$ (*i.e.*, the inverse of the maximum determinant positive definite completion of X). The linearized central path equations are of the form (14.6)–(14.8) with $\mathcal{D} = -\mathcal{F}_2'(\hat{X})^{-1} \circ \mathcal{F}_1'(\hat{S})$, where $\mathcal{F}_1'$ and $\mathcal{F}_2'$ denote the derivatives. It is shown that $\mathcal{D}$ is (nonsymmetric) positive definite in a neighborhood of the central path. Eliminating ΔX and ΔS yields a nonsymmetric Schur complement system. To avoid forming the Schur complement equations explicitly, Burer proposes to solve the equations iteratively using BiCGSTAB.

Srijuntongsiri and Vavasis [209] propose potential reduction methods based on the barrier functions (14.13), using automatic differentiation to evaluate the gradients and Hessians of the primal and dual barrier functions. Their method also avoids forming the Schur complement matrix, and solves the system by a conjugate gradient method.

Andersen *et al.* [8] describe implementations of primal and dual path-following methods that use the multifrontal algorithms described in §9.6 and §10.4 to evaluate the derivatives of the barrier functions (14.13). The Newton equations, or linearized central path equations, have the same form (14.6)–(14.8) as for SDP algorithms, with the crucial difference that both ΔS and ΔX are matrices in $\mathbf{S}^n_E$. The mapping $\mathcal{D}$ is either $\mathcal{D} = \nabla^2\phi(\hat{S})$ in a dual scaling method (based on linearizing (14.19)) or $\mathcal{D} = \nabla^2\phi_*(\hat{X})^{-1}$ in a primal scaling method (based on linearizing (14.16)). Variants of this approach use a factorization $\nabla^2\phi(\hat{S}) = \mathcal{R}^*_{\hat{S}} \circ \mathcal{R}_{\hat{S}}$ of the Hessian of ϕ at $\hat{S}$, where $\mathcal{R}_{\hat{S}}$ is a linear mapping from $\mathbf{S}^n_E$ to $\mathbf{S}^n_E$ and $\mathcal{R}^*_{\hat{S}}$ is the adjoint of $\mathcal{R}_{\hat{S}}$ (see §9.6). Using this factorization, the Schur complement matrix H for $\mathcal{D} = \nabla^2\phi(\hat{S})$ can be expressed as

$$H_{ij} = \mathbf{tr}(A_i\mathcal{D}(A_j)) = \mathbf{tr}(\mathcal{R}_{\hat{S}}(A_i)\mathcal{R}_{\hat{S}}(A_j)), \quad i,j = 1,\ldots,m.$$

A factorization H can be computed from a QR factorization of the matrix

$$\left[\begin{array}{cccc} \mathrm{vec}_E(\mathcal{R}_{\hat{S}}(A_1)) & \mathrm{vec}_E(\mathcal{R}_{\hat{S}}(A_2)) & \cdots & \mathrm{vec}_E(\mathcal{R}_{\hat{S}}(A_m)) \end{array}\right].$$

This approach is interesting from a numerical point of view because it factors H without explicitly forming it. A similar expression applies to a primal scaling method, when $\mathcal{D} = \nabla^2\phi_*(\hat{X})^{-1}$. Here we use the fact that $\nabla^2\phi_*(\hat{X})^{-1} = \nabla^2\phi(\widetilde{S})$, where $\widetilde{S}$ is the inverse of the maximum determinant completion of $\hat{X}$ (see §10.4), to factor $\mathcal{D}$ as $\mathcal{D} = \mathcal{R}^*_{\widetilde{S}} \circ \mathcal{R}_{\widetilde{S}}$. Burer's method can be viewed as based on the nonsymmetric scaling $\mathcal{D} = \mathcal{R}^*_{\widetilde{S}} \circ \mathcal{R}_{\hat{S}}$. An implementation of the primal and dual methods described in [8] is available in the Python package SMCP [14].

Acknowledgments

We thank Stephen Boyd and Yinyu Ye for encouraging us to write this survey, and Juan Carlos Ramirez and the reviewers for their detailed comments on an earlier draft.

The first author's research was supported by the NSF under grants DMS-1115963 and ECCS-1128817.

Notation

Graphs

$G = (V, E)$	Undirected graph (p. 245).
$G(W)$	Subgraph induced by a subset $W \subseteq V$ (p. 246).
$G_\sigma = (V, E, \sigma)$	Ordered undirected graph (p. 247).
$\mathrm{adj}(v)$	Neighborhood of vertex v (p. 247).
$\deg(v)$	Degree of vertex v (p. 247).
$\mathrm{adj}^+(v)$	Higher neighborhood of vertex v (p. 247).
$\mathrm{adj}^-(v)$	Lower neighborhood of vertex v (p. 247).
$\mathrm{col}(v)$	Closed higher neighborhood of vertex v (p. 248).
$\mathrm{row}(v)$	Closed lower neighborhood of vertex v (p. 248).
$\deg^+(v)$	Higher degree of vertex v (p. 247).
$\deg^-(v)$	Lower degree of vertex v (p. 247).
$\mathrm{ch}(v)$	Set of children of vertex v (p. 250).
$\mathrm{lev}(v)$	Depth of vertex v (p. 250).
$\mathrm{fdesc}(v)$	First descendant of vertex v (p. 251).
$\mathrm{lca}(v, w)$	Least common ancestor v and w (p. 251).

Chordal graphs

$p_\mathrm{c}(W)$	Parent of clique W in rooted clique tree (p. 268).
$\mathrm{sep}(W)$, $\mathrm{res}(W)$	Clique separator, clique residual (p. 268).
$p(v)$	Parent of vertex v in elimination tree (p. 278).

V^{c}	Set of clique representative vertices (p. 283).
$\mathrm{snd}(v)$	Supernode with representative v (pp. 283, 288).
$a(v)$	First ancestor of v (pp. 284, 290).
$q(v)$	Parent in supernodal elimination tree (p. 284).

Graph elimination

$G^*_\sigma = (V, E^*_\sigma, \sigma)$	Elimination graph of $G_\sigma = (V, E, \sigma)$ (p. 300).
$\mathrm{adj}^+_*(v)$	Higher neighborhood in G^*_σ (p. 303).
$\mathrm{adj}^-_*(v)$	Lower neighborhood in G^*_σ (p. 303).
$\mathrm{col}_*(v)$	Closed higher neighborhood in G^*_σ (p. 303).
$\mathrm{row}_*(v)$	Closed lower neighborhood in G^*_σ (p. 303).
$\deg^+_*(v)$, $\deg^-_*(v)$	Higher, lower degree in G^*_σ (p. 303).

Sparse symmetric matrices

$\mathbf{tr}(A)$	Trace of A.
A^+	Pseudoinverse of A.
$\mathbf{S}^n$	Set of symmetric matrices of order n (p. 330).
$\mathbf{S}^n_E$	Matrices in $\mathbf{S}^n$ with sparsity pattern E (p. 330).
$\mathrm{vec}(A)$, $\mathrm{vec}_E(A)$	Vectorized matrix $A \in \mathbf{S}^n$, $A \in \mathbf{S}^n_E$ (p. 385)
Π_E	Projection on $\mathbf{S}^n_E$ (p. 350).
P_β	Selection matrix defined by index set β (p. 331).
γ_i	Lower-triangular nonzeros in column i (p. 332).
η_i	Lower-triangular zeros in column i (p. 332).
ν_i	Index set with elements of $\mathrm{snd}(i)$ (p. 332).
α_i	Index set with elements of $\mathrm{col}(i) \setminus \mathrm{snd}(i)$ (p. 332).

Positive semidefinite matrices

$\mathbf{S}^n_+$	Positive semidefinite $n \times n$ matrices (p. 340).
$\mathbf{S}^n_{++}$	Positive definite $n \times n$ matrices (p. 340).
$A \succeq 0$, $A \succ 0$	A is positive semidefinite, A is positive definite.
$\mathcal{S}^n$	Vectorized positive semidefinite matrices (p. 392).
$\phi(S)$	Logarithmic barrier for $\mathbf{S}^n_+ \cap \mathbf{S}^n_E$ (p. 353).
$\phi_*(X)$	Logarithmic barrier for $\Pi_E(\mathbf{S}^n_+)$ (p. 365).
$d_{\mathrm{kl}}(U, V)$	Kullback-Leibler divergence (p. 366).

References

[1] J. Agler, J. W. Helton, S. McCullough, and L. Rodman. Positive semidefinite matrices with a given sparsity pattern. *Linear Algebra and Its Applications*, 107:101–149, 1988.

[2] S. M. Aji and J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):335–343, 2000.

[3] A. Alfakih and H. Wolkowicz. Matrix completion problems. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *Handbook of Semidefinite Programming. Theory, Algorithms, and Applications*, chapter 18, pages 533–545. Kluwer Academic Publishers, 2000.

[4] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, February 1995.

[5] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton. Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM J. on Optimization*, 8(3):746–768, 1998.

[6] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, Y. Robert, F.-H. Rouet, and B. Uçar. On computing inverse entries of a sparse matrix in an out-of-core environment. *SIAM Journal on Scientific Computing*, 34(4):A1975–A1999, 2012.

[7] M. Andersen, L. Vandenberghe, and J. Dahl. Linear matrix inequalities with chordal sparsity patterns and applications to robust quadratic optimization. In *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, 2010.

[8] M. S. Andersen, J. Dahl, and L. Vandenberghe. Implementation of non-symmetric interior-point methods for linear optimization over sparse matrix cones. *Mathematical Programming Computation*, 2:167–201, 2010.

[9] M. S. Andersen, J. Dahl, and L. Vandenberghe. Logarithmic barriers for sparse matrix cones. *Optimization Methods and Software*, 28(3):396–423, 2013.

[10] M. S. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: A Python package for convex optimization, version 1.1.7. Available at `cvxopt.org`, 2014.

[11] M. S. Andersen, A. Hansson, and L. Vandenberghe. Reduced-complexity semidefinite relaxations of optimal power flow problems. *IEEE Transactions on Power Systems*, 29:1855–1863, 2014.

[12] M. S. Andersen and L. Vandenberghe. Support vector machine training using matrix completion techniques. 2010.

[13] M. S. Andersen and L. Vandenberghe. *CHOMPACK: A Python Package for Chordal Matrix Computations, Version 2.1*, 2014. `cvxopt.github.io/chompack`.

[14] M. S. Andersen and L. Vandenberghe. *SMCP: Python Extension for Sparse Matrix Cone Programs, Version 0.4*, 2014. `cvxopt.github.io/smcp`.

[15] M. F. Anjos and J. B. Lasserre, editors. *Handbook on Semidefinite, Conic and Polynomial Optimization.* Springer US, 2012.

[16] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[17] X. Bai, H. Wei, K. Fujisawa, and Y. Wang. Semidefinite programming for optimal power flow problems. *International Journal of Electrical Power & Energy Systems*, 30(6):383–392, 2008.

[18] M. Bakonyi and C. R. Johnson. The Euclidian distance matrix completion problem. *SIAM Journal on Matrix Analysis and Applications*, 16(2):646–654, 1995.

[19] M. Bakonyi and H. J. Woerdeman. *Matrix Completions, Moments, and Sums of Hermitian Squares.* Princeton University Press, 2011.

[20] W. W. Barrett, C. R. Johnson, and M. Lundquist. Determinantal formulation for matrix completions associated with chordal graphs. *Linear Algebra and Appl.*, 121:265–289, 1989.

[21] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces.* Springer, 2011.

[22] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the Association for Computing Machinery*, 30(3):479–513, 1983.

[23] S. Bellavia, J. Gondzio, and B. Morini. A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems. *SIAM Journal on Scientific Computing*, 35(1):A192–A211, 2013.

[24] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization. Analysis, Algorithms, and Engineering Applications.* SIAM, 2001.

[25] S. J. Benson and Y. Ye. Algorithm 875: DSDP5—software for semidefinite programming. *ACM Trans. Math. Softw.*, 34(3):16:1–16:20, May 2008.

[26] S. J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10:443–461, 2000.

[27] C. Berge. Some classes of perfect graphs. In F. Harary, editor, *Graph Theory and Theoretical Physics*, pages 155–165. Academic Press, 1967.

[28] C. Berge and J. L. Ramírez Alfonsín. Origins and genesis. In J. L. Ramírez Alfonsín and B. A. Reed, editors, *Perfect Graphs*, pages 1–12. John Wiley & Sons, 2001.

[29] A. Berry, J. R. S. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004.

[30] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58:33–66, 2006.

[31] M. W. Berry, M. T. Heath, I. Kaneko, M. Lawo, R. J. Plemmons, and R. C. Ward. An algorithm to compute a sparse basis of the null space. *Numerische Mathematik*, 47(4):483–504, 1985.

[32] U. Bertelè and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[33] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, Massachusetts, second edition, 2000.

[34] P. Biswas, T.-C. Lian, T.-C. Wang, and Y. Ye. Semidefinite programming based algorithms for sensor network localization. *ACM Transactions on Sensor Networks*, 2(2):188–220, 2006.

[35] P. Biswas, T.-C. Liang, K.-C. Toh, Y. Ye, and T.-C. Wang. Semidefinite programming approaches for sensor network localization with noisy distance measurements. *IEEE Transactions on Automation Science and Engineering*, 3(4):360–371, 2006.

[36] P. Biswas and Y. Ye. A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization. In W. W. Hager, S.-J. Huang, P. M. Pardalos, and O. A. Prokopyev, editors, *Multiscale Optimization Methods and Applications*, volume 82 of *Nonconvex Optimization and Its Applications*, pages 69–94. Springer US, 2006.

[37] J. R. S. Blair, P. Heggerness, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theoretical Computer Science*, 250:125–141, 2001.

[38] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*. Springer-Verlag, 1993.

[39] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1-2):1–21, 1993.

[40] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[41] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

[42] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

[43] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, 1994.

[44] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[45] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes. A Survey.* Society for Industrial and Applied Mathematics, 1999.

[46] P. Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

[47] S. Burer. Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM Journal on Optimization*, 14(1):139–172, 2003.

[48] S. Burer and R. D. C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (Series B)*, 95(2), 2003.

[49] Y. E. Campbell and T. A. Davis. Computing the sparse inverse subset: an inverse multifrontal approach. Technical Report TR-95-021, Computer and Information Sciences Department, University of Florida, 1995.

[50] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications.* Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 1997.

[51] P. R. Chares. *Cones and Interior-Point Algorithms for Structured Convex Optimization Involving Powers and Exponentials.* PhD thesis, Université Catholique de Louvain, 2009.

[52] T. Coleman and A. Pothen. The null space problem I. Complexity. *SIAM J. on Algebraic Discrete Methods*, 7(4):527–537, 1986.

[53] T. Coleman and A. Pothen. The null space problem II. Algorithms. *SIAM J. on Algebraic Discrete Methods*, 8(4):544–563, 1987.

[54] P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, editors, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer, 2011.

[55] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200, 2005.

[56] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics.* Springer-Verlag, 1992.

[57] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods.* SIAM, 2000.

[58] R. G. Cowell, A. P. Dawid, S. L Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks.* Information Science and Statistics. Springer, 2007.

[59] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-Boolean programming revisited. *Discrete Applied Mathematics*, 29:171–185, 1990.

[60] CVX Research, Inc. *CVX: Matlab Software for Disciplined Convex Programming, version 2.0*, 2012.

[61] J. Dahl, L. Vandenberghe, and V. Roychowdhury. Covariance selection for non-chordal graphs via chordal embedding. *Optimization Methods and Software*, 23(4):501–520, 2008.

[62] Y. H. Dai and N. Yamashita. Analysis of sparse quasi-Newton updates with positive definite matrix completion. *Journal of the Operations Research Society of China*, 2(1):39–56, 2014.

[63] E. Dall'Anese, H. Zhu, and G. B. Giannakis. Distributed optimal power flow for smart microgrids. *IEEE Transactions on Smart Grid*, 4(3):1464–475, 2013.

[64] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[65] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM, 2006.

[66] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38:1–25, 2011.

[67] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[68] A. P. Dempster. Covariance selection. *Biometrics*, 28:157–175, 1972.

[69] Z. Deng, M. Gu, and M. L. Overton. Structured primal-dual interior-point methods for banded semidefinite programming. In I. Gohberg, editor, *Topics in Operator Theory*, volume 202 of *Operator Theory: Advances and Applications*, pages 111–141. Birkhäuser Basel, 2010.

[70] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, 1983.

[71] I. S. Dhillon and J. A. Tropp. Matrix nearness problems with Bregman divergences. *SIAM Journal on Matrix Analysis and Applications*, 29(4), 2007.

[72] S. Diamond, E. Chu, and S. Boyd. CVXPY: a Python-embedded modeling language for convex optimization, version 0.2. `cvxpy.org`, 2014.

[73] Y. Ding, N. Krislock, J. Qian, and H. Wolkowicz. Sensor network localization, Euclidean distance matrix completion, and graph realization. *Optimization and Engineering*, 11(1):45–66, 2010.

[74] G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1-2):71–76, 1961.

[75] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

[76] H. Dym and I. Gohberg. Extensions of band matrices with band inverses. *Linear Algebra and Appl.*, 36:1–24, 1981.

[77] J. Eckstein and D. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[78] A. M. Erisman and W. F. Tinney. On computing certain elements of the inverse of a sparse matrix. *Communications of the ACM*, 18(3):177–179, 1975.

[79] J. Faraut and A. Korányi. *Analysis on Symmetric Cones*. Oxford University Press, 1994.

[80] L. Faybusovich. Euclidean Jordan algebras and interior-point algorithms. *Positivity*, 1(4):331–357, 1997.

[81] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

[82] P. C. Fishburn. *Interval Orders and Interval Graphs*. John Wiley & Sons, 1985.

[83] R. Fletcher. A new variational result for quasi-Newton formulae. *SIAM J. on Optimization*, pages 18–21, 1991.

[84] R. Fletcher. An optimal positive definite update for sparse Hessian matrices. *SIAM Journal on Optimization*, 5(1):192–218, February 1995.

[85] A. B. Frakt, H. Lev-Ari, and A. S. Willsky. A generalized Levinson algorithm for covariance extension with application to multiscale autoregressive modeling. *IEEE Transactions on Information Theory*, 49(2):411–424, 2003.

[86] K. Fujisawa, S. Kim, M. Kojima, Y. Okamoto, and M. Yamashita. User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems. Technical Report B-453, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2009.

[87] M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: general framework. *SIAM Journal on Optimization*, 11:647–674, 2000.

[88] D. R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[89] D. Gabay. Applications of the method of multipliers to variational inequalities. In M. Fortin and R. Glowinski, editors, *Augmented Lagrangian methods: Applications to the numerical solution of boundary-value problems*, Studies in Mathematics and Its Applications, pages 299–331. North-Holland, 1983.

[90] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2:17–40, 1976.

[91] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[92] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory Series B*, 16:47–56, 1974.

[93] J. Gilbert and M. Heath. Computing a sparse basis for the null space. *SIAM J. on Algebraic Discrete Methods*, 8(3):446–459, 1987.

[94] J. R. Gilbert, E. G. Ng, and B. W. Peyton. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 15(4):1075–1091, 1994.

[95] F. Glineur and T. Terlaky. Conic formulations for ℓ_p-norm optimization. *Journal of Optimization Theory and Applications*, 122(2):285–307, 2004.

[96] R. Glowinski and A. Marrocco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle*, 9(2):41–76, 1975.

[97] M. Goemans and F. Rendl. Combinatorial optimization. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *Handbook of Semidefinite Programming*, chapter 12, pages 343–360. Kluwer Academic Publishers, 2000.

[98] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, 1995.

[99] G. H. Golub and R. J. Plemmons. Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition. *Linear Algebra and Its Applications*, 34(3):3–27, 1980.

[100] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.

[101] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Elsevier, second edition, 2004.

[102] J. Gondzio. Matrix-free interior point method. *Computational Optimization and Applications*, 51(2):457–480, 2012.

[103] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control (a tribute to M. Vidyasagar)*, pages 95–110. Springer, 2008.

[104] A. Griewank and Ph. L. Toint. On the unconstrained minimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312. Academic Press, 1982.

[105] A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:119–137, 1982.

[106] A. Griewank and Ph. L. Toint. On the existence of convex decompositions of partially separable functions. *Mathematical Programming*, 28:25–49, 1984.

[107] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Society for Industrial and Applied Mathematics, second edition, 2008.

[108] R. Grone, C. R. Johnson, E. M Sá, and H. Wolkowicz. Positive definite completions of partial Hermitian matrices. *Linear Algebra and Appl.*, 58:109–124, 1984.

[109] R. Grone, S. Pierce, and W. Watkins. Extremal correlation matrices. *Linear Algebra and Its Applications*, 134:63–70, 1990.

[110] M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. *Annals of Discrete Mathematics*, 21:325–356, 1984.

[111] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics.* Springer-Verlag, 1988.

[112] A. Hajnal and J. Surányi. Über die Auflösung von Graphen in vollständige Teilgraphen. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae — Sectio Mathematica*, 1:113–121, 1958.

[113] P. Heggernes. Minimal triangulation of graphs: a survey. *Discrete Mathematics*, 306:297–317, 2006.

[114] P. Heggernes and B. W. Peyton. Fast computation of minimal fill inside a given elimination ordering. *SIAM Journal on Matrix Analysis and Applications*, 30(4):1424–1444, 2008.

[115] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. Technical Report SC 97-37, Konrad-Zuse Zentrum fuer Informationstechnik Berlin, 1997.

[116] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. on Optimization*, 6(2):342–361, 1996.

[117] N. J. Higham. Analysis of the Cholesky decomposition of a semi-definite matrix. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 161–185. Oxford University Press, 1990.

[118] P. L. Hammer (Ivănescu) and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer-Verlag New York, 1968. With a preface by Richard Bellman.

[119] R. A. Jabr. Exploiting sparsity in SDP relaxations of the OPF problem. *IEEE Transactions on Power Systems*, 27(2):1138–1139, 2012.

[120] F. Jarre and F. Rendl. An augmented primal-dual method for linear conic programs. *SIAM Journal on Optimization*, 19(2):808–823, 2008.

[121] C. R. Johnson. Matrix completion problems: A survey. In C. R. Johnson, editor, *Matrix Theory and Applications*, volume 40 of *Proceedings of Symposia in Applied Mathematics*, pages 171–189. American Mathematical Society, 1990.

[122] N. Kakimura. A direct proof for the matrix decomposition of chordal-structured positive semidefinite matrices. *Linear Algebra and Its Applications*, 433:819–823, 2010.

[123] T. Kanamori and A. Ohara. A Bregman extension of quasi-Newton updates I: an information geometrical framework. *Optimization Methods and Software*, 28(1):96–123, 2013.

[124] T. Kanamori and A. Ohara. A Bregman extension of quasi-Newton updates II: analysis of robustness properties. *Journal of Computational and Applied Mathematics*, 253:104–122, 2013.

[125] S. Kim and M. Kojima. Exploiting sparsity in SDP relaxation of polynomial optimization problems. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, volume 166 of *International Series in Operations Research & Management Science*, chapter 18, pages 499–531. Springer, 2012.

[126] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *Mathematical Programming*, 129:33–68, 2011.

[127] S. Kim, M. Kojima, and P. Toint. Recognizing underlying sparsity in optimization. *Mathematical Programming*, 119:273–303, 2009.

[128] S. Kim, M. Kojima, and H. Waki. Generalized Lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems. *SIAM Journal on Optimization*, 15(3):697–719, 2005.

[129] S. Kim, M. Kojima, and H. Waki. Exploiting sparsity in SDP relaxations for sensor network localization. *SIAM Journal on Optimization*, 20(1):192–215, 2009.

[130] K. Kobayashi, S. Kim, and M. Kojima. Correlative sparsity in primal-dual interior-point methods for LP, SDP, and SOCP. *Applied Mathematics and Optimization*, 58(1):69–88, 2008.

[131] M. Kojima, S. Kim, and H. Waki. Sparsity in sums of squares of polynomials. *Mathematical Programming*, 103:45–62, 2005.

[132] M. Kojima, S. Shindoh, and S. Hara. Interior-point methods for the monotone linear complementarity problem in symmetric matrices. *SIAM J. on Optimization*, 7:86–125, February 1997.

[133] D. Koller and N. Friedman. *Probabilistic Graphical Models. Principles and Techniques.* MIT Press, 2009.

[134] M. Kočvara and M. Stingl. PENNON: a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 18(3):317–333, 2003.

[135] M. Kočvara and M. Stingl. On the solution of large-scale SDP problems by the modified barrier method using iterative solvers. *Mathematical Programming, Series B*, 109(2-3):413–444, 2007.

[136] N. Krislock and H. Wolkowicz. Euclidean distance matrices and applications. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, chapter 30, pages 879–914. Springer, 2012.

[137] S. Kullback. *Information Theory and Statistics*. Dover Publications, 1997. Originally published by John Wiley & Sons, 1959.

[138] A. Y. S. Lam, B. Zhang, and D. Tse. Distributed algorithms for optimal power flow problem, 2011. `arxiv.org/abs/1109.5229`.

[139] G. Lan, Z. Lu, and R. D. C. Monteiro. Primal-dual first-order methods with $o(1/\epsilon)$ iteration-complexity for cone programming. *Mathematical Programming*, 126(1):1–29, 2011.

[140] J. B. Lasserre. Convergent SDP-relaxations in polynomial optimization with sparsity. *SIAM J. on Optimization*, 17(3):822–843, 2006.

[141] M. Laurent. A tour d'horizon on positive semidefinite and Euclidean distance matrix completion problems. In P. M. Pardalos and H. Wolkowicz, editors, *Topics in Semidefinite and Interior-Point Methods*, volume 18 of *Fields Institute Communications*, pages 51–76. The American Mathematical Society, 1998.

[142] M. Laurent. Matrix completion problems. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, volume III, pages 221–229. Kluwer, 2001.

[143] M. Laurent and S. Poljak. On a positive semidefinite relaxation of the cut polytope. *Linear Algebra and Its Applications*, 223:439–461, 1995.

[144] M. Laurent and S. Poljak. On the facial structure of the set of correlation matrices. *SIAM Journal on Matrix Analysis and Applications*, 17(3):530–547, 1996.

[145] M. Laurent and A. Varvitsiotis. A new graph parameter related to bounded rank positive semidefinite matrix completions. *Mathematical Programming*, 145(1-2):291–325, 2014.

[146] S. L. Lauritzen. *Graphical Models*. Oxford University Press, Oxford, 1996.

[147] S. L. Lauritzen and F. V. Jensen. Local computation with valuations from a commutative semigroup. *Annals of Mathematics and Artificial Intelligence*, 21:51–69, 1997.

[148] J. Lavaei and S. H. Low. Zero duality gap in optimal power flow problem. *IEEE Trans. Power Systems*, 27(1):92–107, February 2012.

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

[150] J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1146–1173, 1989.

[151] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[152] J. W. H. Liu. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Transactions on Mathematical Software*, 12:127–148, 1986.

[153] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

[154] J. W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review*, 34:82–109, 1992.

[155] J. W. H. Liu, E. G. Ng, and B. W. Peyton. On finding supernodes for sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 14(1):242–252, 1993.

[156] J. Löfberg. *YALMIP : A Toolbox for Modeling and Optimization in MATLAB*, 2004.

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

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

[159] S. H. Low. Convex relaxation of optimal power flow—part I: Formulations and equivalence. *IEEE Transactions on Control of Network Systems*, 1(1):15–27, March 2014.

[160] S. H. Low. Convex relaxation of optimal power flow—part II: Exactness. *IEEE Transactions on Control of Network Systems*, 1(2):177–189, June 2014.

[161] Z. Lu, A. Nemirovski, and R. D.C. Monteiro. Large-scale semidefinite programming via a saddle point mirror-prox algorithm. *Mathematical Programming*, 109(2-3):211–237, 2007.

[162] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[163] N. V. R. Mahadev and U. N. Peled. *Threshold Graphs and Related Topics*. North-Holland, 1995.

[164] M. Mézard and A. Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.

[165] D. K. Mohlzahn, J. T. Holzer, B. C. Lesieutre, and C. L. DeMarco. Implementation of a large-scale optimal power flow solver based on semidefinite programming. *IEEE Transactions on Power Systems*, 28(4):3987–3998, 2013.

[166] R. D. C. Monteiro. Primal-dual path-following algorithms for semidefinite programming. *SIAM Journal on Optimization*, 7:663–678, 1997.

[167] J. J. Moreau. Proximité et dualité dans un espace hilbertien. *Bull. Math. Soc. France*, 93:273–299, 1965.

[168] MOSEK ApS. *The MOSEK Optimization Tools Manual. Version 6.0.*, 2010. Available from `www.mosek.com`.

[169] K. Nakata, K. Fujitsawa, M. Fukuda, M. Kojima, and K. Murota. Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical details. *Mathematical Programming Series B*, 95:303–327, 2003.

[170] H. Nelis, E. Deprettere, and P. Dewilde. Approximate inversion of positive definite matrices, specified on a multiple band. In *Proceedings SPIE*, volume 975, pages 48–58, 1988.

[171] A. S. Nemirovski and M. J. Todd. Interior-point methods for optimization. *Acta Numerica*, 17:191–234, 5 2008.

[172] Y. Nesterov. Squared functional systems and optimization problems. In J. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization Techniques*, pages 405–440. Kluwer Academic Publishers, 2000.

[173] Y. Nesterov. Towards non-symmetric conic optimization. *Optimization Methods and Software*, 27(4-5):893–917, 2012.

[174] Yu. Nesterov. Nonsymmetric potential-reduction methods for general cones. Technical Report 2006/34, CORE Discussion Paper, Université catholique de Louvain, 2006.

[175] Yu. Nesterov and A. Nemirovskii. *Interior-point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[176] Yu. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[177] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 2nd edition, 2006.

[178] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Operater splitting for conic optimization via homogeneous self-dual embedding, 2013. `arxiv.org/abs/1312.3039`.

[179] T. Ohtsuki. A fast algorithm for finding an optimal ordering for vertex elimination on a graph. *SIAM Journal on Computing*, 5(1):133–145, 1976.

[180] T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Mathematical Analysis and Applications*, 54:622–633, 1976.

[181] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[182] N. Parikh and S. Boyd. Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102, 2014.

[183] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, 1961.

[184] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[185] F. M. Q. Pereira and J. Palsberg. Register allocation via coloring of chordal graphs. In K. Yi, editor, *Programming Languages and Systems*, pages 315–329. Springer, 2005.

[186] B. W. Peyton. Minimal orderings revisited. *SIAM Journal on Matrix Analysis and Applications*, 23(1):271–294, 2001.

[187] A. Pothen. *Sparse Null Bases and Marriage Theorems.* PhD thesis, Cornell University, 1984.

[188] A. Pothen and C. Sun. Compact clique tree data structures in sparse matrix factorizations. In T. F. Coleman and Y. Li, editors, *Large-Scale Numerical Optimization*, pages 180–204. SIAM, 1990.

[189] J. L. Ramírez Alfonsín and B. A. Reed, editors. *Perfect Graphs.* Wiley, 2001.

[190] J. Renegar. *A Mathematical View of Interior-Point Methods in Convex Optimization.* SIAM, 2001.

[191] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[192] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, second edition, 1970.

[193] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[194] D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1974.

[195] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[196] S. H. Schmieta and F. Alizadeh. Extension of primal-dual interior point algorithms to symmetric cones. *Mathematical Programming*, 96:409–438, 2003.

[197] I. J. Schoenberg. Remarks to Maurice Fréchet's article "Sur la définition axiomatique d'une classe d'espaces vectoriels distanciés applicables vectoriellement sur l'espace de Hilbert". *Annals of Mathematics*, 36(3):724–732, 1935.

[198] I. J. Schoenberg. Metric spaces and positive definite functions. *Transactions of the American Mathematical Society*, 44(3):522–536, 1938.

[199] A. Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency.* Springer, 2003.

[200] L. K. Schubert. Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian. *Mathematics of Computation*, 24:27–30, 1970.

[201] G. Shafer and P. P. Shenoy. Local computation in hypertrees. Technical report, School of Business, University of Kansas, 1988.

[202] D. F. Shanno. On variable-metric methods for sparse Hessians. *Mathematics of Computation*, 34(150):499–514, 1980.

[203] C. E. Shannon. The zero error capacity of a noisy channel. *IEEE Transactions on Information Theory*, 2(3):8–19, 1956.

[204] D. R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Applied Mathematics*, 7:325–331, 1984.

[205] A. Skajaa and Y. Ye. A homogeneous interior-point algorithm for nonsymmetric convex conic optimization. *Mathematical Programming*, pages 1–32, 2014.

[206] R. L. Smith. The positive definite completion problem revisited. *Linear Algebra and Its Applications*, 429:1442–1452, 2008.

[207] J. E. Spingarn. Partial inverse of a monotone operator. *Applied Mathematics and Optimization*, 10:247–265, 1983.

[208] J. E. Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32:199–223, 1985.

[209] G. Srijuntongsiri and S. Vavasis. A fully sparse implementation of a primal-dual interior-point potential reduction method for semidefinite programming. 2004. `arXiv:cs/0412009`.

[210] J. F. Sturm. Using SEDUMI 1.02, a Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653, 1999.

[211] J. F. Sturm. Similarity and other spectral relations for symmetric cones. *Linear Algebra and Its Applications*, 312:135–154, 2000.

[212] Y. Sun, M. S. Andersen, and L. Vandenberghe. Decomposition in conic optimization with partially separable structure. *SIAM Journal on Optimization*, 24:873–897, 2014.

[213] Y. Sun and L. Vandenberghe. Decomposition methods for sparse matrix nearness problems. 2015.

[214] R. E. Tarjan. Applications of path compression on balanced trees. *Journal of the Association for Computing Machinery*, 26(4):690–715, 1979.

[215] R. E. 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(3):566–579, 1984.

[216] M. J. Todd. A study of search directions in primal-dual interior-point methods for semidefinite programming. *Optimization Methods and Software*, 11(1-4):1–46, 1999.

[217] M. J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.

[218] K.-C. Toh. Solving large scale semidefinite programs via an iterative solver on the augmented systems. *SIAM Journal on Optimization*, 14(3):670–698, 2003.

[219] P. L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation*, 31(140):954–961, 1977.

[220] P. Tseng. Further applications of a splitting algorithm to decomposition in variational inequalities and convex programming. *Mathematical Programming*, 48:249–263, 1990.

[221] P. Tseng. Applications of a splitting algorithm to decomposition in convex programming and variational inequalities. *SIAM Journal on Control and Optimization*, 29(1):119–138, 1991.

[222] P. Tseng. Dual coordinate ascent methods for non-strictly convex minimization. *Mathematical Programming*, 59:231–247, 1993.

[223] L. Tunçel. *Polyhedral and Semidefinite Programming Methods in Combinatorial Optimization.* The American Mathematical Society, 2010.

[224] R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming Series B*, 95:189–217, 2003.

[225] L. Vandenberghe and S. Boyd. A primal-dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming*, 69(1):205–236, July 1995.

[226] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[227] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inferencing. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[228] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. *SIAM Journal on Optimization*, 17(1):218–241, 2006.

[229] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto. Algoritm 883: SparsePOP—a sparse semidefinite programming relaxation of polynomial optimization problems. *ACM Transactions on Mathematical Software*, 35(2):15:1–15:13, 2008.

[230] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70:77–90, 2006.

[231] K. Q. Weinberger, F. Sha, Q. Zhu, and L. K. Saul. Graph Laplacian regularization for large-scale semidefinite programming. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1489–1496, Cambridge, MA, 2007. MIT Press.

[232] Z. Wen. *First-Order Methods for Semidefinite Programming.* PhD thesis, Columbia University, 2009.

[233] Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented Lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3-4):203–230, 2010.

[234] A. S. Willsky. Multiresolution Markov models for signal and image processing. *Proceedings of the IEEE*, 90(8):1396–1458, 2002.

[235] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming*, volume 27 of *International Series in Operations Research and Management Science.* Kluwer Academic Publishers, Boston, MA, 2000.

[236] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[237] L. Xiao, J. Sun, and S. Boyd. A duality view of spectral methods for dimensionality reduction. In *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*, 2006.

[238] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (Semidefinite Programming Algorithm 6.0). *Optimization Methods and Software*, 18(4):491–505, 2003.

[239] N. Yamashita. Sparse quasi-Newton updates with positive definite matrix completion. *Mathematical Programming, Series A*, 115(1):1–30, 2008.

[240] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[241] Y. Zhang. On extending some primal-dual interior-point algorithms from linear programming to semidefinite programming. *SIAM Journal on Optimization*, 8:365–386, 1998.

[242] X. Zhao, D. Sun, and K.-C. Toh. A Newton-CG augmented Lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, 20(4):1737–1765, 2010.

[243] H. Zhu and G. B. Giannakis. Power system nonlinear state estimation using distributed semidefinite programming. *IEEE Journal of Selected Topics in Signal Processing*, 8(6):1039–1050, 2014.