University of California
Los Angeles

Chordal Sparsity in Interior-Point Methods for Conic Optimization

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical Engineering

by

Martin Skovgaard Andersen

2011

The dissertation of Martin Skovgaard Andersen is approved.

Alan J. Laub

Luminita A. Vese

Kung Yao

Lieven Vandenberghe, Committee Chair

University of California, Los Angeles

2011

To my loving family

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AMD	approximate minimum degree
CC	completion classifier
EDM	Euclidean distance matrix
GP	Gaussian process
KKT	Karush-Kuhn-Tucker
LMI	linear matrix inequality
LP	linear program
MAP	maximum a posteriori
MCS	maximum cardinality search
MD	minimum degree
ND	nested dissection
PCA	principal component analysis
PCG	preconditioned conjugate gradients
QP	quadratic program
RBF	radial basis function
SDP	semidefinite program
SOC	second-order cone
SOCP	second-order cone program
SVM	support vector machine

Sets of matrices and vectors

$\mathbf{R}$	set of real numbers
$\mathbf{R}^n_+$	set of nonnegative real n-vectors
$\mathbf{R}^{m\times n}$	set of real $m \times n$ matrices
$\mathbf{S}^n$	set of symmetric matrices of order n
$\mathbf{S}^n_+$	set of positive semidefinite matrices of order n
$\mathbf{S}^n_{++}$	set of positive definite matrices of order n
$\mathbf{S}^n_V$	set of symmetric matrices of order n with sparsity pattern V
$\mathbf{S}^n_{V,+}$	set of positive semidefinite matrices in $\mathbf{S}^n_V$
$\mathbf{S}^n_{V,++}$	set of positive definite matrices in $\mathbf{S}^n_V$
$\mathbf{S}^n_{V,\mathrm{c}+}$	set of positive semidefinite completable matrices in $\mathbf{S}^n_V$
$\mathbf{S}^n_{V,\mathrm{c}++}$	set of positive definite completable matrices in $\mathbf{S}^n_V$

Symbols

$\mathbf{1}$	vector or matrix of ones (its size is context-dependent)
0	vector or matrix of zeros (its size is context-dependent)
A^T	transpose of the matrix A
$A^\dagger$	Moore-Penrose pseudo-inverse of the matrix A
$\langle \cdot, \cdot \rangle$	an inner product
$\\| \cdot \\|_2$	Euclidean norm of a vector, or the spectral norm of a matrix
$\\|A\\|_F$	Frobenius norm of the matrix A
$\mathbf{tr}(A)$	trace of the matrix A
$P_V(A)$	projection of the matrix A on the sparsity pattern V
$A \bullet B$	inner product between symmetric matrices A and B
$\mathcal{G}(A)$	sparsity graph associated with the sparse symmetric matrix A
$\mathbf{diag}(a)$	diagonal matrix with diagonal entries read from the vector a
$\mathbf{diag}(A)$	vector with elements read from the diagonal of A
$\mathbf{dom}\, f$	domain of the function f
$\mathbf{int}\, K$	interior of the set K
$a \succeq_K b$	generalized inequality ($a - b \in K$)
$a \succ_K b$	strict generalized inequality ($a - b \in \mathbf{int}\, K$)
$a \succeq b$	vector inequality: means that $a - b \in \mathbf{R}^n_+$
$A \succeq B$	matrix inequality: means that $A - B \in \mathbf{S}^n_+$ or $A - B \in \mathbf{S}^n_{V,+}$
$A \succeq_{\mathrm{c}} B$	matrix inequality: means that $A - B \in \mathbf{S}^n_{V,\mathrm{c}+}$
∇f	gradient of f: vector with first-order partial derivatives of f
$\nabla^2 f$	Hessian of f: matrix with second-order partial derivatives of f
$\mathcal{T}_{\mathrm{adj}}$	adjoint of the linear operator $\mathcal{T}$: satisfies $\langle \mathcal{T}(x), y \rangle = \langle x, \mathcal{T}_{\mathrm{adj}}(y) \rangle$

Acknowledgments

This dissertation is the culmination of five years of research and study at UCLA. I recognize that this would not have been possible without the help and support of many people.

First, I gratefully acknowledge the funding sources that made my doctoral research possible. My research was supported in part by the Denmark–America Foundation (GN Store Nord Scholarship), Marie & M. B. Richter's Fund, Berg Nielsen's Scholarship, Otto Mønsted Ph.D. Scholarship, Knud Højgaard's Fund, Reinholdt W. Jorck's Fund, The Augustinus Fund, Thomas B. Thrige's Fund, and the U.S. National Science Foundation grants ECS-0524663 and ECCS-0824003.

I am sincerely grateful to my advisor, Lieven Vandenberghe, who has been a great mentor to me in so many ways. Lieven has contributed immensely to my development as a scholar and as a teacher, and for that I cannot thank him enough. Without his guidance and inspiration, this dissertation would never have come together. It has been a great pleasure to work with and learn from Lieven.

My committee members, Alan Laub, Kung Yao, and Luminita Vese, also deserve a special thanks. Alan Laub in particular has been a great source of inspiration. He is a great teacher who brings both rigor and clarity to the classroom, and I thank him for providing me with a solid foundation in numerical linear algebra.

In addition to my committee members, I would like to thank Joachim Dahl at MOSEK. His efforts and invaluable assistance with the Chompack software package have been a tremendous help. I am very grateful to Lin Xiao and John Platt at Microsoft Research for the many helpful and inspiring conversations during a summer internship at Microsoft Research in 2010. I would also like to thank Anders Hansson at Linköping University for an insightful discussion and helpful suggestions during his short visit to UCLA in Spring 2010. Further, I give my special thanks to my former advisor, Søren Holdt Jensen at Aalborg University. My studies at UCLA simply would not have been possible without his thoughtful mentorship.

I wish to express my warm thanks to my colleagues and friends for their moral support and the many helpful technical discussions. I am especially grateful to John Lee, Zhang Liu, Shalom Ruben, and Jitkomut Songsiri for their friendship and the good times shared. I also want to thank the Electrical Engineering Department staff—and in particular Deeona, Mandy, Brenda, Gershwin, and Jose—for their assistance.

My wonderful family deserves my deepest gratitude for all their love and support. I thank my loving parents, Preben & Kirsten, who have always been there for me and encouraged me throughout all stages of my life. I thank my siblings, Nana & Jesper, for being supportive, kind, and patient. I thank Henrik and Alice & Tim for their kind and warm hospitality. Finally, I give my deepest expression of love and appreciation to Cassie, my amazing friend and *kæreste*: thank you for your unwavering love and friendship!

Vita

1980	Born, Silkeborg, Denmark.
2001–06	M.S., Electrical Engineering Institute of Electronic Systems Aalborg University, Denmark.
2007–09	Teaching Assistant Electrical Engineering Department University of California, Los Angeles (UCLA).
2007–11	Research Assistant Electrical Engineering Department University of California, Los Angeles (UCLA).

Publications

M. S. Andersen, J. Dahl, L. Vandenberghe. "Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones." *Mathematical Programming Computation*, **2**:167–201, 2010.

M. S. Andersen, J. Dahl, L. Vandenberghe. "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), Sept. 2010.

M. S. Andersen, J. Dahl, Z. Liu, L. Vandenberghe. "Interior-point methods for large-scale cone programming," to appear in: S. Sra, S. Nowozin, S. J. Wright (editors) *Optimization for Machine Learning*, MIT Press, 2011.

Abstract of the Dissertation

Chordal Sparsity in Interior-Point Methods for Conic Optimization

by

Martin Skovgaard Andersen

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2011

Professor Lieven Vandenberghe, Chair

Conic optimization is an extension of linear optimization in which the linear inequality constraints are replaced with vector inequalities defined by convex cones. The two most important examples, second-order cone programming and semidefinite programming, are used in a wide range of applications and form the basis of modeling tools for convex optimization. In fact, most convex constraints encountered in practice can be formulated as conic inequalities with respect to the second-order or semidefinite cone.

Second-order cone programs and semidefinite programs can be solved in polynomial time using interior-point methods. However, methods for exploiting structure in semidefinite programming problems are not as well-developed as those for linear optimization, and despite its powerful modeling capabilities, the semidefinite cone does not always provide an efficient representation of convex constraints. We therefore consider methods for linear cone programs with two types of sparse matrix cones: the cone of positive semidefinite matrices with a given chordal sparsity pattern and its dual cone, the cone of chordal sparse matrices that have a positive semidefinite completion. These cones include not only the nonnega-

tive orthant, the second-order cone, and the semidefinite cone as special cases, but also a number of useful lower-dimensional, intermediate cones which are generally not self-dual. Our results show that the sparse matrix cone approach is advantageous for many problems with sparse, chordal sparsity patterns, and for problems that have an efficient chordal embedding. For problems with band or block-arrow structure, for example, the cost of a single interior-point iteration grows only linearly in the order of the matrix variable instead of quadratically or worse for general-purpose semidefinite programming solvers.

We also explore other applications in nonlinear optimization. By combining interior-point methods and results from matrix completion theory, we formulate an approximation method for large, dense, quadratic programming problems that arise in support vector machine training. The basic idea is to replace the dense kernel matrix with the maximum determinant positive definite completion of a subset of the entries of the kernel matrix. The resulting approximate kernel matrix has a sparse inverse, and this property can be exploited to dramatically improve the efficiency of interior-point methods. If the sparsity pattern is chordal, the sparse inverse and its Cholesky factors are easily computed by efficient recursive algorithms. The completion technique can also be applied to other dense convex optimization problems arising in machine learning, for example, in Gaussian process classification.

CHAPTER 1

Introduction

1.1 Conic Optimization with Sparse Matrix Cones

Conic optimization (also called cone programming) is an extension of linear optimization in which the componentwise vector inequalities are replaced by linear inequalities with respect to nonpolyhedral convex cones. A cone linear program (cone LP or conic LP) is an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq_K b. \end{array} \tag{1.1}$$

The optimization variable is x, and the inequality $Ax \preceq_K b$ is a generalized inequality, which means that $b - Ax \in K$, where K is a closed, pointed, convex cone with nonempty interior. In linear programming, K is the nonnegative orthant and the cone inequality is interpreted as componentwise inequality. If K is a nonpolyhedral cone, the problem is substantially more general than an LP. In fact, any convex optimization problem in the common format

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \end{array} \tag{1.2}$$

where f_i are real-valued convex functions, can be formulated as a cone LP [NN94]. An important difference between (1.2) and the cone programming format (1.1) is the way nonlinearities are represented. In (1.2), the convex functions $f_i(x)$ can be both nonlinear and nondifferentiable whereas the cone LP (1.1) has a linear objective and the constraint is a linear generalized inequality. Cone programming can therefore be thought of as a natural extension of linear programming. It has been widely used in the convex optimization literature since the early 1990s because it provides a framework for studying extensions of interior-point methods from linear programming to convex optimization [NN94, §4], [Ren01, §3], [BV04].

Research on algorithms for cone programming has focused on cones that are the direct product of three basic cones: the nonnegative orthant, the second-order cone, and the positive semidefinite cone [AHO98, TTT98, Mon98, Stu99, Bor99a, Stu02, TTT03, YFK03, BY05]. The second-order cone in $\mathbf{R}^n$ is defined as points $(t, x) \in \mathbf{R} \times \mathbf{R}^{n-1}$ that satisfy $\|x\|_2 \leq t$, and the positive semidefinite cone $\mathbf{S}^n_+$ is the set of symmetric positive semidefinite matrices of order n. A problem of the form (1.1) is called a second-order cone program (SOCP) if the cone K is the direct product of second-order cones, and similarly, (1.1) is called a semidefinite program (SDP) if K is the direct product of positive semidefinite cones. LPs can be expressed as SOCPs, and SOCPs can be expressed as SDPs. The converse is not true in general, so SDPs are much more general than SOCPs and LPs. The focus on the nonnegative orthant, the second-order cone, and the positive semidefinite cone (and direct products of these) is motivated by the fact that these are the only interesting cones that are symmetric or self-scaled [Gul96, NT97, NT98]. This property is necessary in order to formulate completely symmetric primal–dual interior-point methods, i.e., methods that yield the same iterates if we switch the primal and the dual problem.

The modeling capabilities associated with the three basic cones are surprisingly powerful. With a few exceptions (for example, geometric programming), most nonlinear convex constraints encountered in practice can be expressed as second-order cone or semidefinite constraints [NN94, VB96, BN01, AG03, BV04]. We illustrate this with two examples. Consider the eigenvalue optimization problem

$$\text{minimize} \quad \lambda_{\max}(A(x)) \tag{1.3}$$

where $A(x) = A_0 + \sum_{i=1}^m x_i A_i$ and $A_i \in \mathbf{S}^n$. This is a nonlinear, nondifferentiable, convex problem, and it can be reformulated as an equivalent SDP

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & tI - A(x) \succeq 0 \end{array} \tag{1.4}$$

with an auxiliary variable $t \in \mathbf{R}$ [VB96]. (The inequality $X \succeq 0$ means that X is positive semidefinite.) As another example, consider the nonlinear matrix-fractional optimization problem

$$\text{minimize} \quad c^T A(x)^{-1} c \tag{1.5}$$

with $A(x)$ defined as above and with domain $\{x \mid A(x) \succ 0\}$. This nonlinear problem can be expressed as an SDP [BV04, p. 76]

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \begin{bmatrix} A(x) & c \\ c^T & t \end{bmatrix} \succeq 0. \end{array} \tag{1.6}$$

These examples illustrate just a few of many techniques that can be used to reformulate nonlinear convex problems as LPs, SOCPs, or SDPs. YALMIP [Lof04] and CVX [GB12], two modeling software packages for convex optimization, can convert a high-level problem description of a convex optimization problem into either an LP, an SOCP, or an SDP, and both packages rely on second-order cone and semidefinite programming solvers for the numerical solution.

The three cones form a hierarchy with three levels of generality, and there are large gaps between the linear algebra complexity associated with each level.

Methods for exploiting structure in LPs rely on highly developed sparse Cholesky factorization techniques [Wri97a, GS04], and modern LP solvers can handle a very large number of variables and constraints (i.e., linear inequalities). Second-order cone constraints are more general than linear inequality constraints, and hence also more costly. The techniques for exploiting structure in SOCPs are similar to those used for LPs, and the extra cost depends largely on the number of second-order cone constraints and the dimension of the cones [ART03, GS05]. The positive semidefinite cone is the most general of the three basic cones, and the techniques for exploiting structure in SDPs are not as well-developed as those for LPs and SOCPs. These large differences in linear algebra complexity sometimes lead to inefficiencies when convex optimization problems are converted to the canonical cone program format. For example, a Euclidean norm constraint $\|Ax+b\|_2 \leq t$ is equivalent to a linear matrix inequality (LMI) with arrow structure. An extension is a matrix norm constraint $\|A(x)+B\|_2 \leq t$, where $A(x) = x_1A_1 + \cdots + x_mA_m$ and $A_i \in \mathbf{R}^{p\times q}$. This constraint cannot be expressed as an SOC constraint, but it is equivalent to a $(p+q)\times(p+q)$ LMI

$$\begin{bmatrix} tI & (A(x)+B)^T \\ A(x)+B & tI \end{bmatrix} \succeq 0.$$

Block-arrow structure is also common in robust optimization [EL97, GI03]. Since the block-arrow constraint cannot be reduced to a second-order cone constraint, it must be handled via its SDP formulation. This makes problems with matrix norm constraints substantially more difficult to solve than problems with Euclidean norm constraints, even when q is small. We demonstrate the complexity gap with a simple Matlab experiment based on a CVX model with a single matrix

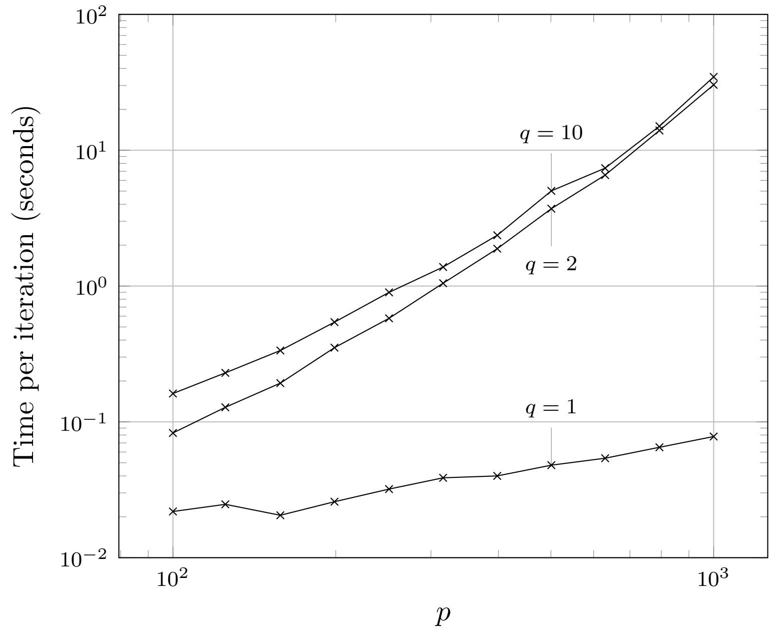


Figure 1.1: Time per iteration (seconds) required to solve (1.7) with CVX/SEDUMI for $m = 100$ and with different values of p and q where $A_i, B \in \mathbf{R}^{p \times q}$. The problem is formulated and solved as an SOCP when $q = 1$ (and $p \geq q$), and it is reformulated and solved as an SDP when $q \geq 2$.

norm constraint

$$\begin{array}{ll} \text{minimize} & c^T x + t \\ \text{subject to} & \|A(x) + B\|_2 \leq t \\ & l \preceq x \preceq u. \end{array} \tag{1.7}$$

CVX uses an SOCP formulation when $q = 1$ and an SDP formulation when $q \geq 2$. Fig. 1.1 shows the relationship between the parameter p and the average time per iteration for different values of q and with $m = 100$. The gap from $q = 1$ to $q = 2$ is quite significant, whereas the difference from $q = 2$ to $q = 10$ is relatively small. Furthermore, the time per iteration grows much faster as a function of p for $q \geq 2$. We give a similar example in Section 2.2 that illustrates the gap in complexity between linear programming and semidefinite programming.

For practical purposes, it is interesting to consider a wider family of cones that allow us to capture more closely structure in the problem. In this dissertation, we therefore consider cone programs defined by *chordal sparse matrix cones*, i.e., cones of positive semidefinite matrices with chordal sparsity patterns. In addition to the three basic cones (and direct products of them), the family of sparse chordal matrix cones also includes a wide range of useful intermediate cones. These cones are interesting because, as we will see in Section 2.4, there exist fast recursive algorithms for evaluating the function values and derivatives of the logarithmic barrier functions associated with these cones and their dual cones. This allows us to efficiently implement any primal or dual interior-point method. By replacing the three symmetric cones with the family of chordal sparse matrix cones, we gain in modeling flexibility and, potentially, efficiency. For matrix cones with band structure and block-arrow structure, the cost of one interior-point iteration can be reduced from being quadratic in the order of the matrix variable to being only linear. This allows us to solve problems with very large (but sparse) matrix variables on a standard desktop PC.

Another advantage of the chordal sparse matrix cones is that they belong to a lower-dimensional subspace. As a result, it is sometimes feasible to solve the Newton equations, which define the search direction in an interior-point method, without explicitly forming the normal equations. This improves the accuracy of the computed search direction when the Newton equations are ill-conditioned.

We will discuss sparse matrix cones and a set of chordal matrix techniques in Chapter 2, and in Chapter 3, we describe implementations of primal and dual interior-point methods for linear optimization with sparse matrix cone constraints.

1.2 Sparse Inverse Approximation

We also explore other applications of the chordal matrix techniques in nonlinear optimization. Chordal matrix techniques can be used to compute sparse inverse approximations of dense positive definite matrices. The maximum determinant positive definite completion of a partial matrix (if it exists) has a sparse inverse, and it can be computed efficiently when the underlying sparsity pattern is chordal. Similar techniques for computing sparse inverse approximations have been developed for preconditioning of iterative methods [BF82, KY93, BT99]. Unlike these methods, the completion-based sparse inverse approximations are guaranteed to be positive definite.

We make use of the completion-based sparse inverse approximation technique in a method for nonlinear support vector machine (SVM) training. SVMs are discriminative classifiers, and the training stage requires the solution of a constrained convex quadratic optimization problem where the Hessian of the quadratic objective is a (typically dense) kernel matrix. The order of the kernel matrix is equal to the number of data points, and computing the entire kernel matrix is therefore prohibitively expensive for large data sets. Massive data sets with many thousands – or even millions – of data points are increasingly common in many machine learning applications. In practice a subset of the data points is therefore often used to compute a low-complexity approximation of the full kernel matrix that is cheaper to compute, requires less storage, and makes the (approximate) training problem easier to solve. Research on kernel matrix approximation has focussed on randomized techniques for low-rank approximation [WS01, FS02, DM05, BJ05, KMT09a, KMT09b], and these methods generally work quite well for problems with a numerically rank-deficient kernel matrix.

In Chapter 4 we explore the use of matrix completion techniques as a means of computing sparse inverse approximations of partial positive kernel matrices. Using this technique, we implement and test an efficient interior-point method for approximate support vector machine training.

1.3 Notation

$\mathbf{S}^n$ is the set of symmetric matrices of order n. $\mathbf{S}^n_+ = \{X \in \mathbf{S}^n \mid X \succeq 0\}$ and $\mathbf{S}^n_{++} = \{X \in \mathbf{S}^n \mid X \succ 0\}$ are the sets of positive semidefinite, respectively, positive definite matrices of order n. The notation $S \bullet X = \mathbf{tr}(SX) = \sum_{i,j=1}^n S_{ij}X_{ij}$ denotes the standard inner product of symmetric matrices of order n.

A sparsity pattern of a symmetric matrix is defined by the set V of positions (i,j) where the matrix is allowed to be nonzero, i.e., $X \in \mathbf{S}^n$ has sparsity pattern V if $X_{ij} = X_{ji} = 0$ for $(i,j) \notin V$. It is assumed that all the diagonal entries are in V. Note that the entries X_{ij} for $(i,j) \in V$ are allowed to be zero as well. In that case we refer to them as *numerical zeros*, as opposed to the *structural zeros*. The number of nonzero elements in the lower triangle of V is denoted $|V|$.

$\mathbf{S}^n_V$ is the subspace of $\mathbf{S}^n$ of matrices with sparsity pattern V. $\mathbf{S}^n_{V,+}$ and $\mathbf{S}^n_{V,++}$ are the sets of positive semidefinite and positive definite matrices in $\mathbf{S}^n_V$. The projection Y of a matrix $X \in \mathbf{S}^n$ on the subspace $\mathbf{S}^n_V$ is denoted $Y = P_V(X)$, i.e., $Y_{ij} = X_{ij}$ if $(i,j) \in V$ and otherwise $Y_{ij} = 0$.

$\mathbf{S}^n_{V,\mathrm{c}+} = \{P_V(X) \mid X \succeq 0\}$ is the cone of matrices in $\mathbf{S}^n_V$ that have a positive semidefinite completion, and $\mathbf{S}^n_{V,\mathrm{c}++}$ is the interior of $\mathbf{S}^n_{V,\mathrm{c}+}$. The inequalities $\succeq_\mathrm{c}$ and $\succ_\mathrm{c}$ denote (strict) matrix inequality with respect to $\mathbf{S}^n_{V,\mathrm{c}+}$, i.e., $X \succeq_\mathrm{c} 0$ if and only if $X \in \mathbf{S}^n_{V,\mathrm{c}+}$ and $X \succ_\mathrm{c} 0$ if and only if $X \in \mathbf{S}^n_{V,\mathrm{c}++}$. The functions ϕ and ϕ_c are logarithmic barrier functions for $\mathbf{S}^n_{V,+}$ and $\mathbf{S}^n_{V,\mathrm{c}+}$, and are defined in Section 2.4 (equations (2.11) and (2.16)).

CHAPTER 2

Optimization with Sparse Matrix Cones

2.1 Cone Programs with Matrix Inequalities

Despite the progress made in the last fifteen years, exploiting sparsity in semidefinite programming remains an important challenge. A fundamental difficulty is that the variable X in the standard form SDP

$$\begin{array}{ll} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m, \\ & X \succeq 0 \end{array} \tag{2.1a}$$

is generally dense, even when the coefficients A_i, C are sparse with a common sparsity pattern. This complicates the implementation of primal and primal–dual interior-point methods. Exploiting sparsity in dual methods is more straightforward, because the slack variable S in the dual problem

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} y_i A_i + S = C \\ & S \succeq 0 \end{array} \tag{2.1b}$$

has the same (aggregate) sparsity pattern as C and the matrices A_i. However, computing the gradient and Hessian of the dual logarithmic barrier function requires the inverse of S, and S^{-1} is in general a dense matrix.

Fukuda and Nakata et al. [FKM00, NFF03], Burer [Bur03], and Srijuntongsiri and Vavasis [SV04] propose to pose the problems as optimization problems in the subspace of symmetric matrices with a given sparsity pattern. Specifically, assume that C, $A_1, \ldots A_m \in \mathbf{S}^n_V$, where $\mathbf{S}^n_V$ denotes the symmetric matrices of order n with sparsity pattern V. Then the pair of SDPs (2.1a–2.1b) is equivalent to a pair of *sparse matrix cone programs*

$$\begin{array}{ll} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m \\ & X \succeq_{\mathrm{c}} 0 \end{array} \tag{2.2a}$$

with primal variable $X \in \mathbf{S}^n_V$, and

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} y_i A_i + S = C \\ & S \succeq 0, \end{array} \tag{2.2b}$$

with dual variables $y \in \mathbf{R}^m$, $S \in \mathbf{S}^n_V$. The inequality $X \succeq_{\mathrm{c}} 0$ means that the sparse matrix X is in the cone $\mathbf{S}^n_{V,\mathrm{c}+}$ of matrices in $\mathbf{S}^n_V$ that have a positive semidefinite completion, i.e., X is the projection on $\mathbf{S}^n_V$ of a positive semidefinite matrix

$$X \succeq_{\mathrm{c}} 0 \quad \Leftrightarrow \quad X \in \mathbf{S}^n_{V,\mathrm{c}+} \equiv \{P_V(\bar{X}) \,|\, \bar{X} \in \mathbf{S}^n_+\}.$$

Similarly, the inequality $S \succeq 0$ in (2.2b) means that S is in the cone

$$\mathbf{S}^n_{V,+} \equiv \mathbf{S}^n_V \cap \mathbf{S}^n_+.$$

The equivalence between (2.2a–2.2b) and (2.1a–2.1b) is easily established: if X is optimal in (2.2a), then any positive semidefinite completion of X is optimal in (2.1a). Conversely, if X is optimal in (2.1a), then $P_V(X)$, the projection of X on $\mathbf{S}^n_V$, is optimal in (2.2a). An important difference between (2.2a–2.2b) and (2.1a–2.1b) is that different types of inequalities are used in the primal and dual problems of (2.2a–2.2b).

By formulating the SDPs (2.1a–2.1b) as optimization problems in $\mathbf{S}^n_V$, we achieve a dimension reduction from $n(n+1)/2$ (the dimension of the space $\mathbf{S}^n$ of symmetric matrices of order n) to $|V|$, the number of lower-triangular nonzeros in V. It is reasonable to expect that this can reduce the linear algebra complexity of interior-point methods for these problems. We will see that this is the case for methods based on primal or dual scaling, if the sparsity pattern V is *chordal*. The reduction in complexity follows from efficient methods for evaluating the primal and dual barrier functions for the problems (2.2a–2.2b), their gradients, and Hessians [DVR08].

In the remainder of this chapter we will first discuss sparse matrix cones and related work. This is followed by an introduction to chordal sparsity and an overview of efficient algorithms for evaluating primal and dual barriers and their derivatives for sparse matrix cones with chordal structure. We also discuss techniques for handling nonchordal sparsity patterns.

2.2 Sparse Matrix Cones

The optimization problems (2.2a–2.2b) are an example of a pair of primal and dual *cone programs*

$$\begin{array}{lll} \text{P:} & \text{minimize} & \langle c, x\rangle \\ & \text{subject to} & \mathcal{A}(x) = b \\ & & x \succeq_K 0 \end{array} \qquad \begin{array}{lll} \text{D:} & \text{maximize} & b^T y \\ & \text{subject to} & \mathcal{A}_{\text{adj}}(y) + s = c \\ & & s \succeq_{K^*} 0. \end{array} \tag{2.3}$$

The variables x, s in these problems are vectors in a vector space E, with inner product $\langle u, v\rangle$. The inequality $x \succeq_K 0$ means $x \in K$, where K is a closed, convex, pointed cone with nonempty interior. The inequality $s \succeq_{K^*} 0$ means that s is in the dual cone $K^* = \{s \mid \langle x, s\rangle \geq 0 \text{ for all } x \in K\}$. The mapping $\mathcal{A}$ in the primal equality constraint is a linear mapping from E to $\mathbf{R}^m$, and $\mathcal{A}_{\mathrm{adj}}$ is its adjoint, defined by the identity $u^T\mathcal{A}(v) = \langle \mathcal{A}_{\mathrm{adj}}(u), v\rangle$.

2.2.1 Nonsymmetric Sparse Matrix Cones

In the three-cone format used by SDP packages, any constraint that cannot be expressed as componentwise linear inequalities or second-order cone constraints must be converted to a semidefinite cone constraint. This has surprising consequences for the complexity of handling certain types of constraints.

Consider, for example, an SDP in which the coefficient matrices A_i, C are banded with bandwidth $2w + 1$. If $w = 0$ (diagonal matrices), the problem reduces to an LP and the cost of solving it by an interior-point method increases linearly in n. (For dense problems, the cost per iteration is $O(m^2n)$ operations.) If $w > 0$, the band SDP cannot be cast as an LP or an SOCP, and must be solved as an SDP. However, as Fig. 2.1 demonstrates, the cost per iteration of SDP solvers increases at least quadratically with n. This is surprising, because one would expect the complexity to be close to the complexity of an LP, i.e., linear in n for fixed m and w (see Section 3.5 for numerical experiments).

Band and block-arrow sparsity patterns are two examples of chordal structure. As the experiments in Section 3.5 will show, handling these constraints as nonsymmetric cone constraints results in a complexity per iteration that is linear in n, if the other dimensions are fixed. This provides one motivation for developing interior-point methods for the sparse matrix cone programs (2.2a–2.2b) with chordal sparsity patterns: the chordal sparse matrix cones form a family of useful convex cones that can be handled efficiently, without incurring the overhead of

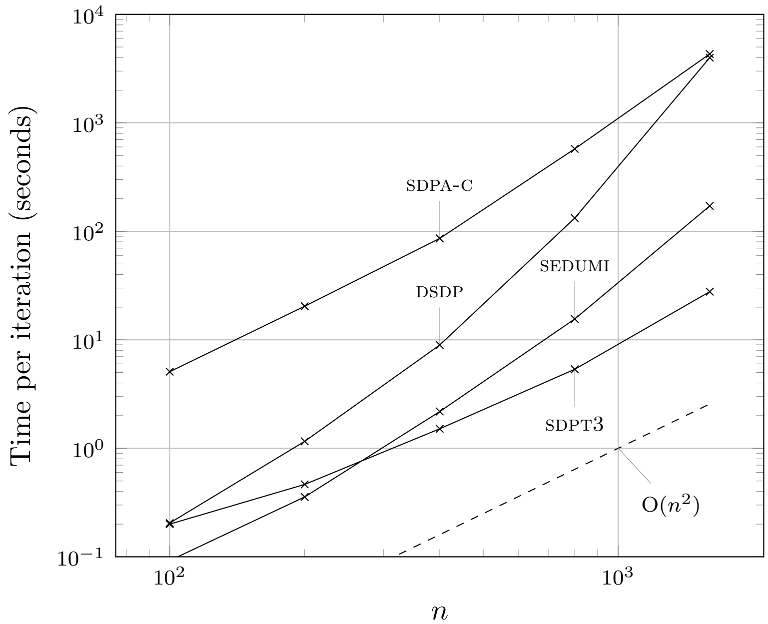


Figure 2.1: Time per iteration for SDPs with band structure as a function of n and with half-bandwidth $w = 5$ and $m = 100$ constraints. The complexity grows quadratically or worse.

the embedding in the positive semidefinite cone. Since block-diagonal combinations of chordal sparsity patterns are also chordal, a solver that efficiently handles a single matrix inequality with a general chordal pattern applies to a wide variety of convex optimization problems. Moreover, general (nonchordal) sparsity patterns can often be embedded in a chordal pattern by adding a relatively small number of nonzeros (see §2.5.1).

2.2.2 Related Work

Chordality is a fundamental property in sparse matrix theory, and its role in sparse semidefinite programming has been investigated by several authors. The first papers to point out the importance of chordal sparsity in semidefinite programming were by Fukuda et al. [FKM00] and Nakata et al. [NFF03]. Two techniques are proposed in these papers. The first technique exploits chordal sparsity to reformulate an SDP with a large matrix variable as a problem with several smaller diagonal blocks and additional equality constraints. This is often easier to solve using standard semidefinite programming algorithms. The second method is a primal–dual path-following method for the optimization problems (2.2a–2.2b). The algorithm uses the HRVW/KSH/M search direction for general semidefinite programming [HRV96, KSH97, Mon95], but applies it to the maximum determinant positive definite completion of the primal variable X. The authors show that the search direction can be computed without explicitly forming the completion of X. This method is outlined in §2.4.6, and it has been implemented in the SDPA-C package.

Burer's method [Bur03] is a nonsymmetric primal–dual path-following method for the pair of cone programs (2.2a–2.2b). It is based on a formulation of the central path equations in terms of the Cholesky factors of the dual variable S and the maximum determinant completion of the primal variable X. Linearizing the reformulated central path equations results in a new primal–dual search direction. The resulting algorithm is shown to have a polynomial-time worst-case complexity.

It is well known that positive definite matrices with chordal sparsity have a Cholesky factorization with zero fill-in. This provides a fast method for evaluating the standard logarithmic barrier function for the dual problem (2.2b), and via the chain rule, also for its gradient and Hessian. Srijuntongsiri and Vavasis [SV04] exploit this property in the computation of the dual Newton direction, by applying ideas from automatic differentiation in reverse mode. They also describe a fast algorithm for computing the primal barrier function (defined as the Legendre transform of the dual logarithmic barrier function) and its gradient and Hessian.

The algorithm is derived from explicit formulas for the maximum determinant positive definite completion [GJS84] which we discuss in §2.3.4.

Most semidefinite programming solvers also incorporate techniques for exploiting sparsity that are not directly related to chordal sparsity. For example, Fujisawa, Kojima, and Nakata [FKN97] present optimized methods for exploiting sparsity of the matrices A_i when computing the quantities $H_{ij} = A_i \bullet (UA_jV)$, $i, j = 1, \ldots, m$, with U, V dense. The matrix H is known as the Schur complement matrix, and its computation is a critical step in interior-point methods for semidefinite programming. Implementations of the techniques in [FKN97] are available in the SDPA and SDPT3 packages. Other recent work has focused on exploiting sparsity in specific classes of SDPs (notably, SDPs derived from sum-of-squares relaxations of polynomial optimization problems [WKK06]), and types of sparsity that ensure sparsity of the Schur complement matrix [KKK08]. For SDPs with band structure, it is possible to exploit so-called *sequentially semiseparable* structure [DGO10].

2.3 Chordal Sparsity

In this section, we introduce chordal graphs and matrices. We present some examples of classes of chordal graphs, and we describe important properties of chordal graphs and clique trees. We also discuss the maximum determinant positive definite completion problem which provides useful insights into some of the properties of the barrier function for the primal cone $\mathbf{S}^n_{V,\mathrm{c}+}$.

2.3.1 Chordal Graphs and Matrices

The sparsity pattern associated with a symmetric matrix X of order n can be represented by an undirected graph $\mathcal{G}(X)$ with nodes $1, \ldots, n$. The edges in $\mathcal{G}(X)$ correspond to the nonzero elements of X, i.e., $\mathcal{G}(X)$ has an edge between node i and node j $(i \neq j)$ if $X_{ij} \neq 0$. Although all the diagonal entries are assumed to be nonzero, the edges (i, i) are not included in the graph $\mathcal{G}(X)$.

Before we formally define chordal graphs and matrices, we will briefly review a few basic graph theoretic concepts (see e.g. [Gol04] or [Die10] for further details). A *path* of length k in a graph is a sequence of $k+1$ distinct and pair-wise adjacent vertices $v_0 v_1 \cdots v_k$, and a *cycle* is a path with no repeated vertices other than the end vertices (i.e., $v_0 = v_k$). A *chord* of a cycle is an edge that joins two nonadjacent vertices of the cycle. A *complete graph* is a graph in which every pair of distinct vertices is adjacent, i.e., any two vertices are connected by a unique edge. A *clique* W of a graph $\mathcal{G}$ is a subset of nodes such that the subgraph induced by W is a maximal complete subgraph[1] of $\mathcal{G}$. The cliques of a sparsity graph $\mathcal{G}(X)$ correspond to the (dense) maximal principal submatrices of X. The number of vertices in the largest clique of a graph is called the *clique number* of the graph. Thus, the clique number of $\mathcal{G}(X)$ corresponds to the order of the largest dense principal submatrix of the matrix X. A graph is said to be *connected* if there exists a path between any two of its vertices. A maximal connected subgraph of $\mathcal{G}$ is a *component* of $\mathcal{G}$. Components of the sparsity graph $\mathcal{G}(X)$ correspond to diagonal blocks of the matrix X after a symmetric permutation. In the rest of this section we assume for the sake of simplicity that $\mathcal{G}(X)$ is connected. However all algorithms extend in a straightforward manner to graphs consisting of several

[1]Some authors define a clique as any subset of nodes that induces a (not necessarily maximal) complete subgraph. Using this terminology, a maximal clique is a clique that is not included in any larger clique.

(a) Nonchordal graph.

(b) Chordal graph.

Figure 2.2: The graph (a) is nonchordal because there is a cycle of length four (1-2-3-4-1) without a chord. The graph (b) is chordal because all cycles of length four or greater have a chord.

connected chordal components.

We are now ready to define chordal graphs and matrices.

Definition 2.1. A graph $\mathcal{G}$ is *chordal* if every cycle of length at least four has a chord. We say that a symmetric matrix X is chordal if its sparsity graph $\mathcal{G}(X)$ is chordal.

Chordal graphs are also known as a *triangulated* graphs since all induced cycles (i.e., chordless cycles) are triangles. Fulkerson and Gross [FG65] use the term *rigid circuit graph*, and Lu et al. [LNM07] describe matrices with chordal structure as *well-structured* matrices. Fig. 2.2 shows an example of a nonchordal and a chordal graph. Notice that the graph in Fig. 2.2a is a composition of triangles, but the graph is nonchordal since there is an induced cycle of length four without a chord. The chordal graph in Fig. 2.2b can be obtained from the nonchordal graph in Fig. 2.2a by adding a chord to the chordless cycle, and hence the graph in Fig. 2.2b is a chordal embedding of the nonchordal graph in Fig. 2.2a. All induced subgraphs of a chordal graph are chordal, and hence all principal submatrices of a chordal matrix are chordal. Furthermore, a graph with several connected components is chordal if and only if each of its connected components is a chordal graph. The number of cliques in a chordal graph is at most n (with equality if and only if the graph has no edges) whereas the number of cliques in a nonchordal graph can grow exponentially in n [FG65, MM65].

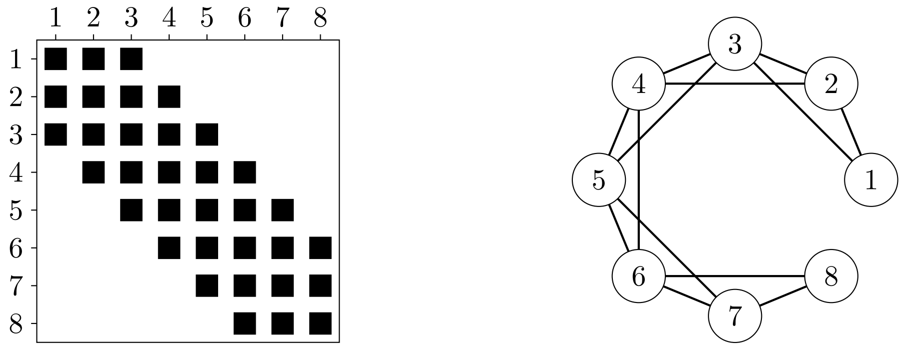
2.3.2 Examples of Chordal Sparsity

The class of chordal graphs includes a number of other classes of graphs. For example, interval graphs, split graphs, trees, k-trees, and complete graphs are all chordal graphs. We now briefly discuss these classes of graphs and their properties. We start with trees and k-trees which include as special cases the sparsity graphs of matrices with band and block-arrow sparsity.

Trees and k-trees It is easy to see that trees are chordal graphs since trees have no cycles. The cliques of a tree with two or more nodes therefore all consist of two nodes. The k-tree can be thought of as a generalization of a tree, and it can be defined recursively as follows [Ros74].

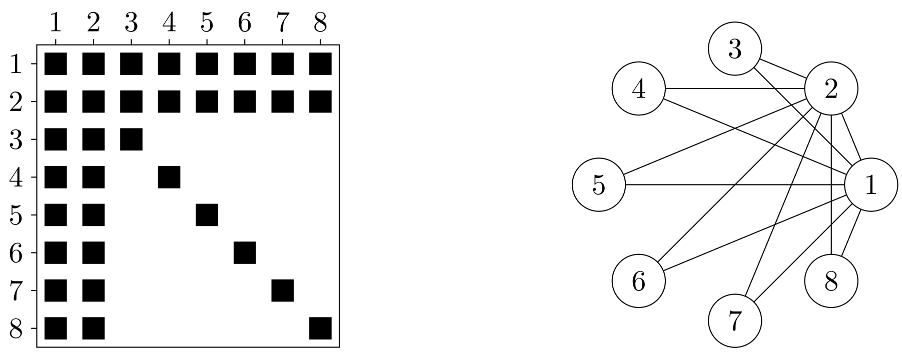
Definition 2.2. A complete graph on k vertices is a k-tree. A k-tree on $l+1$ vertices ($l \geq k$) can be constructed from a k-tree on l vertices by adding a vertex adjacent to exactly k vertices that induce a complete subgraph.

The definition implies that a k-tree has cliques of order at most $k+1$, and hence a 1-tree is simply a tree. The sparsity graphs associated with band matrices and block-arrow matrices are k-trees. In the band case, k is equivalent to the half-bandwidth, and in the block-arrow case, k is the block-width, i.e., the number of dense columns/rows in the block-arrow structure. Examples of band structure and block-arrow structure, as well as the corresponding sparsity graphs, are shown in Fig. 2.3 and Fig. 2.4, respectively. The block-arrow structure arises naturally in a number of applications, for example, in robust optimization. When $k=1$, the block-arrow structure reduces to arrow structure, and the associated sparsity



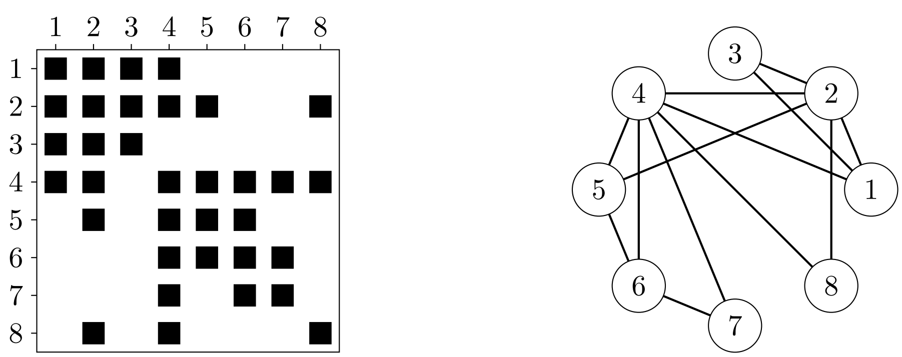
(a) (b)

Figure 2.3: Band sparsity pattern and sparsity graph.



(a) (b)

Figure 2.4: Block-arrow sparsity pattern and sparsity graph.



(a) (b)

Figure 2.5: Example of k-tree ($k = 2$) sparsity pattern and sparsity graph.

pattern is a so-called star graph which is a tree with one root and $n-1$ leaves if n is the number of nodes. Similarly, for band matrices, $k = 1$ corresponds to tridiagonal matrices for which the sparsity graph is a chain. An example of a more general k-tree is shown in Fig. 2.5. The recursive definition of k-trees can easily be verified. We start with a complete graph on the nodes 1 through k. Nodes are then added recursively such that an added node is adjacent to a complete subgraph on k nodes.

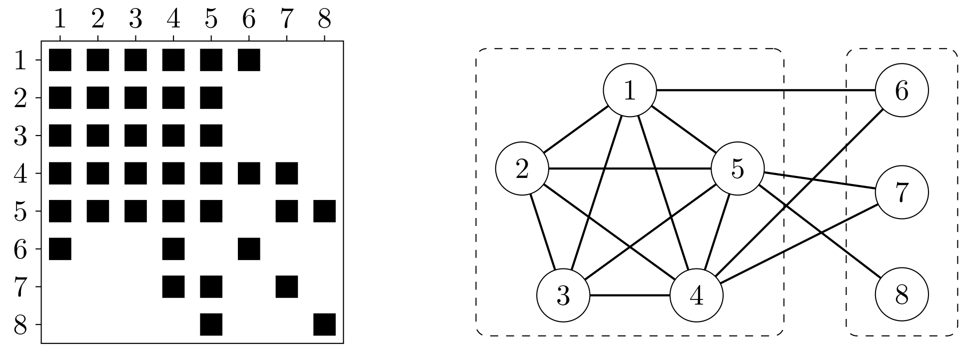
Interval graphs Interval graphs are defined in terms of a set of intervals on the real line. Each interval corresponds to a node in the interval graph, and there is an edge between two nodes if their corresponding intervals intersect [FG65]. Interval graphs are chordal graphs, and they arise naturally in certain scheduling and resource allocation problems.

Split graphs A split graph is a graph whose vertices can be split into two sets where one set induces a complete subgraph, and the other set is independent (i.e., a set of mutually nonadjacent nodes). In general, such a splitting need not be unique. The cliques in a split graph must naturally be of order $k+1$ or lower if k is the order of the complete subgraph in the splitting. A split graph is a special case of a partial k-tree,[2] and it arises in several applications. For example, the sparsity graph associated with a matrix of the form

$$A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix} \tag{2.4}$$

is clearly a split graph if B is a dense symmetric matrix and D is a diagonal matrix. Notice that A has block-arrow structure if C is dense. The structure in

[2] A *partial* k-tree contains all the vertices and a subset of the edges of a k-tree.



(a) (b)

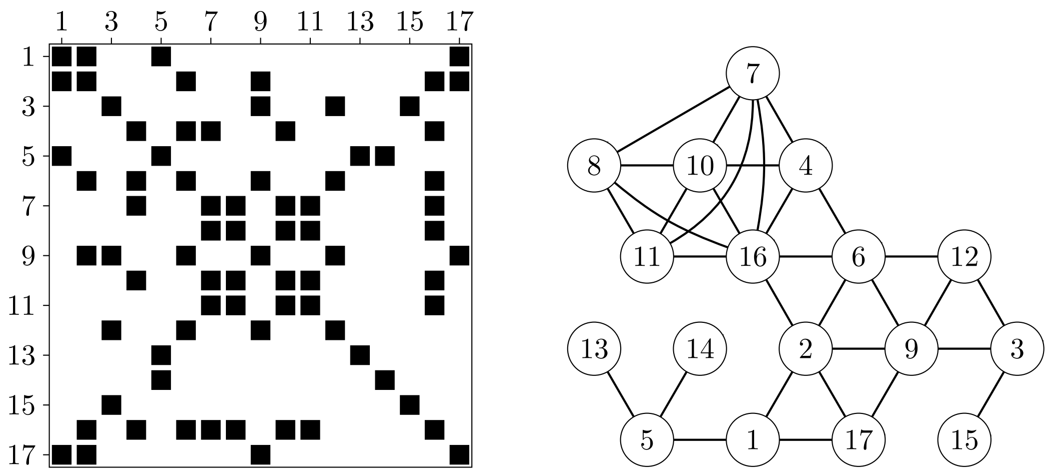
Figure 2.6: A split graph is a chordal graph that consists of a complete subgraph and an independent set of nodes.

(2.4) is common in barrier methods. For example, a barrier method for problems of the form

$$\begin{array}{ll} \text{minimize} & f(x) + \gamma \mathbf{1}^T u \\ \text{subject to} & -u \le x \le u \end{array}$$

yields a Hessian matrix of the form (2.4) with C and D diagonal, and $B = \nabla^2 f(x) + D$. It is easy to see that if both C and D are diagonal, then A is chordal (but not necessarily a split graph) if and only if B is chordal.

Powers of graphs The kth power of a graph $\mathcal{G}$ is a graph $\mathcal{G}^k$ with the same set of nodes as $\mathcal{G}$, and there is an edge between nodes i and j in $\mathcal{G}^k$ if and only if there is a path of length at most k between nodes i and j in $\mathcal{G}$. It can be shown that powers of a tree graph are chordal, i.e., if $\mathcal{G}$ is a tree (and hence chordal), then $\mathcal{G}^k$ is chordal for $k \ge 1$. The square of a general graph, however, is not necessarily chordal. Moreover, even when the square of a chordal graph is chordal, the corresponding sparsity pattern needs not be sparse. For example, the square of a star graph (which is a special case of a tree) is a complete graph. A detailed treatment of powers of chordal graphs can be found in [LS83].



(a) (b)

Figure 2.7: An example of a matrix with a general chordal sparsity pattern and the corresponding sparsity graph.

General chordal graphs General chordal graphs can be thought of as a generalization of interval graphs in that a chordal graph can be represented as the intersection graph of a family of subtrees of a tree [Gav74, Gol04]. An example of a general chordal sparsity pattern and its sparsity graph is shown in Fig. 2.7. In the next subsection we discuss how general chordal graphs can be represented in terms of a *clique tree*. This representation plays an important role in the formulation of efficient chordal matrix algorithms in Section 2.4.

2.3.3 Clique Trees

The cliques of a graph can be represented by a weighted undirected graph, with the cliques as its nodes, an edge between two cliques W_i and W_j ($i \neq j$) with a nonempty intersection, and a weight for the edge (W_i, W_j) equal to the cardinality of $W_i \cap W_j$. A clique tree is a maximum weight spanning tree of the clique graph. Clique trees of chordal graphs can be efficiently computed by the *maximum cardinality search* (MCS) algorithm [Ros70, RTL76, TY84]. The time and space complexity of the MCS algorithm is $O(n + e)$ if e is the number of edges in the graph, and hence this algorithm also provides an efficient test for chordality.

The useful properties of chordal sparsity patterns follow from a basic property known as the *running intersection property* [BP93]. Suppose a chordal graph $\mathcal{G}$ has l cliques $W_1, \ldots, W_l$, numbered so that W_1 is the root of a clique tree, and every parent in the tree has a lower index than its children. We will refer to this as a reverse topological ordering of the cliques. Define $U_1 = \emptyset$, $V_1 = W_1$, and, for $i = 2, \ldots, l$,

$$U_i = W_i \cap (W_1 \cap W_2 \cap \cdots \cap W_{i-1}), \quad V_i = W_i \setminus (W_1 \cup W_2 \cup \cdots \cup W_{i-1}). \quad (2.5)$$

The sets V_i are sometimes called the residuals, and the sets U_i the separators of the clique tree. The running intersection property then states that

$$U_i = W_i \cap W_{\mathrm{par}(i)}, \quad V_i = W_i \setminus W_{\mathrm{par}(i)}, \quad (2.6)$$

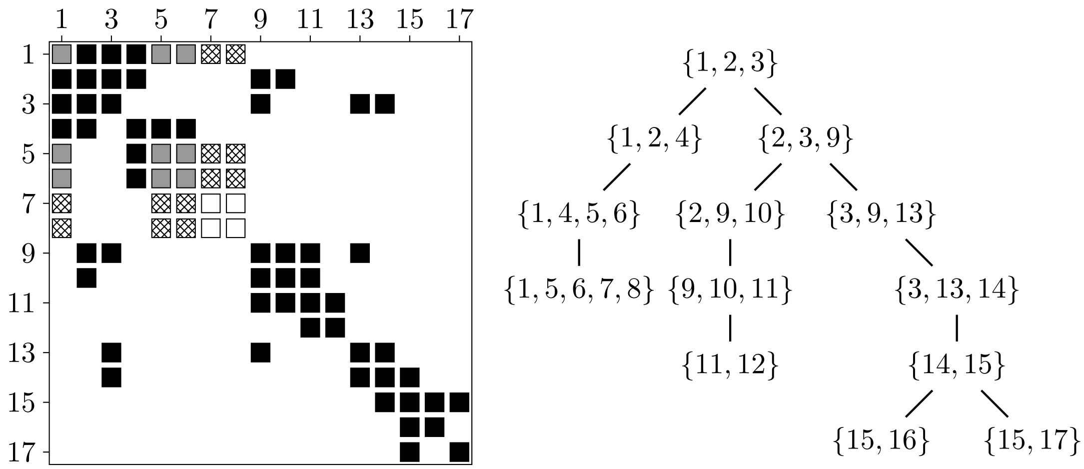
where $W_{\mathrm{par}(i)}$ is the parent of W_i in the clique tree. Fig. 2.8 shows an example of a chordal sparsity pattern and a clique tree that satisfies the running intersection property.

2.3.4 Maximum Determinant Positive Definite Completion

Chordal sparsity plays a fundamental role in the maximum determinant matrix completion problem: given a matrix $X \in \mathbf{S}^n_V$, find the positive definite solution $\bar{X} = Z^\star$ of the optimization problem

$$\begin{array}{ll} \text{maximize} & \log\det Z \\ \text{subject to} & P_V(Z) = X. \end{array} \quad (2.7)$$

A necessary condition for X to have a positive definite completion is clearly that all its fully specified submatrices (i.e., its cliques) are positive definite. We say



(a) Chordal sparsity pattern. (b) Clique tree.

Figure 2.8: The chordal sparsity pattern (a) is obtained by a symmetric permutation of the sparsity pattern in Fig. 2.7. The clique tree (b) satisfies the running intersection property. The left-most leaf in the clique tree, $W_i = \{1, 5, 6, 7, 8\}$, corresponds to the principal submatrix $X_{W_iW_i}$ in (a) and it can be decomposed into $X_{U_iU_i}$ (gray), $X_{V_iV_i}$ (white), and $X_{U_iV_i} = X^T_{V_iU_i}$ (hatched).

that such a matrix is *partial positive*. Partial positiveness of X, however, is not a sufficient condition for X to be positive completable, unless V is chordal.

Theorem 2.1. *Grone et al. [GJS84, Thm. 7]. The set of partial positive (nonnegative) matrices in $\mathbf{S}^n_V$ is equal to the set of positive (nonnegative) completable matrices in $\mathbf{S}^n_V$ if and only if V is chordal.*

Thus, if V is chordal and X is partial positive, then the solution $\bar{X} = Z^\star$ to (2.7) is unique and can be computed from X via closed-form expressions [GJS84, BJL89], [Lau96, page 146], [FKM00, §2], [NFF03], [Wer80, §3.2]. An equivalent algorithm for computing the Cholesky factor of $\bar{X}^{-1}$ is outlined in [DVR08].

The dual of the maximum determinant positive definite completion problem

(2.7) can be expressed as

$$\begin{array}{ll} \text{minimize} & X \bullet S - \log\det S - n \\ \text{subject to} & S_{ij} = 0, \ \forall (i,j) \notin V \end{array} \tag{2.8}$$

with variable S. It follows from convex duality that $\hat{S} = \bar{X}^{-1}$ is the unique minimizer of (2.8). This means that the maximum determinant positive definite completion $\bar{X}$ of a partial positive matrix X (if it exists) has a sparse inverse with the same sparsity pattern as X. Furthermore, $\hat{S}$ satisfies the nonlinear equation

$$P_V(\hat{S}^{-1}) = X. \tag{2.9}$$

The maximum determinant completion algorithm can therefore be interpreted as a method for solving the nonlinear equation (2.9) with variable $\hat{S} \in \mathbf{S}^n_V$.

For partial matrices with band structure, the inverse of $\bar{X}$ is also a band matrix, and hence $\bar{X}$ is a so-called *k-semiseparable* matrix. A symmetric matrix X is called k-semiseparable (where $k \geq 1$) if the maximum rank of all subblocks strictly below the kth superdiagonal of X is at most k (see e.g. [VBM07, p. 316]). An example of a 3-semiseparable matrix is shown in Fig. 2.9. Efficient algorithms for computations involving semiseparable matrices exist (see e.g. [VBM07, VBM09]).

Remark. Partial positive matrices with nonchordal sparsity patterns are not necessarily positive definite completable. For example, the nonchordal matrices

$$A = \begin{bmatrix} 3/2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 3/2 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 3/2 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 3/2 & 1 \\ -1 & 0 & 1 & 1 \end{bmatrix}$$

are both partial positive, but only A has a positive definite completion.

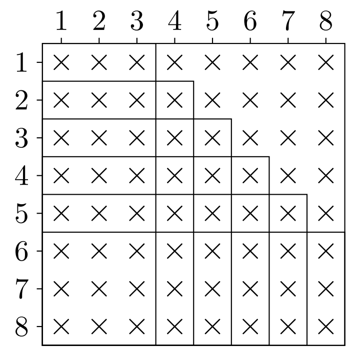


Figure 2.9: The structure of a semiseparable matrix with semiseparability rank 3: the submatrices below the third superdiagonal are of rank at most 3.

2.4 Chordal Matrix Algorithms

In this section we list a number of sparse matrix problems that can be solved by specialized methods if the underlying sparsity pattern is chordal. The algorithms consist of one or two recursions over the clique tree. Each recursion traverses the cliques in the tree in reverse topological order (starting at the root) or topological order (starting at the leaves). We omit the details of the algorithms, which can be found in [DVR08] and in Appendix A. A software implementation is available in the CHOMPACK library [DV09a].

2.4.1 Cholesky Factorization

Gauss elimination of a symmetric matrix X can be modelled as a so-called *elimination game* on the undirected adjacency graph $\mathcal{G}(X)$ [Par61]. The elimination game algorithm takes an elimination ordering α (i.e., a permutation of the vertex set $\{1, \ldots, n\}$) and returns a chordal graph.

Algorithm 2.1. Elimination game.

Input: a graph $\mathcal{G}$ with n vertices and an ordering α
  for $i = 1$ **to** n **do**
    Eliminate vertex v of $\mathcal{G}$ for which $\alpha(v) = i$.
    Pair-wise connect all uneliminated vertices that are adjacent to v in $\mathcal{G}$.
  end for

The added edges are called *fill edges* or *fill-in* and they correspond to the added nonzero entries in the factorization of X. It is easy to see that no fill edges are incurred at step i if and only if the vertex v is a *simplical* vertex, i.e., a vertex whose neighbors induce a complete subgraph. An ordering α is called a *perfect elimination ordering* if no fill edges are incurred by the elimination game. Fulkerson & Gross [FG65] showed that a graph has a perfect elimination ordering if and only if the graph is chordal. This implies that a positive definite matrix with a chordal sparsity pattern has a Cholesky factorization with zero fill-in [Ros70, BP93], i.e., if $S \in \mathbf{S}^n_{V,++}$, then there exists a permutation matrix P and a lower triangular matrix L such that

$$P^T S P = L L^T, \tag{2.10}$$

and $P(L+L^T)P^T$ has the sparsity pattern V. This is perhaps the most important property of chordal graphs and the basis of the algorithms described in this section. The factorization algorithm follows a recursion on the clique tree and visits the cliques in topological order. The permutation matrix P corresponds to a perfect elimination ordering which can be found with the MCS algorithm.

2.4.2 Value and Gradient of Dual Barrier

The Cholesky factorization provides an efficient method for evaluating the logarithmic barrier function for the cone $\mathbf{S}^n_{V,+}$, defined as

$$\phi : \mathbf{S}^n_V \to \mathbf{R}, \qquad \phi(S) = -\log\det S, \qquad \mathbf{dom}\,\phi = \mathbf{S}^n_{V,++}. \tag{2.11}$$

To evaluate $\phi(S)$, we compute the Cholesky factorization (2.10) and evaluate

$$\phi(S) = -2\sum_{i=1}^{n} \log L_{ii}.$$

The gradient of ϕ is given by

$$\nabla\phi(S) = -P_V(S^{-1}). \tag{2.12}$$

Although S^{-1} is generally dense, its projection on the sparsity pattern can be computed from the Cholesky factorization of S without computing any other entries of S^{-1}. The algorithm is recursive and visits the nodes of the clique tree in reverse topological order.

2.4.3 Hessian and Inverse Hessian of Dual Barrier

The Hessian of ϕ at $S \in \mathbf{S}^n_{V,++}$, applied to a matrix $Y \in \mathbf{S}^n_V$, is given by

$$\nabla^2\phi(S)[Y] = P_V(S^{-1}YS^{-1}). \tag{2.13}$$

Evaluating this expression efficiently for large sparse matrices requires methods for computing the projection of $S^{-1}YS^{-1}$ on $\mathbf{S}^n_V$, without, however, computing $S^{-1}YS^{-1}$. This can be accomplished by exploiting chordal structure. The expression (2.13) can be evaluated from the Cholesky factorization of S and the projected inverse $P_V(S^{-1})$ via two recursions on the clique tree. The first recursion visits the cliques in topological order while the second recursion visits the cliques in reverse topological order.

The two recursions form a pair of adjoint linear operators, and the algorithm for applying the Hessian can be interpreted as evaluating the Hessian in a factored form,

$$\nabla^2\phi(S)[Y] = \mathcal{L}_{\mathrm{adj}}(\mathcal{L}(Y)), \tag{2.14}$$

by first evaluating a linear mapping $\mathcal{L}$ via an algorithm that visits the cliques in topological order, followed by the evaluation of the adjoint $\mathcal{L}_{\mathrm{adj}}$ via an algorithm that follows the reverse topological order. Furthermore, the factor $\mathcal{L}$ of the Hessian is easily inverted, and this provides a method for evaluating

$$\nabla^2\phi(S)^{-1}[Y] = \mathcal{L}^{-1}(\mathcal{L}_{\mathrm{adj}}^{-1}(Y)) \tag{2.15}$$

(equivalently, for solving the linear equation $\nabla^2\phi(S)[U] = Y$) at the same cost as the evaluation of $\nabla^2\phi(S)[Y]$.

2.4.4 Value and Gradient of Primal Barrier

As a logarithmic barrier function for the matrix cone

$$\mathbf{S}^n_{V,\mathrm{c}+} = \{X \in \mathbf{S}^n_V \mid X \succeq_{\mathrm{c}} 0\} = (\mathbf{S}^n_{V,+})^*$$

we can use the Legendre transform of the barrier ϕ of $\mathbf{S}^n_{V,+}$ [NN94, p. 48]. For $X \succ_{\mathrm{c}} 0$, the barrier function is defined as

$$\phi_{\mathrm{c}}(X) = \sup_{S \succ 0} \left(-X \bullet S - \phi(S)\right). \tag{2.16}$$

(This is the Legendre transform of ϕ evaluated at $-X$.) The optimization problem (2.16) is the dual of the maximum determinant positive definite completion problem (2.7), and it can be solved analytically if the sparsity pattern is chordal. The solution is the positive definite matrix $\hat{S} \in \mathbf{S}^n_V$ that satisfies

$$P_V(\hat{S}^{-1}) = X, \tag{2.17}$$

and as we have seen in the previous section, $\hat{S}^{-1}$ is the maximum determinant positive definite completion of X. This provides an efficient method for evaluating ϕ_{c}: we first compute the Cholesky factorization $\hat{S} = LL^T$ of the solution $\hat{S}$ of the maximization problem in (2.16), or equivalently, the nonlinear equation (2.17), and then compute

$$\phi_{\mathrm{c}}(X) = \log\det \hat{S} - n = 2\sum_{i=1}^{n} \log L_{ii} - n.$$

It follows from properties of Legendre transforms that

$$\nabla \phi_{\mathrm{c}}(X) = -\hat{S}, \tag{2.18}$$

and, from (2.17), this implies that $X \bullet \nabla \phi_{\mathrm{c}}(X) = -n$.

2.4.5 Hessian and Inverse Hessian of Primal Barrier

The Hessian of the primal barrier function is given by

$$\nabla^2 \phi_{\mathrm{c}}(X) = \nabla^2 \phi(\hat{S})^{-1}, \tag{2.19}$$

where $\hat{S}$ is the maximizer in the definition of $\phi_{\mathrm{c}}(X)$. This result follows from standard properties of the Legendre transform. We can therefore evaluate $\nabla \phi_{\mathrm{c}}(X)[Y]$ using the methods for evaluating the inverse Hessian of the dual barrier (2.15), and we can compute $\nabla^2 \phi_{\mathrm{c}}(X)^{-1}[Y]$ using the algorithm for evaluating the dual Hessian.

2.4.6 Step-length Calculation

Another problem that comes up in interior-point methods for conic optimization problems is a step-length calculation of the form

$$\alpha = \arg\max_{\alpha} \{\alpha \mid Z + \alpha \Delta Z \in K\} \tag{2.20}$$

where K is a proper convex cone, and $Z \in \mathbf{int}\, K$ and ΔZ are given. The cost of computing an accurate solution to this problem is generally much higher than the cost of computing a lower bound on the step-length using, for example, a backtracking line search. In practice the step-length calculation (2.20) is therefore often approximated very roughly using a backtracking line search. The downside of this is that the number of interior-point iterations may increase if the step-lengths obtained via backtracking are too conservative.

The step-length calculation (2.20) can be performed efficiently for chordal sparse matrix cones. When K is the cone $\mathbf{S}^n_{V,+}$, the problem (2.20) can be expressed as

$$\alpha = \arg\max_{\alpha}\{\alpha \mid \lambda_{\min}(S + \alpha\Delta S) \geq 0\}$$

where $S \in \mathbf{S}^n_{V,++}$ and $\Delta S \in \mathbf{S}^n_V$. The optimal α can be expressed in terms of the largest eigenvalue of the matrix $U = -L^{-1}\Delta S L^{-T}$ where $S = LL^T$ is the Cholesky factorization of S, i.e.,

$$\alpha = \arg\max_{\alpha}\{\alpha \mid I - \alpha U \succeq 0\} = \begin{cases} \frac{1}{\lambda_{\max}(U)} & \text{if } \lambda_{\max}(U) > 0 \\ \infty & \text{if } \lambda_{\max}(U) \leq 0. \end{cases}$$

A good approximation of $\lambda_{\max}(U)$ can be computed efficiently using Lanczos iteration with reorthogonalization (see [CW02]). Note that the matrix U need not be formed explicitly since we only need to evaluate matrix–vector products with U.

For step-length calculations in $\mathbf{S}^n_{V,\mathrm{c}+}$ we can apply Theorem 2.1 when V is chordal, i.e.,

$$X + \alpha\Delta X \succeq_{\mathrm{c}} 0 \quad \Leftrightarrow \quad X_{W_kW_k} + \alpha\Delta X_{W_kW_k} \succeq 0, \;\; k = 1, \ldots, l.$$

This means that we can compute α by first solving l dense generalized symmetric eigenvalues problems

$$\alpha_i = \arg\max_{\alpha}\{\alpha \mid \lambda_{\min}(X_{W_kW_k} + \alpha\Delta X_{W_kW_k}) \geq 0\}, \quad k = 1, \ldots, l,$$

and the step-length is then equal to

$$\alpha = \min(\alpha_1, \ldots, \alpha_l).$$

When the cliques are small, this approach can be much cheaper than computing α by solving a single large eigenvalue problem.

2.5 Chordal Embedding, Restriction, and Decomposition

When the data matrices A_i, C have a common chordal sparsity pattern V, we have seen that it is possible to formulate and take advantage of efficient algorithms for evaluating barriers and their first and second derivatives. In this section, we discuss how a nonchordal sparsity pattern can be extended to a chordal one by means of chordal embedding. We also briefly discuss chordal restriction, which extracts a chordal subgraph from a nonchordal graph. Finally, we outline some chordal decomposition techniques that can be used to convert a sparse matrix cone program into a standard SDP with a number of smaller but dense blocks.

2.5.1 Chordal Embedding of Nonchordal Sparsity Patterns

For nonchordal sparsity patterns, it is possible to exploit chordality by constructing a *chordal embedding* or *triangulation* of V, i.e., by adding edges to the graph to make it chordal. Chordal embeddings are not only useful for nonchordal sparsity patterns. By extending a chordal sparsity pattern, it is sometimes possible to "shape" the clique tree to improve the computational efficiency of the algorithms. For example, merging cliques with large relative overlap[3] often improves

[3] We refer to the cardinality of the intersection of two cliques W_i and W_j as the *overlap* between W_i and W_j. Similarly, the *relative overlap* between W_i and W_j is defined as the overlap between W_i and W_j, normalized by $\min(|W_i|, |W_j|)$.

performance. A similar observation was made by Fukuda et al. [FKM00] who demonstrated that with their conversion method, a balance between the number of cliques and the size of the cliques and the separator sets is critical in order to obtain good performance.

In practice, a chordal embedding is easily constructed by computing a symbolic Cholesky factorization of the sparsity aggregate pattern. The amount of fill-in (i.e., the number of added edges) generally depends heavily on the ordering of the nodes, and different chordal embeddings can be obtained simply by reordering the nodes. In sparse matrix computations, it is generally desirable to minimize fill-in, since fill-in requires additional storage and often increases the time needed to solve a system of equations. However, the problem of finding a minimum fill ordering is an NP-complete problem [RTL76, Yan81], and when practical considerations such as data locality and cache efficiency are considered, minimizing fill-in is not necessarily optimal in terms of run-time complexity. This can be seen in BLAS-based supernodal Cholesky codes where additional fill-in may be created in order to increase the size of the supernodes (so-called relaxed supernodal amalgamation). Some common fill reducing ordering heuristics are minimum degree (MD) orderings [Mar57, TW67, Ros72], approximate minimum degree (AMD) orderings [ADD96] and nested dissection (ND) orderings [Geo73]. The MD algorithm repeatedly selects a node of minimum degree to eliminate. This greedy heuristic works well in many cases, but it may produce fill even for chordal graphs. The chordal graph in Fig. 2.10 is an example of a graph for which a minimum degree ordering is not a perfect elimination ordering. The AMD algorithm is based on approximate node degrees instead of the true node degrees used in the MD algorithm. This modification is generally faster, and it typically results in nearly identical amounts of fill.

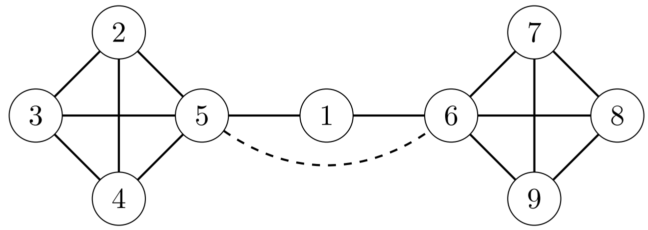


Figure 2.10: A chordal graph with a minimum degree ordering of the nodes. Eliminating node 1 results in a fill edge between nodes 8 and 9 (dashed arc), and hence a minimum degree ordering is not a perfect elimination ordering in general.

An alternative to embeddings based on heuristic fill reducing orderings is to compute a *minimal* chordal embedding. A chordal embedding is called minimal if the removal of any fill edge yields a nonchordal graph. A *minimum* chordal embedding (i.e., an embedding obtained from a minimum fill ordering) is a minimal chordal embedding, but the converse is not true in general. The MCS algorithm can be modified such that it produces a minimal chordal embedding when the sparsity pattern is nonchordal [BBH04]. This variant is known as MCS-M and its complexity is $O(ne)$ if e is the number of edges in the sparsity graph. A survey of minimal triangulation techniques can be found in [Heg06].

2.5.2 Chordal Restriction

A chordal restriction of a nonchordal graph $\mathcal{G}$ is a spanning subgraph $\mathcal{G}'$ of $\mathcal{G}$ that is chordal. The problem of finding a *maximal* chordal subgraph can be thought of as the opposite of the minimal triangulation problem. Given a nonchordal graph $\mathcal{G}$, we seek to remove a minimal subset of the edges in $\mathcal{G}$ such that the resulting spanning subgraph $\mathcal{G}'$ is chordal. In other words, we say that $\mathcal{G}'$ is a maximal chordal subgraph of $\mathcal{G}$ if adding an any edge from $\mathcal{G}$ to $\mathcal{G}'$ results in a nonchordal graph. A maximal chordal subgraph can be computed by the MAX-CHORD algorithm [DSW88]. The worst-case time complexity of this algorithm is $O(e\delta)$ if δ is the maximum degree of any vertex in $\mathcal{G}$. Unlike chordal embeddings, chordal restrictions discard one or more nonzero elements from the sparsity pattern to make it chordal, and hence chordal restrictions are mainly useful for approximation purposes.

2.5.3 Chordal Decomposition

Chordal structure can also be used to decompose sparse matrix cone constraints into a set of coupled constraints. The first chordal decomposition method to appear in the literature was pioneered by Fukuda et al. [FKM00]. Chordal decomposition has since been applied to semidefinite programming as a preprocessing technique [FFN06] and to reformulate an SDP into an equivalent convex–concave saddle point problem [LNM07]. A more general and systematic set of decomposition methods have recently been proposed by Kim et al. [KKM10]. We will discuss a simplified version of these methods for two types of constraints: *domain space* decomposition for the primal form inequality constraint $X \succeq_{\mathrm{c}} 0$ and *range space* decomposition for the dual form linear matrix inequality $C - \sum_{i=1}^{m} y_i A_i \succeq 0$.

We will assume that $C, A_i, X \in \mathbf{S}_V^n$ and that V is chordal. Furthermore, we denote with $\bar{W}_k \triangleq \{1, \ldots, n\} \setminus W_k$. The linear mapping $G_k^T X G_k \mapsto X_{W_k W_k}$ extracts the $|W_k| \times |W_k|$ principal submatrix $X_{W_k W_k}$ from X. If $W_k = \{j_1, j_2, \ldots, j_r\}$ and $j_1 < j_2 < \cdots j_r$, then $G_k \in \mathbf{R}^{n \times r}$ is defined as

$$G_k = \begin{bmatrix} e_{j_1} & e_{j_2} & \cdots & e_{j_r} \end{bmatrix}$$

where e_i is the ith column of the $n \times n$ identity matrix. The adjoint mapping $G_k Y G_k^T$ yields an $n \times n$ matrix Z that satisfies

$$Z_{\bar{W}_k \bar{W}_k} = 0, \quad Z_{W_k W_k} = Y, \quad Z_{W_k \bar{W}_k} = Z_{\bar{W}_k W_k}^T = 0.$$

Domain space decomposition Theorem 2.1 allows us to express the chordal sparse matrix cone constraint $X \succeq_{\mathrm{c}} 0$ as l standard positive semidefiniteness constraints, i.e.,

$$X \succeq_{\mathrm{c}} 0 \quad \Leftrightarrow \quad X_{W_k W_k} \succeq 0,\ k = 1, \dots, l. \tag{2.21}$$

These l constraints are implicitly coupled: X_{ij} appears in both $X_{W_p W_p}$ and $X_{W_q W_q}$ if $i, j \in W_p \cap W_q$. Since the cliques satisfy the running intersection property, it is sufficient to ensure consistency between pairs $X_{W_p W_p}, X_{W_q W_q}$ that correspond to parent–child pairs in the clique tree (i.e., $p = \mathrm{par}(q)$ or $q = \mathrm{par}(p)$). This allows us to decompose the constraint $X \succeq_{\mathrm{c}} 0$ as l standard matrix inequalities and $\tilde{m}$ equality constraints

$$\tilde{X}_k \succeq 0, \quad k = 1, \dots, l, \tag{2.22}$$

$$(G_k \tilde{X}_k G_k^T)_{U_k U_k} = (G_{\mathrm{par}(k)} \tilde{X}_{\mathrm{par}(k)} G_{\mathrm{par}(k)}^T)_{U_k U_k}, \quad k = 2, \dots, l. \tag{2.23}$$

The variable $\tilde{X}_k \in \mathbf{S}^{|W_k|}$ is associated with $X_{W_k W_k}$ in X, and the total number of (componentwise) equality constraints $\tilde{m}$ is

$$\tilde{m} = \sum_{i=2}^{l} \frac{1}{2} |U_i| (|U_i| + 1). \tag{2.24}$$

Recall that $U_1 = \emptyset$ by assumption, and hence the index k starts at 2 in (2.23) and (2.24).

The sparse matrix cone constraint $X \succeq_{\mathrm{c}} 0$ can also be decomposed by dualizing each of the constraints $X_{W_k W_k} \succeq 0$ in (2.21). Let $x = \mathbf{vec}(X)$ denote the vector of length $|V|$ with the lower-triangular nonzeros of $X \in \mathbf{S}_V^n$, scaled such that $X \bullet Y = \mathbf{vec}(X)^T \mathbf{vec}(Y)$. Then $X_{W_k W_k} \succeq 0$ can be expressed as an LMI $F_k(x) \succeq 0$ where $F_k(x) \in \mathbf{S}^{|W_k|}$ is a basis decomposition of $X_{W_k W_k}$, i.e.,

$$F_k(x) = \sum_{\substack{i,j \in W_k \\ i \geq j}} X_{ij} \cdot G_k^T E_{ij} G_k$$

where

$$E_{ij} = \begin{cases} e_i e_i^T & i = j \\ e_i e_j^T + e_j e_i^T & i \neq j. \end{cases}$$

Consequently, the sparse matrix cone constraint $X \succeq_{\mathrm{c}} 0$ can be expressed as k LMIs

$$F_k(x) \succeq 0, \quad k = 1, \ldots, l \tag{2.25}$$

with variable $x \in \mathbf{R}^{|V|}$. Note that only a subset of the elements of x is required in each of these LMIs.

Range space decomposition The range space decomposition methods can be thought of as the dual of the domain space decomposition methods.

The following theorem plays a fundamental role in range space decompositions. It asserts that a positive semidefinite matrix S with chordal sparsity can be decomposed as a sum of positive semidefinite matrices that correspond to the cliques of $\mathcal{G}(S)$.

Theorem 2.2. *Agler et al. [AHM88, Thm. 2.3]. Let V be a sparsity pattern and denote with $W_1, \ldots, W_l$ the cliques of the corresponding sparsity graph. The following statements are equivalent:*

(a) V is chordal.

(b) For any $S \in \mathbf{S}^n_{V,+}$ there exist $Y_k \in \mathbf{S}_+^{|W_k|}$, $k = 1, \ldots, l$, such that

$$S = \sum_{k=1}^{l} G_k Y_k G_k^T. \tag{2.26}$$

We remark that when S is positive definite, a decomposition of the form (2.26) can easily be obtained from the columns of the zero-fill Cholesky factor of $S = LL^T$.

If l_i is the ith column of L, the nonzeros in the outer product $l_i l_i^T$ are fully contained in one of the l cliques. It is possible to extend Theorem 2.2 with the following rank condition on the decomposition (2.26) [Kak10]

$$\mathbf{rank}(S) = \sum_{k=1}^{l} \mathbf{rank}(Y_k). \tag{2.27}$$

Since $\mathbf{rank}(A+B) \leq \mathbf{rank}(A)+\mathbf{rank}(B)$, this implies that there exists a decomposition $S = \sum_{k=1}^{l} G_k Y_k G_k^T$ such that $Y_k \succeq 0$ minimizes $\sum_{k=1}^{l} \mathbf{rank}(Y_k)$. Clearly, when $S = LL^T$ is positive definite, the clique-based decomposition extracted from the columns of L satisfies (2.27).

Theorem 2.2 can be used to decompose the sparse matrix cone constraint $C - A(y) \succeq 0$ where $A(y) = \sum_{i=1}^{m} y_i A_i$. Let $\tilde{C}_k, \tilde{A}_k(y) \in \mathbf{S}^{|W_k|}$, $k = 1, \ldots, l$, be any clique-based splitting of $C, A(y)$ such that

$$C = \sum_{k=1}^{l} G_k \tilde{C}_k G_k^T \quad \text{and} \quad A(y) = \sum_{k=1}^{l} G_k \tilde{A}_k(y) G_k^T.$$

To find a splitting of $C - A(y)$ of the form (2.26) such that $\tilde{C}_k - \tilde{A}_k(y) \succeq 0$, we parameterize all possible splittings

$$\begin{aligned} \tilde{S}_k(y,z) &= \tilde{C}_k - \tilde{A}_k(y) + \sum_{\substack{i,j \in U_k \\ i \geq j}} z_{ijk} G_k^T E_{ij} G_k - \sum_{q \in \mathrm{ch}(k)} \sum_{\substack{i,j \in U_q \\ i \geq j}} z_{ijq} G_k^T E_{ij} G_k & (2.28) \\ &= \tilde{C}_k - \tilde{A}_k(y) - \tilde{F}_k(z) & (2.29) \end{aligned}$$

where $\mathrm{ch}(i)$ is the set of indices of the children of the ith clique, and z is a vector of length $\tilde{m}$ with components

$$\{z_{ijk} \,|\, i,j \in U_k, i \geq j, k = 2, \ldots, l\}.$$

It is easy to verify that

$$F(z) = \sum_{k=1}^{l} G_k \tilde{F}_k(z) G_k^T = 0$$

and hence the sparse matrix cone constraint $C - A(y) \succeq 0$ can be decomposed into l constraints

$$\tilde{C}_k - \tilde{A}_k(y) - \tilde{F}_k(z) \succeq 0, \quad k = 1, \ldots, l. \tag{2.30}$$

An alternative decomposition can be obtained by dualizing the constraint $C - A(y) \succeq 0$, i.e.,

$$\sum_{k\,:\,i,j \in W_k}^{l} (G_k \tilde{S}_k G_k^T)_{ij} = C_{ij} - (A(y))_{ij}, \quad \forall (i,j) \in V, \tag{2.31}$$

$$\tilde{S}_k \succeq 0, \quad k = 1, \ldots, l. \tag{2.32}$$

Remark. The chordal sparsity pattern V plays an important role in the decomposition techniques. There is a trade-off between the number of additional variables and/or constraints in the decomposition and the number and the order of the matrix inequalities in the decomposition. Heuristic embedding techniques can therefore be useful as a means to obtain a computationally efficient decomposition. Moreover, when applied to SDPs as a preprocessing technique, chordal decomposition may induce sparsity in the Schur complement system, and this further complicates the choice of decomposition method.

2.6 Summary

In this chapter, we have described how a pair of standard primal and dual SDPs of the form (2.1a–2.1b) can be reformulated as a pair of sparse matrix cone programs (2.2a–2.2b) when the data matrices C, A_i share a common sparsity pattern V. In addition to the nonnegative orthant, the second-order cone, and the cone of positive semidefinite matrices, the sparse matrix cones include a number of useful intermediate cones that can be handled efficiently. Specifically, when the sparsity pattern V is chordal, the powerful recursive matrix algorithms described in (2.4) can be used to evaluate the barrier functions ϕ_c and ϕ and their first and second derivatives. These algorithms can be extended to nonchordal sparsity pattern by means of chordal embedding.

CHAPTER 3

Nonsymmetric Interior-Point Methods

In this chapter, we describe how the recursive matrix algorithms from the previous chapter can be used in interior-point methods for the nonsymmetric pair of sparse matrix cone problems (2.2a–2.2b). Interior-point methods for conic optimization require a barrier function that ensures that the iterates stay in the interior of a cone. Two different (but closely related) interior-point schemes exist: potential-reduction and path-following schemes. Karmarkar's groundbreaking polynomial-time algorithm for linear programming [Kar84], which sparked what is often referred to as "the interior-point revolution," is an example of a potential reduction method. We will focus on path-following methods, but the techniques that we describe also apply to potential-reduction methods.

3.1 Primal and Dual Path-following Methods

Polynomial-time interior-point methods for convex optimization belong to the class of *interior penalty schemes* developed by Fiacco and McCormick [FM68]. Given a general convex optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \end{array} \tag{3.1}$$

with convex objective and constraints, Fiacco and McCormick's *sequential unconstrained minimization technique* transforms (3.1) into a family of unconstrained problems, parameterized by $\mu > 0$,

$$\text{minimize} \quad F(x,\mu) = f_0(x) + \mu \textstyle\sum_{i=1}^m \psi(-f_i(x)). \tag{3.2}$$

The function $\psi : \mathbf{R}_+ \to \mathbf{R}$ is a strongly convex barrier function that satisfies $\psi(x) \to \infty$ as $x \to 0^+$. By choosing a decreasing parameter sequence $\mu_1, \mu_2, \ldots$ such that $\mu_k \to 0$ as $k \to \infty$, the minimizer $x(\mu_k)$ of $F(x, \mu_k)$ converges to the minimizer $x^\star$ of (3.1), provided that some mild conditions hold. Specifically, if the feasible set

$$\{x \mid f_i(x) \le 0, i = 1, \ldots, m\}$$

is bounded, then the minimizer of $F(x,\mu)$ is unique, and the points $\{x(\mu) \mid \mu > 0\}$ define a smooth curve that is known as the *central path*. This path is used in *barrier methods* as a "rough guide" toward the solution $x^\star$ by approximately solving a sequence of problems $F(x, \mu_k)$ using e.g. Newton's method.

Although the classical interior penalty scheme converges under rather mild conditions, its convergence may be too slow in practice. However, it is possible to construct interior-point methods with polynomial complexity for linear optimization over a closed convex set[1] if one can find a computable *self-concordant* barrier function for the feasible set [NN94].

In the previous chapter we saw that the barriers ϕ_c and ϕ as well as their first and second derivatives can be evaluated efficiently when V is chordal, using recursive matrix algorithms. The standard logarithmic barrier function $\bar{\phi}(X) = -\log\det X$ for the cone $\mathbf{S}^n_+$ is self-concordant, and hence the barrier ϕ for the cone $\mathbf{S}^n_{V,+}$ is also self-concordant (provided that $\mathbf{S}^n_{V,++} \neq \emptyset$ which is obviously satisfied when V includes the diagonal nonzeros). Self-concordance of ϕ_c, the barrier for $\mathbf{S}^n_{V,c+}$, follows from the properties of the Legendre transformation of

[1]The epigraph form of the convex optimization problem (3.1) is a linear optimization problem over a closed convex set.

a self-concordant barrier [NN94, p. 46]. The barriers can therefore be used as basic building blocks in interior-point methods for the pair of sparse matrix cone problems (2.2a–2.2b).

3.1.1 Central Path

If the primal problem (2.2a) is strictly feasible, we can define a family of equality-constrained problems, parameterized by a positive parameter μ,

$$\begin{array}{ll} \text{minimize} & C \bullet X + \mu\phi_{\mathrm{c}}(X) \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m. \end{array} \tag{3.3}$$

The cone constraint $X \succeq_{\mathrm{c}} 0$ in (2.2a) has been replaced with a barrier term in the objective in (3.3). The central path associated with (2.2a) is defined as points $X(\mu) \succ_{\mathrm{c}} 0$ that satisfy the optimality conditions accompanying the barrier problem (3.3)

$$A_i \bullet X(\mu) = b_i, \quad i = 1, \ldots, m \tag{3.4a}$$

$$\sum_{i=1}^{m} y_i A_i + S = C \tag{3.4b}$$

$$\mu \nabla \phi_{\mathrm{c}}(X(\mu)) = -S. \tag{3.4c}$$

Here $y \in \mathbf{R}^m$ is a Lagrange multiplier for the equality constraints in (3.3), and S is an auxiliary variable. The central path for the dual problem (2.2b) can be defined in a similar way. The dual barrier problem is given by

$$\begin{array}{ll} \text{maximize} & b^T y - \mu\phi(S) \\ \text{subjecto to} & \sum_{i=1}^{m} y_i A_i + S = C \end{array} \tag{3.5}$$

and the dual central path consists of points $y(\mu), S(\mu) \succ 0$ that satisfy the optimality conditions

$$A_i \bullet X = b_i, \quad i = 1, \ldots, m \tag{3.6a}$$

$$\sum_{i=1}^{m} y_i(\mu) A_i + S(\mu) = C \tag{3.6b}$$

$$\mu \nabla \phi(S(\mu)) = -X \tag{3.6c}$$

where X is a Lagrange multiplier for the equality constraint in (3.5). The optimality conditions (3.4) and (3.6) are, in fact, equivalent, and the points $X(\mu) \succ_{\mathrm{c}} 0, y(\mu), S(\mu) \succ 0$ define a primal–dual central path. The equivalence of (3.4) and (3.6) follows from properties of Legendre transform pairs. Interior-point methods that compute search directions by linearizing (3.4) are called *primal scaling* methods, and methods that linearize (3.6) are called *dual scaling* methods. We will refer to the corresponding linearized equations as the primal or dual Newton equations. Points on the primal–dual central path satisfy

$$C \bullet X(\mu) - b^T y(\mu) = C \bullet X(\mu) - \sum_{i=1}^{m} y_i(\mu) A_i \bullet X(\mu) = S(\mu) \bullet X(\mu) = \mu n,$$

i.e., the duality gap is equal to μn on the central path.

A symmetric tangent direction $\Delta \bar{X}, \Delta \bar{y}, \Delta \bar{S}$ at a point $X(\mu), y(\mu), S(\mu)$ on the central path can be derived from (3.4) or (3.6) by taking the derivative with respect to μ. This direction is a descent direction and it satisfies the equations

$$A_i \bullet \Delta \bar{X} = 0, \quad i = 1, \ldots, m \tag{3.7a}$$

$$\sum_{i=1}^{m} \Delta \bar{y}_i A_i + \Delta \bar{S} = 0 \tag{3.7b}$$

$$\mu \nabla^2 \phi_{\mathrm{c}}(X(\mu))[\Delta \bar{X}] + \Delta \bar{S} = -S(\mu). \tag{3.7c}$$

The equivalence of (3.4) and (3.6) on the central path means that the tangent direction is symmetric, i.e., we obtain the same direction when the last equation is expressed in terms of the dual barrier

$$\mu \nabla^2 \phi(S(\mu))[\Delta \bar{S}] + \Delta \bar{X} = -X(\mu). \tag{3.8}$$

Unfortunately, this symmetry is lost outside the central path.

3.1.2 Search Directions

At each iteration, a feasible path-following method computes a search direction and takes a step in this direction. We will distinguish between two kinds of search directions: *centering* directions and *approximate tangent* directions. The purpose of a centering step is to move closer to a point $X(\mu), y(\mu), S(\mu)$ on the central path. Short-step path-following methods rely only on centering steps, and progress is made by gradually decreasing μ between the centering steps. An approximate tangent direction, on the other hand, can be used to *predict* a new value of μ adaptively, and this typically leads to more aggressive updates of μ. It is also possible to take a step in an approximate tangent direction, or one can use a linear combination of a centering direction and an approximate tangent direction. Methods that take steps in an approximate tangent direction are often referred to as long-step methods. Although the short-step methods yield the best complexity bounds, the more aggressive long-step methods are generally faster in practice.

We obtain a *primal scaling* centering direction if we linearize the primal optimality conditions (3.4) around the current feasible iterate $X \succ_{\mathrm{c}} 0, y, S \succ 0$ (the current primal and dual iterates in an interior-point method), i.e.,

$$A_i \bullet \Delta X = 0, \quad i = 1, \ldots, m \tag{3.9a}$$

$$\sum_{i=1}^{m} \Delta y_i A_i + \Delta S = 0 \tag{3.9b}$$

$$\mu \nabla^2 \phi_c(X)[\Delta X] + \Delta S = -R \tag{3.9c}$$

where the right-hand side is $R = \mu \nabla \phi_c(X) + S$. The solution $\Delta X, \Delta y, \Delta S$ can be interpreted as a Newton step for the nonlinear system (3.4) so we refer to (3.9) as the primal Newton equations. We obtain the dual Newton equations and a *dual scaling* centering direction if we linearize the dual optimality conditions (3.6). Notice the similarity between the Newton equations (3.9) and the central path equations (3.7).

For a symmetric cone K with barrier ψ and any points $x \in \mathbf{int}\, K$ and $s \in \mathbf{int}\, K^*$, it is always possible to compute a so-called *scaling-point* $w \in \mathbf{int}\, K$ that satisfies the scaling relation

$$\nabla^2 \psi(w)[x] = s. \tag{3.10}$$

This property is used in symmetric primal–dual interior-point methods to compute symmetric approximate tangent directions at points outside the central path. For nonsymmetric cones, an approximate tangent direction can be computed using the Hessians of both the primal and the dual barriers. Unfortunately, this method is expensive [Nes96]. Another possibility is to use a sequence of primal or dual centering steps in combination with a technique referred to as *lifting* to find a point in the vicinity of the central path that satisfies a scaling relation similar to (3.10) [Nes06b, Nes06a]. We now outline the primal scaling variant of the technique. (The dual scaling variant is similar.) Suppose that the primal scaling centering step computed at iteration k satisfies

$$\lambda = (\Delta X \bullet \nabla^2 \phi_c(X^{(k)})[\Delta X])^{1/2} < 1$$

i.e., the Newton decrement is less than one which is true if X is sufficiently close to a target point $X(\mu), y(\mu), S(\mu)$ on the central path. A full step in the negative centering direction is therefore feasible [NN94, p. 13]. We can now use (3.9c) to link $\tilde{S} = S^{(k)} + \Delta S$ and a *lifted* point $\tilde{X} = X^{(k)} - \Delta X$ as

$$\mu \nabla^2 \phi_c(X^{(k)})[\tilde{X}] = \tilde{S}. \tag{3.11}$$

This scaling relation provides a way to compute an approximate tangent direction $\Delta X_{\mathrm{at}}, \Delta y_{\mathrm{at}}, \Delta S_{\mathrm{at}}$ that satisfies (3.9) with $R = \tilde{S}$. It follows from (3.9a) and (3.9b) that $\Delta S_{\mathrm{at}} \bullet \Delta X_{\mathrm{at}} = 0$, and moreover,

$$\begin{aligned} S \bullet \Delta X + \Delta S \bullet X &= S \bullet \Delta X + (-S - \mu \nabla^2 \phi_c(X(\mu))[\Delta X]) \bullet X \\ &= S \bullet \Delta X - S \bullet X - S \bullet \Delta X = -S \bullet X. \end{aligned}$$

A full step in the approximate tangent direction therefore reduces the duality gap to zero, i.e.,

$$(\tilde{X} + \Delta X_{\mathrm{at}}) \bullet (\tilde{S} + \Delta S_{\mathrm{at}}) = 0. \tag{3.12}$$

A full approximate tangent step, however, is never feasible.

3.1.3 Feasible Start Path-Following Method

Feasible-start interior-point methods require a strictly feasible starting point, i.e., a point $X^{(0)} \succ_c 0, y^{(0)}, S^{(0)} \succ 0$ that satisfies

$$A_i \bullet X^{(0)} = b_i, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} y_i^{(0)} A_i + S^{(0)} = C.$$

We will discuss initialization from infeasible (but interior) points in Section 3.3. An outline of a basic path-following method for the pair of problems (2.2a–2.2b) is given in Algorithm 3.1. The data matrices A_i are assumed to be linearly independent, and we also assume that a chordal embedding is available and that a clique tree has been computed. The line search can be implemented in different ways. We will discuss our implementation of a slightly modified version of Algorithm 3.1 in Section 3.4. Note that the Newton equations are solved more than once (with different right-hand sides) at some iterations. However, the cost associated with solving multiple systems is negligible since only one factorization is needed. In the next section we discuss how the Newton equations can be solved efficiently using the chordal matrix algorithms from the previous chapter.

3.2 Newton System

The solution of the Newton equations (3.9) forms the bulk of the computation in any interior-point method. In this section, we describe methods for solving the Newton equations when V is chordal.

3.2.1 Primal Scaling Methods

If we eliminate ΔS in the Newton equations (3.9), we obtain the system

$$A_i \bullet \Delta X = r_i, \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} \Delta y_i A_i - \mu \nabla^2 \phi_{\mathrm{c}}(X)[\Delta X] = R, \tag{3.13}$$

where $r_i = b_i - A_i \bullet X$ and

$$R = C - \sum_{i=1}^{m} y_i A_i + \mu \nabla \phi_{\mathrm{c}}(X).$$

The variables are $\Delta X \in \mathbf{S}_V^n$, $\Delta y \in \mathbf{R}^m$, so this is a set of $|V| + m$ equations in $|V| + m$ variables (recall that $|V|$ is the number of lower triangular nonzeros in

Algorithm 3.1. Feasible Start Primal Scaling Path-Following Method

---

Input: strictly feasible starting point $X^{(0)}, y^{(0)}, S^{(0)}$
Input: parameters $\delta \in (0,1)$, $\epsilon_{\text{rel}} > 0$, and $\epsilon_{\text{abs}} > 0$
for $k = 0, 1, 2, \dots$ **do**

Compute gap: $\mu = (X^{(k)} \bullet S^{(k)})/n$.

Terminate if the (approximate) optimality condition $n\mu \le \epsilon_{\text{abs}}$ or

$$\min\{C \bullet X^{(k)}, -b^T y^{(k)}\} < 0 \quad \text{and} \quad \frac{n\mu}{-\min\{C \bullet X^{(k)}, -b^T y^{(k)}\}} \le \epsilon_{\text{rel}}$$

is satisfied.

Compute centering direction $\Delta X_{\text{cnt}}, \Delta y_{\text{cnt}}, \Delta S_{\text{cnt}}$:
– solve (3.9) with $R = \mu \nabla \phi_{\text{c}}(X^{(k)}) + S^{(k)}$

if $(\Delta X_{\text{cnt}} \bullet \nabla^2 \phi_{\text{c}}(X^{(k)})[\Delta X_{\text{cnt}}])^{1/2} \le \delta$ **then**

Compute approximate tangent direction $\Delta X_{\text{at}}, \Delta y_{\text{at}}, \Delta S_{\text{at}}$:
– solve (3.9) with $R = \tilde{S} = S^{(k)} + \Delta S_{\text{cnt}}$

Calculate $\hat{\mu} := (1-\alpha)(\tilde{X} \bullet \tilde{S})/n$ where $\tilde{X} = X^{(k)} - \Delta X_{\text{cnt}}$ and

$$\alpha = 0.98 \cdot \sup_{\alpha} \left\{ \alpha \in [0,1) \,|\, \tilde{X} + \Delta X_{\text{at}} \succ_{\text{c}} 0, \tilde{S} + \Delta S_{\text{at}} \succ 0 \right\}.$$

Compute new search direction $\Delta X, \Delta y, \Delta S$:
– solve (3.9) with $\mu = \hat{\mu}$ and $R = \hat{\mu} \nabla \phi_{\text{c}}(X^{(k)}) + \hat{S}$

Line search: compute primal and dual step sizes α_{p} and α_{d}.

Update the variables:

$$X^{(k+1)} := \tilde{X} + \alpha_{\text{p}} \Delta X,\ y^{(k+1)} := \tilde{y} + \alpha_{\text{d}} \Delta y,\ S^{(k+1)} := \tilde{S} + \alpha_{\text{d}} \Delta S$$

else

Line search: compute primal and dual step sizes α_{p} and α_{d}.

Update the variables:

$$X^{(k+1)} := X^{(k)} + \alpha_{\text{p}} \Delta X_{\text{cnt}},\ y^{(k+1)} := y^{(k)} + \alpha_{\text{d}} \Delta y_{\text{cnt}},\ S^{(k+1)} := S^{(k)} + \alpha_{\text{d}} \Delta S_{\text{cnt}}$$

end if
end for

V). By eliminating ΔX we can further reduce the Newton equations to

$$H\Delta y = g, \tag{3.14}$$

where H is the positive definite symmetric matrix with entries

$$H_{ij} = A_i \bullet \left(\nabla^2\phi_c(X)^{-1}[A_j]\right), \quad i,j = 1,\ldots,m, \tag{3.15}$$

and $g_i = \mu r_i + A_i \bullet (\nabla^2\phi_c(X)^{-1}[R])$.

We now outline two methods for solving (3.14), and we will compare their practical performance in Section 3.5. Recall from Section 2.4 that $\nabla^2\phi_c(X) = \nabla^2\phi(\hat{S})^{-1}$ where $\hat{S}$ solves $P_V(\hat{S}^{-1}) = X$, and that the Cholesky factor L of $\hat{S}$ is readily computed from X using a recursive algorithm. The entries of H can therefore be written

$$H_{ij} = A_i \bullet (\nabla^2\phi(\hat{S})[A_j]), \quad i,j = 1,\ldots,m. \tag{3.16}$$

Method 1: Cholesky factorization

The first method explicitly forms H, column by column, and then solves (3.14) using a dense Cholesky factorization. We distinguish two techniques for computing column j. The first technique is a straightforward evaluation of (3.16). It first computes $\nabla^2\phi(\hat{S})[A_j]$ and then completes the lower-triangular part of column j of H by making inner products with the matrices A_i:

T1:
$U := \nabla^2\phi_c(X)^{-1}[A_j]$
for $i = j$ to m **do**
  $H_{ij} := A_i \bullet U$
end for

Additional sparsity of A_i, relative to V, can be exploited in the inner products, but additional sparsity in A_j is not easily exploited. The matrix U is a dense matrix in $\mathbf{S}^n_V$, and A_j is handled as a dense element in $\mathbf{S}^n_V$.

The second technique is useful when A_j is very sparse, relative to V, and has only a few nonzero columns. We write A_j as

$$A_j = \sum_{(p,q)\in I_j} (A_j)_{pq} e_p e_q^T,$$

where e_k is the kth unit vector in $\mathbf{R}^n$ and I_j indexes the nonzero entries in A_j. The expression for H_{ij} in (3.14) can then be written as

$$\begin{aligned} H_{ij} &= \sum_{(p,q)\in I_j} (A_j)_{pq} \left(A_i \bullet (\nabla^2 \phi_c(X)^{-1}[e_p e_q^T])\right) \\ &= \sum_{(p,q)\in I_j} (A_j)_{pq} \left(A_i \bullet P_V(\hat{S}^{-1} e_p e_q^T \hat{S}^{-1})\right) \\ &= \sum_{(p,q)\in I_j} (A_j)_{pq} \left(A_i \bullet (L^{-T} L^{-1} e_p e_q^T L^{-T} L^{-1})\right). \end{aligned}$$

If A_j has only a few nonzero columns, it is advantageous to precompute the vectors $u_k = L^{-T}L^{-1}e_k$ that occur in this expression. This results in the following algorithm for computing column j of H:

T2: Solve $LL^T u_k = e_k$ for $k \in \{i \mid A_j e_i \neq 0\}$

for $i = j$ to m **do**

$H_{ij} := \sum_{(p,q)\in I_j} (A_j)_{pq} u_q^T A_i u_p$

end for

Since A_j is symmetric, the summation can easily be modified to use only the lower triangular entries of A_j. If A_j has ζ_j nonzero columns, T2 requires that ζ_j vectors u_k be computed and stored in order to form column j of H. Since this is generally the most expensive part of T2, the technique is efficient only when ζ_j is small.

T2 is similar to a technique used in SDPA-C [NFF03, p. 316]. However, SDPA-C only stores two vectors at the time, and forming the jth column of H requires that $2\zeta_j$ systems of the form $LL^Tx = b$ be solved (twice the number of T2). It is also worth mentioning that low-rank techniques such as those used in DSDP and SDPT3 provide a more general and often faster alternative to T2. However, these low rank techniques require that a low-rank factorization of the data matrices be computed and stored as a preprocessing step. Furthermore, we remark that in its current form, T2 does not exploit any block structure of $\hat{S}$ when computing u_k.

A threshold on the number of nonzero columns ζ_j can be used as a heuristic for choosing between T1 and T2. Based on some preliminary experiments, we set the threshold to $n/10$ in the implementation used in Section 3.5 (in other words, we use T1 if $\zeta_j > n/10$ and T2 otherwise). Finally we note that the total flop count can be reduced by permuting the order of the data matrices.

Method 2: QR factorization

The second method for solving the reduced Newton equation (3.14) avoids the explicit calculation of H by applying the factorization of the Hessian matrix in Section 2.4. Using the factorization $\nabla^2\phi_{\mathrm{c}}(X)^{-1} = \nabla^2\phi(\hat{S}) = \mathcal{L}_{\mathrm{adj}} \circ \mathcal{L}$, we can write

$$H_{ij} = A_i \bullet \left(\nabla^2\phi_{\mathrm{c}}(X)^{-1}[A_j]\right) = \mathcal{L}(A_i) \bullet \mathcal{L}(A_j).$$

The factorization allows us to express H as $H = \tilde{A}^T\tilde{A}$ where $\tilde{A}$ is a $|V| \times m$ matrix with columns $\mathbf{vec}(\mathcal{L}(A_i))$. (Recall that the $\mathbf{vec}(\cdot)$ operator converts matrices in $\mathbf{S}^n_V$ to vectors containing the lower-triangular nonzeros, scaled so that $\mathbf{vec}(U_1)^T\,\mathbf{vec}(U_2) = U_1 \bullet U_2$ for all $U_1, U_2 \in \mathbf{S}^n_V$.) Instead of computing a Cholesky factorization of H (constructed via (3.15) or as $\tilde{A}^T\tilde{A}$), we can compute a QR-decomposition of $\tilde{A}$ and use it to solve (3.14). This is important, because the explicit computation of H is a source of numerical instability.

The second method is a variation of the *augmented systems* approach in linear programming interior-point methods. In the augmented systems approach, one computes the solution of the Newton equation

$$\begin{bmatrix} -D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \tag{3.17}$$

(the linear programming equivalent of (3.13)) via an $\mathrm{LDL^T}$ or QR factorization, instead of eliminating Δx and solving the 'Schur complement' equation $AD^{-1}A^T\Delta y = r_2 + AD^{-1}r_1$ by a Cholesky factorization [FM93], [Wri97a, p. 220]. The augmented system (3.17) can be solved using a sparse $\mathrm{LDL^T}$ factorization (for sparse A), or via a QR decomposition of $\tilde{A} = D^{-1/2}A^T$ (for dense A). The augmented system method is known to be more stable than methods based on Cholesky factorization of $AD^{-1}A^T$. It is rarely used in practice, since it is slower for large, sparse LPs than computing the Cholesky factorization of $AD^{-1}A^T$. In semidefinite programming, the loss of stability in forming H is more severe than in linear programming [Stu03]. This would make the augmented system approach even more attractive than for linear programming, but unfortunately the large size of the Newton equations makes it very expensive. In our present context, however, the augmented system approach is often feasible, since we work in the subspace $\mathbf{S}^n_V$ where the row dimension of $\tilde{A}$ is equal to $|V|$.

3.2.2 Dual Scaling Methods

The dual Newton equations can be derived in a similar way, by linearizing (3.6):

$$-\mu\nabla^2\phi(S)^{-1}[\Delta X] + \sum_{i=1}^{m} \Delta y_i A_i = R, \qquad A_i \bullet \Delta X = r_i, \quad i = 1, \ldots, m \tag{3.18}$$

with $r_i = b_i - A_i \bullet X$ and

$$R = C - \sum_{i=1}^{m} y_i A_i - 2S + \mu \nabla^2 \phi(S)^{-1}[X].$$

Eliminating ΔX and ΔS gives a set of linear equations (3.15) with

$$H_{ij} = A_i \bullet (\nabla^2 \phi(S)[A_j]), \quad i, j = 1, \ldots, m.$$

This matrix has the same structure as (3.16). Therefore, the same methods can be used to compute the dual scaling direction as for the primal scaling direction, and the complexity is exactly the same.

3.2.3 Complexity

The cost of solving the Newton system (3.13) is dominated by the cost of solving (3.14). Method 1 solves (3.14) by explicitly forming H and then applying a Cholesky factorization. Method 2 avoids explicitly forming H, by applying a QR factorization to a matrix $\tilde{A}$ that satisfies $H = \tilde{A}^T \tilde{A}$.

Method 1 The cost of solving (3.14) via the Cholesky factorization of H depends on the sparsity of the data matrices and on the techniques used to form H. The matrix H is generally dense, and hence the cost of factorizing H is $O(m^3)$. If all data matrices share the same sparsity pattern V and are dense relative to $\mathbf{S}^n_V$, the cost of forming H using technique T1 is $mK + O(m^2|V|)$ where K is the cost of evaluating $\nabla^2 \phi_c(X)^{-1}[A_j]$. If the data matrices A_i are very sparse relative to V, the cost of computing H using T1 is roughly mK.

With technique T2, the dominating cost of computing the columns of H is solving the systems $LL^T u_k = e_k$ for $k \in \{i \mid A_j e_i \neq 0\}$. Thus, the cost mainly depends on $|V|$ and the number of nonzero columns in the data matrices. When the data matrices only have a small number of nonzeros, T2 is generally many times faster than T1.

The cost K depends on the clique distribution (i.e., the structure of V) in a complicated way, but for special cases such as banded and arrow matrices we have $|V| = O(n)$ and $K = O(n)$, and in these special cases the cost of one iteration is *linear* in n.

Method 2 Solving (3.14) via a QR decomposition of $\tilde{A}$ costs $O(mK)$ to form $\tilde{A}$ and $O(m^2|V|)$ to compute the QR decomposition. In particular, the cost of one iteration is also *linear* in n for banded and arrow matrices, when the other dimensions are fixed. If the coefficients A_i are dense relative to V, the cost of Method 2 is at most twice that of Method 1 when only T1 is used.

3.3 Initialization From Infeasible Starting-points

The path-following algorithm that we have described in the previous sections is a feasible start algorithm. While it is sometimes possible to find a feasible starting point analytically, this is generally not the case. We now describe two different methods for initializing the algorithm from an infeasible (but interior) point. The first method solves a so-called "Phase I" problem in order to find a feasible starting point. The phase I problem is typically as expensive to solve as the original problem itself. The second method relies on a *self-dual embedding* of the original problem for which it is straightforward to find a centered starting point. The self-dual embedding also provides an elegant way to detect infeasibility, and the solution to the original problem can easily be recovered from the solution to the embedding problem.

3.3.1 Phase I

Algorithm 3.1 is a feasible-start method, and hence we need a feasible starting point. To this end we first solve the least-norm problem

$$\begin{array}{ll} \text{minimize} & \|X\|_F^2 \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m. \end{array} \tag{3.19}$$

If the solution X_{ln} satisfies $X_{\mathrm{ln}} \succ_{\mathrm{c}} 0$, we use it as the starting point. On the other hand, if $X_{\mathrm{ln}} \preceq_{\mathrm{c}} 0$, we solve the phase I problem

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m \\ & \mathbf{tr}(X) \leq M \\ & X + (s - \epsilon) I \succeq_{\mathrm{c}} 0, \quad s \geq 0 \end{array} \tag{3.20}$$

where $\epsilon > 0$ is a small constant. The constraint $\mathbf{tr}(X) \leq M$ (with M large and positive) is added to bound the feasible set. The optimal solution $X^\star$ is strictly feasible in (2.2a) if $s^\star < \epsilon$. The phase I problem (3.20) can be cast in standard form and solved using the feasible start algorithm.

3.3.2 Self-dual Embedding

Self-dual embeddings were first used in interior-point methods for linear programming [YTM94], and later extended to semidefinite and cone programming [PS95, KRT97, LSZ00], and nonlinear convex optimization [AY98]. The embedded problem is defined as

$$
\begin{array}{llllllll}
\text{minimize} & (n+1)\theta & & & & & & \\
\text{subject to} & & A_i \bullet X & -b_i\tau & +r_i^{(0)}\theta & = & 0, & i=1,\ldots,m \\
& -\sum_{i=1}^m y_i A_i & & +C\tau & +R^{(0)}\theta & = & S & \\
& b^T y & -C\bullet X & & +\rho^{(0)}\theta & = & \kappa & \\
& -(r^{(0)})^T y & -R^{(0)}\bullet X & -\rho^{(0)}\tau & & = & -(n+1) & \\
& \multicolumn{7}{l}{X \succeq_{\mathrm{c}} 0, \quad \tau \geq 0, \quad S \succeq 0, \quad \kappa \geq 0}
\end{array}
\tag{3.21}
$$

with variables $X, S \in \mathbf{S}_V^n$, $y \in \mathbf{R}^m$, $\kappa, \tau, \theta \in \mathbf{R}$. The extra parameters in the problem are chosen as

$$
r_i^{(0)} = b_i - A_i \bullet X^{(0)}, \qquad R^{(0)} = S^{(0)} + \sum_{i=1}^m y_i^{(0)} A_i - C, \qquad \rho^{(0)} = 1 + C\bullet X^{(0)} + b^T y^{(0)},
$$

where $X^{(0)} = I$, $S^{(0)} = I$, $y^{(0)} = 0$. It can be verified that the values

$$
X = X^{(0)}, \qquad S = S^{(0)}, \qquad y = y^{(0)}, \qquad \tau = \kappa = \theta = 1
$$

are strictly feasible. These values are used as a starting point in the algorithm. (The formulation is easily extended to general $X^{(0)} \succ_{\mathrm{c}} 0$, $S^{(0)} \succ 0$ and $y^{(0)}$.)

Problem (3.21) is formally identical to its dual, and therefore the optimal solution satisfies the complementarity condition $X \bullet S + \kappa\tau = 0$. Furthermore, by taking the inner product with (y, X, S, τ, θ) of each side of the equality in (3.21), we find that

$$
\theta = \frac{X \bullet S + \kappa\tau}{n+1}
$$

for all feasible points. Therefore $\theta = 0$ at the optimum.

Depending on the sign of the optimal κ and τ, we can extract from the optimal solution of the embedded problem either (a) the primal and dual solutions of (2.2a–2.2b), or (b) certificates of primal and dual infeasibility. If $\tau > 0$, $\kappa = 0$

at the optimum of (3.21), then X/τ is optimal in (2.2a) and y/τ, S/τ are optimal in (2.2b). If $\tau = 0$, $\kappa > 0$, then from the third equation, $C \bullet X$ or $-b^T y$ are negative. If $C \bullet X < 0$, then X is a certificate of dual infeasibility. If $b^T y > 0$, then y, S are a certificate of primal infeasibility. If $\tau = \kappa = 0$ at the optimum, then no conclusion about the original problem (2.2a–2.2b) can be made from the solution of (3.21).

The central path for the embedded problem is defined as the solutions X, S, y, κ, τ, θ that satisfy the constraints in (3.21) and

$$S = -\mu \nabla \phi_{\mathrm{c}}(X), \qquad \kappa = \frac{\mu}{\tau}, \tag{3.22}$$

where μ is a positive parameter. An equivalent condition is

$$X = -\mu \nabla \phi(S), \qquad \tau = \frac{\mu}{\kappa}. \tag{3.23}$$

We have noted in Section 2.4 that $X \bullet \nabla \phi_{\mathrm{c}}(X) = -n$ for all $X \succ_{\mathrm{c}} 0$. From (3.22) this implies that $X \bullet S + \tau\kappa = \mu(n+1)$ on the central path. On the other hand, we have seen above that $X \bullet S + \tau\kappa = \theta(n+1)$ for all feasible points in (3.21). Hence $\theta = \mu$ on the central path. These observations can be used to simplify the central path equations as

$$A_i \bullet X - b_i \tau = -\mu r_i^{(0)}, \quad i = 1, \ldots, m \tag{3.24}$$

$$-\sum_{i=1}^{m} y_i A_i + C\tau = S - \mu R^{(0)} \tag{3.25}$$

$$b^T y - C \bullet X = \kappa - \mu \rho^{(0)} \tag{3.26}$$

plus (3.22) or (3.23). The fourth equality constraint in (3.21) can be omitted because it follows from (3.24)–(3.26).

3.4 Implementation

The techniques described in Section 3.2 can be used in any interior-point methods based on primal or dual scaling directions, for example, barrier methods, infeasible primal or dual path-following methods, or nonsymmetric primal–dual methods [Nes06b, Nes06a]. In this section, we describe the algorithm used in the numerical experiments of Section 3.5. The implementation is a feasible-start path-following algorithm based on Algorithm 3.1 with some minor differences to allow initialization from a primal strictly feasible point X or a dual strictly feasible point S, y.

3.4.1 Algorithm Outline

The algorithm depends on parameters $\delta \in (0,1)$, $\gamma \in (0,1/2)$, $\beta \in (0,1)$, and tolerances ϵ_{abs} and ϵ_{rel}. (The values used in our experiments are $\delta = 0.9$, $\gamma = 0.1$, $\beta = 0.7$, $\epsilon_{\mathrm{abs}} = \epsilon_{\mathrm{rel}} = 10^{-7}$.) The algorithm also requires a strictly feasible starting point X and an initial value of the positive parameter μ (see §3.3.1).

Repeat the following steps.

1. *Primal centering.* Solve (3.9) with $R = C + \mu \nabla \phi_{\mathrm{c}}(X)$ and denote the solution by ΔX_{cnt}, Δy_{cnt}, ΔS_{cnt}. Evaluate the Newton decrement

$$\lambda = \left(\Delta X_{\mathrm{cnt}} \bullet \nabla^2 \phi_{\mathrm{c}}(X)[\Delta X_{\mathrm{cnt}}] \right)^{1/2}.$$

   If $\lambda \leq \delta$, set

$$S := C + \Delta S_{\mathrm{cnt}}, \qquad y := \Delta y_{\mathrm{cnt}}$$

   and proceed to step 2. Otherwise, conduct a backtracking line search to find a step size α that satisfies $X + \alpha\Delta X_{\mathrm{cnt}} \succ_{\mathrm{c}} 0$ and the Armijo condition

$$\frac{1}{\mu}(C \bullet (X + \alpha\Delta X_{\mathrm{cnt}})) + \phi_{\mathrm{c}}(X + \alpha\Delta X_{\mathrm{cnt}}) \le \frac{1}{\mu}(C \bullet X) + \phi_{\mathrm{c}}(X) - \alpha\gamma\lambda^2. \tag{3.27}$$

Update X as $X := X + \alpha\Delta X_{\mathrm{cnt}}$. Repeat step 1.

2. *Prediction step.* Solve (3.9) with $R = S$ and denote the solution by ΔX_{at}, Δy_{at}, ΔS_{at}. Calculate

$$\hat{\mu} := \frac{(\tilde{X} + \alpha\Delta X_{\mathrm{at}}) \bullet (S + \alpha\Delta S_{\mathrm{at}})}{n} = (1-\alpha)\frac{\tilde{X} \bullet S}{n}$$

where $\tilde{X} := X - \Delta X_{\mathrm{cnt}}$ and

$$\alpha = 0.98 \cdot \sup\left\{\alpha \in [0,1) \,\middle|\, \tilde{X} + \alpha\Delta X_{\mathrm{at}} \succ_{\mathrm{c}} 0, S + \alpha\Delta S_{\mathrm{at}} \succ 0\right\}.$$

Solve (3.9) with $\mu = \hat{\mu}$ and $R = S + \hat{\mu}\nabla\phi_{\mathrm{c}}(X)$. Conduct a backtracking line search to find a primal step size α_{p} such that $X + \alpha_{\mathrm{p}}\Delta X \succ_{\mathrm{c}} 0$ and the Armijo condition

$$\frac{1}{\hat{\mu}}(C \bullet (X + \alpha\Delta X)) + \phi_{\mathrm{c}}(X + \alpha\Delta X) \le \frac{1}{\hat{\mu}}(C \bullet X) + \phi_{\mathrm{c}}(X) - \alpha\gamma\lambda^2$$

is satisfied, where $\lambda = (\Delta X \bullet \nabla^2\phi_{\mathrm{c}}(X)[\Delta X])^{1/2}$. Conduct a backtracking line search to find the dual step size

$$\alpha_{\mathrm{d}} = \max_{k=0,1,\dots}\{\beta^k \mid S + \beta^k\Delta S \succ 0\}, \tag{3.28}$$

where $\beta \in (0,1)$ is an algorithm parameter. Update the variables

$$X := X + \alpha_{\mathrm{p}}\Delta X, \qquad y := y + \alpha_{\mathrm{d}}\Delta y, \qquad S := S + \alpha_{\mathrm{d}}\Delta S. \tag{3.29}$$

3. *Stopping criteria.* Terminate if the (approximate) optimality conditions

$$X \bullet S \le \epsilon_{\mathrm{abs}} \quad \text{or} \quad \left(\min\{C \bullet X, -b^T y\} < 0,\ \frac{X \bullet S}{-\min\{C \bullet X, -b^T y\}} \le \epsilon_{\mathrm{rel}}\right)$$

are satisfied. Otherwise, set $\mu := (X \bullet S)/n$ and go to step 1.

The three systems of equations that are solved at each iteration have the same coefficient matrix (if we absorb the scalar μ in ΔX) and therefore require only one factorization. They can be solved using either method 1 or method 2 from Section 3.2. We will refer to these two variants of the algorithm as SMCP-CH and SMCP-QR, respectively. Moreover the right-hand side of the third system is a linear combination of the right-hand sides of the first two systems, so the solution can be obtained by combining the first two solutions.

The prediction direction in step 3 is based on the approximate tangent direction proposed by Nesterov [Nes06a, Nes06b], who refers to the intermediate point $\tilde{X} = X - \Delta X$ as a *lifting*. Here the approximate tangent is used only to calculate $\hat{\mu}$; the actual step made in step 2 is a centering step with centering parameter $\hat{\mu}$.

The algorithm can be improved in several ways, most importantly by allowing infeasible starting points and by making more efficient steps in the approximate tangent direction. However these improvements would be of no consequence for the two questions we aim to answer in the experiments: how does the cost per iteration compare with general-purpose sparse semidefinite programming solvers and, secondly, how does the choice for primal scaling affect the accuracy that can be attained?

The line search in step 3 is carried out by means of bisection in our preliminary implementation. Note, however, that the step length can also be computed using the step-length calculation techniques outlined in §2.4.6.

3.4.2 Numerical Stability

As mentioned in Section 3.2, the augmented systems approach used in SMCP-QR is more stable than methods based on the Cholesky factorization of the Schur complement matrix H (method SMCP-CH). To illustrate the difference we consider

Solver	ϵ_1	ϵ_3	ϵ_5	ϵ_6
SMCP-CH	1.63e−07	0.00e+00	1.04e−05	6.79e−06
SMCP-QR	9.97e−14	0.00e+00	4.30e−10	3.63e−10
CSDP	5.67e−08	9.41e−09	3.66e−08	1.42e−08
SDPA	4.17e−07	1.81e−09	1.15e−06	1.03e−06
SDPT3	3.50e−07	1.80e−09	7.80e−07	7.40e−07
SEDUMI	1.45e−06	0.00e+00	-2.92e−08	3.28e−06

Table 3.1: DIMACS error measures for `control6` from SDPLIB.

the problem `control6` from SDPLIB [Bor99b]. This problem has a dual degenerate solution. Using default accuracy settings, the symmetric primal–dual codes CSDP, SDPA, SDPT3, and SEDUMI all stop prematurely because the Cholesky factorization of the Schur complement matrix fails near the solution where the Schur complement system is severely ill-conditioned. This is also the case for SMCP-CH. However the augmented systems approach in SMCP-QR solves the problem to high accuracy. Table 3.1 shows four of six so-called DIMACS error measures which are defined as [JPA00]

$$\epsilon_1 = \frac{\|r\|_2}{1+\|b\|_\infty}, \qquad \epsilon_3 = \frac{\|R\|_F}{1+\|C\|_{\max}},$$
$$\epsilon_5 = \frac{C \bullet X - b^T y}{1+|C \bullet X| + |b^T y|}, \qquad \epsilon_6 = \frac{S \bullet X}{1+|C \bullet X| + |b^T y|},$$

where $\|C\|_{\max} = \max_{i,j} |C_{ij}|$. (The remaining two DIMACS error measures are equal to zero for all the solvers.) Although the sparsity pattern associated with `control6` is not very sparse, the results in Table 3.1 demonstrate the benefit of the augmented systems approach in terms of numerical accuracy. Similar results can be observed for several other SDPLIB problems.

3.5 Numerical Experiments

To assess the speed and accuracy of the algorithm described in §3.4.1, we have conducted a series of experiments using a preliminary implementation of the algorithm. The implementation is based on a combination of Python code and C code, and it relies on two Python packages, CVXOPT 1.1.2 [DV08] and CHOMPACK 1.1 [DV09a]. We have also implemented and tested a method based on dual scaling. These results are not included but were similar to the results for the primal scaling method.

The experiments consist of some families of randomly generated problems, selected sparse problems from SDPLIB, and a set of very large sparse SDPs. All the experiments were conducted on a desktop computer with an Intel Core2 Quad Q6600 CPU (2.4 GHz), 4 GB RAM, and Ubuntu 9.10 (64-bit). Each problem instance was solved with the interior-point SDP solvers DSDP 5.8, SDPA-C 6.2.1, SDPT3 4.0b, and SEDUMI 1.2. The Matlab-based solvers SDPT3 and SEDUMI were run in Matlab R2008b while the other solvers are stand-alone and linked to ATLAS/LAPACK. It should be noted that DSDP reports "wall time" (i.e., real time) whereas the other solvers report CPU time. However, since a single-threaded version of DSDP is used, the difference between wall time and CPU time is negligible in practice.

We ran all the solvers with their default parameters and starting points. In our solver, we used the tolerances $\epsilon_{\mathrm{abs}} = \epsilon_{\mathrm{rel}} = 10^{-7}$ as exit conditions. Moreover, we applied one iterative refinement step when solving the Newton systems using SMCP-QR, and three iterative refinement steps when using SMCP-CH. We used the centering parameter $\delta = 0.9$, the backtracking parameters $\beta = 0.7$ and $\gamma = 0.1$, and the initial μ was set to 100.

For nonchordal sparsity patterns, we used the AMD ordering to compute chordal embeddings. Recall that SMCP-CH uses a combination of two techniques (referred to as T1 and T2 in §3.2.1) to form the Schur complement matrix. For each column of the Schur complement matrix, we employ a threshold to select between the two techniques. The threshold is based on the number of nonzero columns in A_i: T1 is used if A_i has more than $0.1 \cdot n$ nonzero columns, and otherwise T2 is used. This criterion is a simple heuristic based on experimentation, and leaves room for improvement. The second method, SMCP-QR, does not explicitly form the Schur complement matrix, and it treats each of the data matrices A_i as a dense element in $\mathbf{S}^n_V$.

The main purpose of the experiments is to compare the linear algebra complexity of the techniques described in the previous sections with existing interior-point solvers. For most test problems, the number of iterations needed by the different solvers was comparable and ranged between 20 and 50 iterations. We report the average time per iteration as well as the total time. The cost per iteration is arguably the most fair basis for a comparison, since the solvers use different stopping criteria, start at different initial points, and use different strategies for step size selection and for updating the central path parameter.

The solution times for SMCP-CH and SMCP-QR do not include the time spent in phase I. Most test problems did not require a phase I because the solution of the least-norm problem (3.19) happened to be strictly feasible. The few problems that did require the solution of a full phase I problem are pointed out in the text.

3.5.1 SDPs with Band Structure

In the first experiment we consider randomly generated SDPs with band structure, i.e., the matrices $C, A_i \in \mathbf{S}^n$ $(i = 1, \ldots, m)$ are all banded with a common bandwidth $2w+1$. The special case where all A_i are diagonal ($w = 0$) corresponds to a linear programming problem.

Table 3.2 shows the total time and the time per iteration as a function of n, with a fixed number of constraints ($m = 100$) and fixed bandwidth ($w = 5$). The time per iteration is also shown in the plot in Fig. 3.1. We see that, in terms of

n	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	0.12	0.05	0.20	5.1	0.20	0.09
200	0.23	0.10	1.2	20	0.46	0.36
400	0.48	0.22	9.0	86	1.5	2.2
800	1.0	0.45	132	576	5.4	16
1600	2.0	0.98	3988	4312	28	171

(a) Time per iteration (sec.)

n	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	4.2	1.8	6.3	117	3.0	1.2
200	9.3	4.3	41	510	7.9	5.7
400	18	8.3	314	2065	23	35
800	41	18	4766	13247	91	249
1600	94	46	171500	116426	500	3086

(b) Total time (sec.)

Table 3.2: Time per iteration and total time for randomly generated band SDPs with $m = 100$ and $w = 5$.

time per iteration, SMCP-CH and SMCP-QR scale linearly with n. The advantage is clear for large values of n. The high iteration times for DSDP and SDPA-C can be explained by implementation choices rather than fundamental limitations. The problem data in this experiment have full rank, and as a consequence, the low-rank techniques used in DSDP become expensive. Furthermore, recall that SDPA-C uses a technique similar to technique T2 described in §3.2.1. This is clearly inefficient when all the columns of the data matrices contain nonzeros.

In the next part of the experiment, we fix n and w and vary the number of

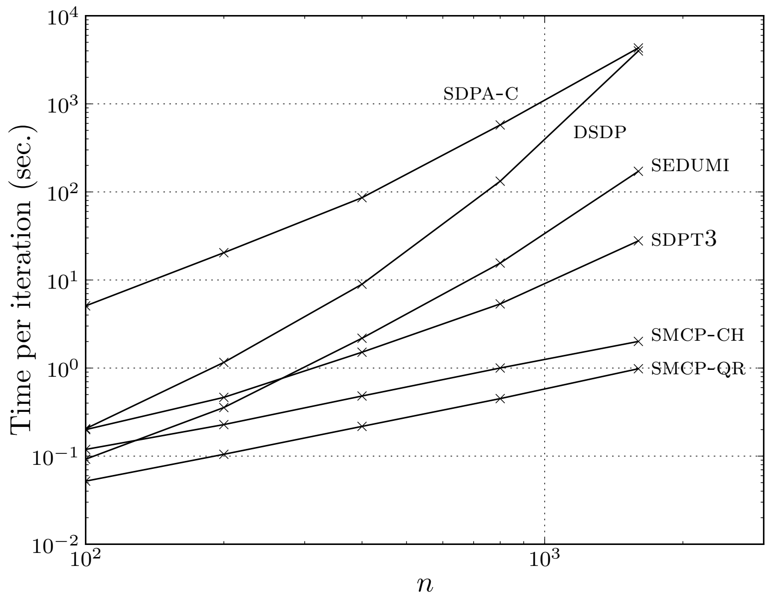


Figure 3.1: Time per iteration as a function of the problem parameter n for randomly generated band SDPs with half-bandwidth $w = 5$ and $m = 100$ constraints. The cost of one iteration increases linearly with n for SMCP-CH and SMCP-QR whereas the cost grows at least quadratically for the other solvers.

constraints m (Table 3.3). Here SMCP-QR is the fastest method, and for these

m	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
10	0.07	0.03	13	1.9	0.46	3.4
50	0.19	0.08	27	36	0.96	3.7
100	0.41	0.18	31	135	1.7	3.9
200	1.1	0.37	46	555	3.1	4.3
400	3.5	0.80	89	1571	6.3	6.0
800	12	1.8	-	6373	14	10

(a) Time per iteration (sec.)

m	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
10	2.8	1.2	450	47	7.4	62
50	7.5	3.2	1052	891	15	59
100	15	6.8	1102	3241	27	55
200	46	15	1888	13318	53	69
400	145	33	5493	40845	107	85
800	535	80	-	159337	246	159

(b) Total time (sec.)

Table 3.3: Time per iteration and total time for randomly generated band SDPs with $n = 500$ and $w = 3$. The SMCP-CH and SMCP-QR results for $m = 800$ do not include the time of the phase I. The other problems did not require a phase I.

problems it scales much better than SMCP-CH. We remark that DSDP did not return a solution for $m = 800$ (possibly due to insufficient memory). Again, the poor performance of SDPA-C on these chordal problems is likely a consequence of the technique used to compute the Schur complement matrix rather than a shortcoming of the completion technique.

Finally we consider a set of SDPs with different bandwidths and with n and m fixed. From the results in Table 3.4 it is clear that, for most of the solvers, there is a gap in time per iteration from $w = 0$ (i.e., a linear programming problem cast as an SDP) to $w = 1$ (i.e., a tridiagonal sparsity pattern). The gap is less evident for the chordal methods. Interestingly, SDPT3 is slightly faster when the bandwidth is increased from 16 to 32.

w	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0	0.03	0.02	0.39	3.3	0.07	0.02
1	0.10	0.05	2.1	8.4	0.24	0.31
2	0.14	0.06	1.2	10	0.28	0.33
4	0.19	0.09	1.2	17	0.41	0.36
8	0.34	0.16	1.2	31	0.66	0.40
16	0.71	0.33	2.2	65	1.2	0.50
32	1.7	0.84	2.2	127	0.87	0.68

(a) Time per iteration (sec.)

w	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0	1.4	0.65	20	95	1.2	0.20
1	4.4	2.0	109	235	4.0	4.7
2	4.9	2.2	42	259	4.5	4.6
4	7.3	3.4	41	393	6.5	4.7
8	13	6.0	39	782	9.9	5.6
16	26	12	73	1355	16	7.0
32	69	34	74	2660	11	8.2

(b) Total time (sec.)

Table 3.4: Time per iteration and total time for randomly generated band SDPs with $n = 200$ and $m = 100$. SMCP-CH and SMCP-QR required a phase I for $w = 0$ and $w = 1$ (time not included).

3.5.2 Matrix Norm Minimization

In the next experiment, we consider a family of matrix norm minimization problems of the form

$$\text{minimize} \quad \|F(x) + G\|_2 \tag{3.30}$$

where $F(x) = x_1 F_1 + \cdots + x_r F_r$, and with randomly generated problem data $G, F_i \in \mathbf{R}^{p \times q}$ with $p \geq q$. The matrix G is dense while the sparsity of each F_i is determined by a sparsity parameter $d \in (0, 1]$. The number of nonzero elements

F_i is given by $\max(1, \mathrm{round}(dpq))$, and the locations of the nonzero elements are selected at random for each F_i. All nonzero elements are randomly generated from a normal distribution.

The matrix norm minimization problem (3.30) can be formulated as an SDP:

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \begin{bmatrix} tI & F(x)+G \\ (F(x)+G)^T & tI \end{bmatrix} \succeq 0. \end{array} \tag{3.31}$$

The variables are $x \in \mathbf{R}^r$ and $t \in \mathbf{R}$. Since G is dense, the aggregate sparsity pattern is independent of the sparsity parameter d. Moreover, this aggregate sparsity pattern is nonchordal for $q > 1$, but a chordal embedding is easily obtained by filling the smaller of the two diagonal blocks in the aggregate sparsity pattern associated with (3.31). In the special case where $q = 1$, the SDP (3.31) is equivalent to a second-order cone program.

Of the $r+1$ data matrices associated with the SDP (3.31), one matrix always has full rank (the identity matrix) whereas the rank of each of the remaining r data matrices is at most $\min(2p, 2q)$. This means that low-rank structure may be exploited when $p \gg q$ or $p \ll q$.

In the first part of the experiment we look at the time per iteration for increasing values of p and with q and r fixed. The data matrices F_i are dense. The results in Table 3.5 verify that the time per iteration is roughly linear in $n = p+q$ for SMCP-CH and SMCP-QR. This is also shown in the plot in Fig. 3.2. Indeed, for large $p + q$, SMCP-CH and SMCP-QR are significantly faster in terms of time per iteration than the other interior-point solvers.

In the second part of the experiment we use matrix norm SDPs with F_i dense, a varying number of constraints $m = r + 1$, and p and q fixed. Table 3.6 summarizes the results. Notice that SMCP-QR performs quite well, and it scales much

$p+q$	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	0.23	0.12	0.09	0.91	0.29	0.09
200	0.48	0.24	0.39	2.1	0.78	0.34
400	1.0	0.51	1.8	5.5	2.5	1.8
800	2.1	1.1	10	15	9.1	15
1600	4.3	2.5	77	43	35	162

(a) Time per iteration (sec.)

$p+q$	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	6.2	3.1	3.1	21	3.5	1.2
200	13	6.3	8.2	55	9.3	4.4
400	29	14	38	132	32	25
800	60	32	219	437	136	223
1600	146	84	1772	1372	491	2424

(b) Total time (sec.)

Table 3.5: Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with r variables, for $q = 10$, $r = 100$.

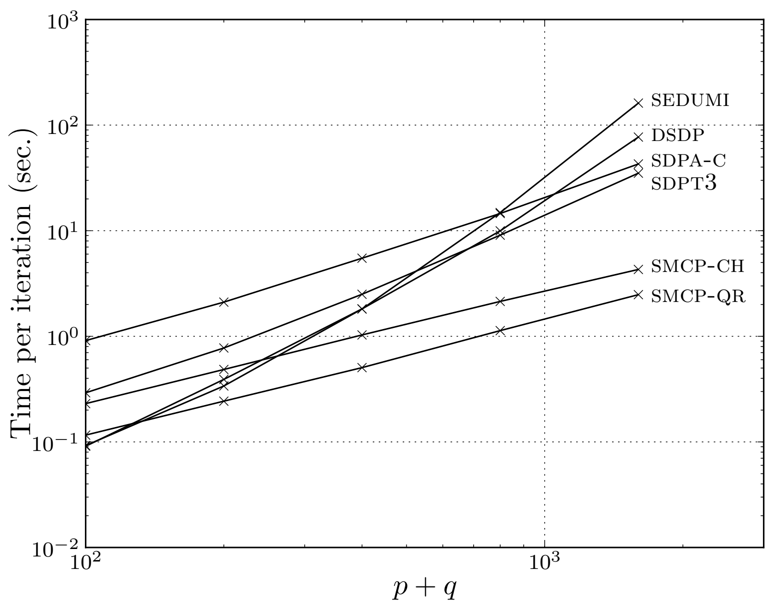


Figure 3.2: Time per iteration as a function of $p+q$ for randomly generated dense matrix norm problems with $q = 10$, $r = 100$. The iteration complexity is roughly linear for SMCP-CH and SMCP-QR.

r	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
50	0.51	0.23	1.0	1.8	1.4	1.8
100	1.1	0.52	1.9	5.6	2.6	2.0
200	2.7	1.0	4.0	19	5.2	2.3
400	7.8	2.2	8.5	69	11	3.3
800	26	4.9	15	262	24	6.6

(a) Time per iteration (sec.)

r	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
50	15	6.9	20	47	19	25
100	33	16	40	140	36	30
200	70	27	80	471	72	30
400	204	57	169	1523	141	43
800	682	128	434	5756	291	80

(b) Total time (sec.)

Table 3.6: Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with r variables, with $p = 400$, $q = 10$.

better than SMCP-CH. The main reason is that SMCP-CH will use Technique 1 to compute the columns of H since the data matrices F_i are dense (i.e., the number of nonzero columns in A_i is $n = p + q$, and as a consequence, Technique 2 is expensive). Notice also that SDPA-C is slow for large m. This can be attributed to the fact that SDPA-C uses an algorithm similar to Technique 2 to compute the Schur complement matrix H, and this is expensive whenever the data matrices have a large number of nonzero columns.

The effect of varying the density of F_i can be observed in Table 3.7 which shows the time per iteration for matrix norm problems with varying density and fixed dimensions. For d small, the data matrices A_i generally have only a few nonzero columns, and this can be exploited by SMCP-CH and SDPA-C. For the two most sparse problems ($d < 0.01$), method SMCP-CH used technique T2 for all but one column in the Schur complement matrix; for the problems with $d \geq 0.01$,

d	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0.001	0.31	0.96	0.09	0.22	0.22	1.7
0.005	0.69	0.97	0.28	0.42	0.33	1.7
0.01	1.2	0.97	0.36	0.53	0.60	1.8
0.02	1.2	0.96	0.96	0.71	1.6	1.8
0.05	1.2	0.95	1.6	1.2	2.3	1.8
0.1	1.3	0.96	1.5	2.1	2.5	1.8
0.25	1.5	0.98	2.5	4.7	3.2	1.9
0.5	1.9	1.0	3.0	9.6	4.0	2.0

(a) Time per iteration (sec.)

d	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0.001	9.2	29	2.0	4.9	2.9	24
0.005	19	26	6.5	10	4.6	26
0.01	34	28	8.2	13	8.4	26
0.02	33	27	21	16	21	24
0.05	40	31	36	30	30	23
0.1	37	28	34	49	33	26
0.25	39	25	54	117	48	27
0.5	50	26	67	232	56	28

(b) Total time (sec.)

Table 3.7: Time per iteration and total time for randomly generated sparse matrix norm problems of size $p \times q$ with r variables and density d, for $p = 400$, $q = 10$, and $r = 200$.

T1 was used for all columns. The times for SMCP-QR on the other hand are more or less independent of d since SMCP-QR handles the data matrices A_i as dense matrices in $\mathbf{S}_V^n$.

In the last part of this experiment, we use matrix norm problems with different values of q, with r and $p+q$ fixed, and with dense problem data F_i. The results are shown in Table 3.8. Both SMCP-CH and SMCP-QR perform relatively

q	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
1	0.10	0.05	1.6	0.19	1.5	28
2	0.12	0.05	1.7	0.32	1.7	28
5	0.25	0.12	1.9	0.72	1.7	27
10	0.51	0.23	2.3	1.4	2.8	26
20	1.3	0.58	3.8	3.0	4.0	27
50	7.8	3.5	5.8	9.8	7.3	28

(a) Time per iteration (sec.)

q	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
1	3.2	1.5	18	3.4	15	142
2	3.6	1.7	31	5.7	22	305
5	7.4	3.5	36	18	22	356
10	15	6.8	44	32	39	397
20	50	23	69	72	53	383
50	234	105	110	264	102	478

(b) Total time (sec.)

Table 3.8: Time per iteration and total time for randomly generated dense matrix norm problems of size $p \times q$ with r variables, for $p+q = 1000$, $r = 10$.

well, but the time per iteration does not scale well with q. Observe that in the special case where $q = 1$, the chordal techniques appear superior. However it should be noted that both SDPT3 and SEDUMI can handle second-order cone constraints directly with greater efficiency when the second-order cone constraints are explicitly specified as such.

3.5.3 Overlapping Cliques

In this experiment, we consider a family of SDPs with chordal sparsity, obtained by modifying a block diagonal sparsity pattern so that neighboring blocks overlap. Let l be the number of cliques, all of order N, and denote with u the overlap between neighboring cliques. Note that u must satisfy $0 \leq u \leq N-1$ where $u = 0$ corresponds to a block diagonal sparsity pattern and $u = N-1$ corresponds to a banded sparsity pattern with bandwidth $2u+1$. The order of V is $n = l(N-u)+u$, and the l cliques are given by

$$W_i = \{(i-1)(N-u)+1, \ldots, i(N-u)+u\}, \quad i = 1, \ldots, l.$$

For the experiment, we use $l = 50$ cliques of order $N = 16$ and $m = 100$ constraints. The aggregate sparsity pattern as a function of the clique overlap u is illustrated in Fig. 3.3. Each of the data matrices A_i has roughly 10% nonzeros

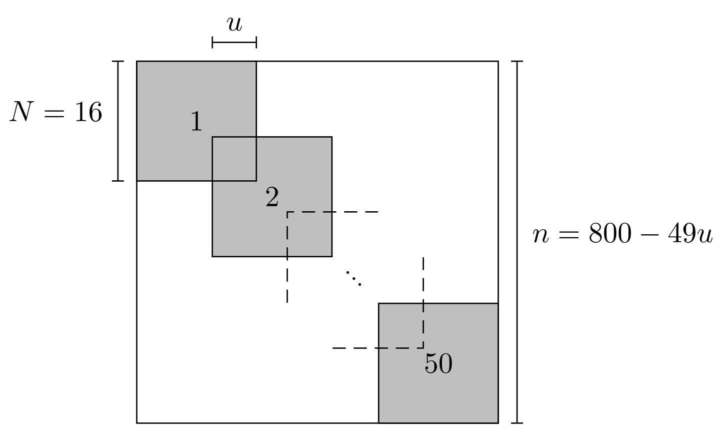


Figure 3.3: Sparsity pattern with 50 overlapping cliques of order 16. The parameter u is the overlap between adjacent cliques, i.e., $|U_i| = u$ for $i = 2, \ldots, l$.

(relative to the aggregate sparsity pattern). Table 3.9 shows the results for different values of u. Not surprisingly, all the nonchordal solvers do quite well in the block diagonal case ($u = 0$), but there is a significant jump in complexity

u	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0	0.23	0.31	0.16	47	0.32	0.12
1	0.28	0.35	79	43	3.8	12
2	0.28	0.34	100	38	3.5	9.5
4	0.26	0.32	124	31	2.9	6.5
8	0.26	0.27	55	21	1.5	2.2
15	0.12	0.08	0.08	0.60	0.12	0.05

(a) Time per iteration (sec.)

u	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
0	10	14	2.9	1027	4.2	1.7
1	11	14	3415	1035	61	181
2	12	14	4515	920	56	143
4	10	13	5817	776	43	97
8	9.7	10	2621	530	24	37
15	3.5	2.4	1.7	11	1.5	0.60

(b) Total time (sec.)

Table 3.9: Time per iteration and total time for random SDPs with $l = 50$ cliques of order $N = 16$ and $m = 100$ constraints. Neighboring cliques overlap by u nodes.

for these solvers when the cliques overlap just slightly and thereby destroy the block structure. On the other hand, SMCP-CH and SMCP-QR appear to be much less sensitive to such overlaps. We remark that SMCP-CH used only T1 for these problems. When $u = 15$, the sparsity pattern is banded and has order $n = 65$. This is a relatively small problem that can be solved quite easily with symmetric primal–dual methods.

We should point out that the conversion method by Fukuda et al. [FKM00], which also exploits chordal structure, can be a viable alternative for this type of sparsity pattern when the clique overlaps are small and when the number of cliques is not too large. The conversion method is closely related to one of the domain space decomposition methods described in §2.5.3, and it can be implemented as a preprocessing step and used in conjunction with existing primal–dual interior-point codes.

3.5.4 Robust Convex Quadratic Optimization

Sparsity patterns with block-arrow structure are chordal, and matrix inequalities with block-arrow patterns arise in several applications. Many robust counterparts of quadratically constrained quadratic programs (QCQPs) or SOCPs fall in this category [EL97, BN98, KBM96]. If the block-width of the arrow is not too large, it is often advantageous to exploit chordal structure.

Uncertain convex quadratic constraints In our first example of a robust optimization problem, we look at quadratic programs with one or more uncertain convex quadratic constrains of the form

$$x^T A^T Ax \leq 2b^T x + d. \tag{3.32}$$

Here the problem data $A \in \mathbf{R}^{p\times q}$, $b \in \mathbf{R}^q$, and $d \in \mathbf{R}$ are uncertain. If we choose as an uncertainty set a bounded ellipsoid

$$\mathcal{U} = \left\{ (\bar{A}, \bar{b}, \bar{d}) + \sum_{i=1}^{r} u_i(A_i, b_i, d_i) \;\middle|\; u^T u \leq 1 \right\} \tag{3.33}$$

where $\bar{A}, \bar{b}, \bar{d}$ are nominal values, the robust counterpart of the uncertain quadratic constraint (3.32) can be formulated as an LMI [BN98]

$$\begin{bmatrix} tI & G(x)^T & h(x) \\ G(x) & I & \bar{A}x \\ h(x)^T & (\bar{A}x)^T & f(x) - t \end{bmatrix} \succeq 0 \tag{3.34}$$

with variables $t \in \mathbf{R}$ and $x \in \mathbf{R}^q$ and where

$$\begin{aligned} G(x) &= [A_1 x \ \cdots \ A_r x], \\ h(x) &= (b_1^T x + d_1/2, \ldots, b_r^T x + d_r/2), \\ f(x) &= 2\bar{b}^T x + \bar{d}. \end{aligned}$$

The sparsity pattern associated with the LMI (3.34) has block-arrow structure, and it can be either chordal or nonchordal, depending on the structure of $G(x)$ (which, in turn, is determined by the choice of uncertainty set). It is typically worthwhile to exploit chordal sparsity in two cases. If $G(x)$ is dense and $p \gg r$ or $r \gg p$, an efficient chordal embedding can easily be constructed by filling the smaller of the two diagonal blocks. This chordal embedding will have cliques of order at most $\min(r+1, p+1)$. If on the other hand $G(x)$ is sparse, the sparsity pattern may have an efficient chordal embedding with small cliques even when $p \approx r$. As a special case we mention that if $G(x)$ has at most one structural nonzero entry in each column (or alternatively, at most one structural nonzero entry in each row), then the sparsity pattern associated with (3.34) is chordal and the cliques are of order at most three. Here we will consider a numerical

p	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
200	0.25	0.13	0.20	0.70	0.54	0.34
400	0.51	0.26	0.91	1.4	1.7	1.9
800	1.0	0.53	5.1	3.1	5.8	15.3
1600	2.1	1.2	33.6	6.5	21.6	166.9
3200	4.3	2.7	–	13.8	90.2	1416.9

(a) Time per iteration (sec.)

p	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
200	7.0	3.7	7.3	13.9	7.0	6.1
400	16.8	8.6	32.7	34.0	23.2	34.5
800	40.8	21.1	190.5	76.4	87.5	185.4
1600	80.8	47.7	1377.0	157.1	345.8	2335.9
3200	162.0	101.0	–	429.3	1352.3	19836.0

(b) Total time (sec.)

Table 3.10: Average time per iteration and total time for randomly generated uncertain QCQPs with $q = 100$ and $r = 5$.

experiment with $G(x)$ dense and $r \ll p$, and in the next experiment we look at an example where $G(x)$ is sparse and where the LMI has a chordal sparsity pattern.

In our first experiment, we are interested in the average CPU time per iteration as a function of p for randomly generated uncertain QCQPs with q, r fixed and with r small. Specifically, we minimize a linear objective $f_0(x) = c^T x$ subject to a single uncertain quadratic constraint of the form (3.32). For this experiment we choose $q = 100$, $r = 5$, and we generate problem instances with $\bar{A}, \bar{b}, A_i$ random, $b_i = 0$, $\bar{d} = 1$, and $d_i = 0$. The results are listed in Table 3.10. It is easy to verify that the average time per iteration grows roughly linearly in p for SMCP. Notice also that SDPA-C, which also exploits chordal structure, is quite fast as well. The other solvers scale quadratically or worse, and hence the benefit of exploiting chordal sparsity becomes evident for p large. We remark that DSDP crashed on the largest problem instance. Finally we note that having multiple uncertain quadratic constraints gives rise to an LMI with block-diagonal structure and with blocks of the form (3.34).

Robust least-squares Our next experiment is based on robust least-squares (RLS) which is a special case of robust QCQP. Suppose we want to minimize $\|Ax-b\|_2$ where $A \in \mathcal{U}$ is uncertain but assumed to lie in the ellipsoidal uncertainty set

$$\mathcal{U} = \{\bar{A} + u_1 A_1 + \cdots + u_r A_r \mid \|u\|_2 \leq 1\}. \tag{3.35}$$

Here $\bar{A} \in \mathbf{R}^{p \times q}$ is a known nominal coefficient matrix, $b \in \mathbf{R}^p$ is a known vector, and the matrices $A_i \in \mathbf{R}^{p \times q}$ define the structure of the set $\mathcal{U}$. The RLS problem seeks a solution $x \in \mathbf{R}^q$ that minimizes the worst-case error which is defined as

$$e_{\mathrm{wc}}(x) = \sup_{\|u\|_2 \leq 1} \|G(x)u + h(x)\|_2, \tag{3.36}$$

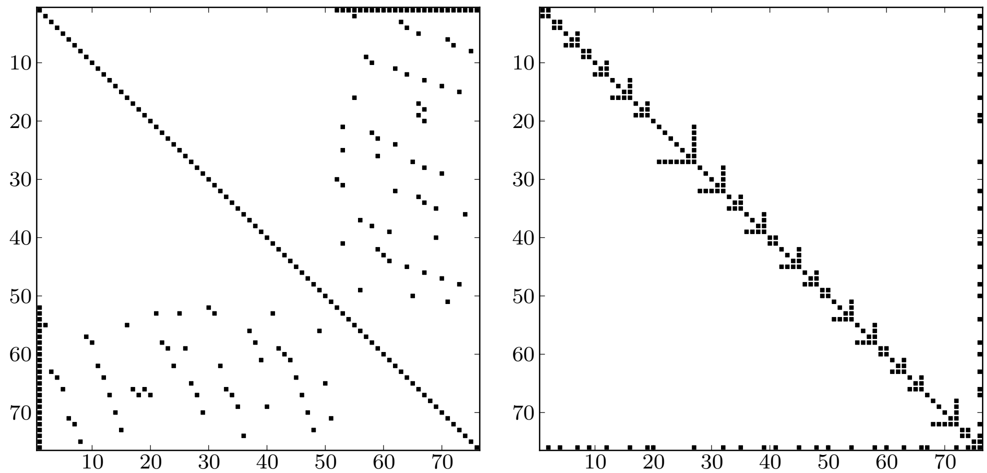
where $G(x) = [A_1 x \ \cdots \ A_r x]$ and $h(x) = \bar{A}x - b$. This problem can be cast as an SDP [EL97]:

$$\begin{array}{ll} \text{minimize} & t + \lambda \\ \text{subject to} & \begin{bmatrix} t & 0 & h(x)^T \\ 0 & \lambda I & G(x)^T \\ h(x) & G(x) & I \end{bmatrix} \succeq 0 \end{array} \tag{3.37}$$

with variables $t, \lambda \in \mathbf{R}$ and $x \in \mathbf{R}^q$. Note that the SDP (3.37) has $m = q + 2$ variables, and the constraint is an LMI of order $n = p + r + 1$.

In the following experiment we consider a family of RLS problems defined as follows. Suppose $\bar{A}$ has r uncertain entries indexed by $(i_1, j_1), \ldots, (i_r, j_r)$. Furthermore, let the matrices $A_1, \ldots, A_r$ be defined as

$$(A_k)_{ij} = \begin{cases} \gamma & i = i_k, j = j_k \\ 0 & \text{otherwise} \end{cases} \qquad k = 1, \ldots, r, \tag{3.38}$$



(a) Before reordering. (b) After reordering.

Figure 3.4: Sparsity pattern associated with an RLS problem instance with dimensions $p = 25$, $q = 10$, and $r = 50$.

where $\gamma > 0$ is a parameter that controls the size of the uncertainty set $\mathcal{U}$. The resulting SDP has a chordal sparsity pattern with $|V| = 2p + 2r + 1$ nonzeros in the lower triangle of V, and furthermore, the sparsity pattern has $p + r$ cliques of order two. An example of a sparsity pattern from a randomly generated RLS problem is shown in Fig. 3.4.

As in the previous experiment, we are interested in the computational cost per interior-point iteration as a function of p. We generate random problem instances as follows. The vector b is computed as $b = \bar{A}\bar{x} + \sigma w$ where $\bar{A}$ is a (dense) random matrix, $\bar{x}$ and w are random vectors, and σ is a positive parameter. The number of uncertain entries of $\bar{A}$ is parameterized by $d \in (0, 1]$ such that $r = \lceil pqd \rceil$ where $\lceil x \rceil$ denotes the smallest integer larger than or equal to x. The positions of the unknown entries are selected at random.

Table 3.11 shows the average CPU time per iteration (in seconds) for problem instances with different values of p and with the remaining problem parameters

p	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	0.19	0.14	0.20	0.18	1.0	4.8
200	0.41	0.31	0.81	0.62	4.8	54.5
400	0.82	0.64	4.4	2.8	27.6	512.5
800	1.7	1.4	25.2	11.1	125.2	–
1600	3.6	3.2	153.6	45.3	–	–

(a) Time per iteration (sec.)

p	SMCP-CH	SMCP-QR	DSDP	SDPA-C	SDPT3	SEDUMI
100	7.4	5.2	11.5	3.9	15.1	86.0
200	24.4	18.3	91.5	26.9	96.1	1690.3
400	67.1	52.1	672.0	136.5	606.1	14350.0
800	190.5	161.8	5549.0	587.3	2755.1	–
1600	572.7	504.4	45780.0	2987.2	–	–

(b) Total time (sec.)

Table 3.11: Average time per iteration (seconds) for randomly generated RLS problems with $q = 100$ and $r = 5p$.

fixed ($q = 100$ and $d = 0.05$). Notice once again that for SMCP the time per iteration grows roughly linearly with the parameter p. The other solvers scale quadratically or worse, and for the largest instance, the general purpose solver SDPT3 ran out of memory. Note also that in this example, SDPA-C does not scale as well as in the previous experiment.

3.5.5 Sparse SDPs from SDPLIB

Our next experiment is based on a set of problem instances from SDPLIB [Bor99b]. We include only the largest and most sparse problem instances (specifically, sparse problems with $n \geq 500$), since these are the most interesting problems in regard to chordal matrix techniques. Before we present and discuss our results, we briefly give some definitions pertaining to problem dimension and data sparsity.

Suppose V is a sparsity pattern with k diagonal blocks of order $n_1, \ldots, n_k$, where block i has sparsity pattern V_i. In other words, $\mathbf{S}^n_V = \mathbf{S}^{n_1}_{V_1} \times \cdots \times \mathbf{S}^{n_k}_{V_k}$ and $n = \sum_{i=1}^k n_i$. We denote with $n_{\max} = \max_i n_i$ the order of the largest block, and we define the *density* of V as

$$\rho_V = \frac{2|V| - n}{\sum_{i=1}^k n_i^2}.$$

(Recall that $|V|$ is the number of lower triangular nonzeros in the sparsity pattern.) Note that $\rho_V = 1$ corresponds to dense blocks in which case the chordal techniques become trivial. Similarly, we define the *average density* of the problem data, relative to V, as

$$\bar{\rho}_{V,\mathrm{rel}} = \frac{1}{m} \sum_{i=1}^{m} \frac{\mathrm{nnz}(A_i)}{2|V| - n},$$

where $\mathrm{nnz}(A_i)$ is the number of nonzeros in A_i. This is a measure of the sparsity of the coefficients A_i in the subspace $\mathbf{S}^n_V$. If $\bar{\rho}_{V,\mathrm{rel}} = 1$, the coefficient matrices are all dense relative to $\mathbf{S}^n_V$. Note that SMCP-QR is generally inefficient when $\bar{\rho}_{V,\mathrm{rel}}$ is small since SMCP-QR treats A_i as a dense element in $\mathbf{S}^n_V$.

For a chordal sparsity pattern with l cliques, we define the following two measures:

$$\mathcal{W} = \sum_{i=1}^{l} |W_i| \qquad \text{and} \qquad \mathcal{U} = \sum_{i=1}^{l} |U_i|,$$

i.e., $\mathcal{W}$ is the sum of clique cardinalities, and $\mathcal{U}$ is the sum of separator cardinalities. The sum of residual cardinalities is given by $\sum_{i=1}^{l} |V_i| = \mathcal{W} - \mathcal{U} = n$, and hence does not carry any information about the sparsity pattern. The measure $\mathcal{U}$ can be thought of as the total overlap between cliques, where $\mathcal{U} = 0$ corresponds to nonoverlapping cliques, i.e., V is block diagonal with dense blocks. Finally, we denote with $w_{\max}$ order of the maximum clique in V (the clique number).

The set of SDPLIB problems with $n \geq 500$ is listed in Table 3.12 with selected problem statistics. With the exception of one problem (`truss8`), all problem

Instance	Dimensions m	n	k	$n_{\max}$	Sparsity (%) ρ_V	$\bar{\rho}_{V,\text{rel}}$
maxG11	800	800	1	800	0.62	0.025
maxG32	2,000	2,000	1	2,000	0.25	0.010
maxG51	1,000	1,000	1	1,000	1.28	0.008
maxG55	5,000	5,000	1	5,000	0.14	0.003
maxG60	7,000	7,000	1	7,000	0.08	0.002
mcp500-1	500	500	1	500	0.70	0.057
mcp500-2	500	500	1	500	1.18	0.034
mcp500-3	500	500	1	500	2.08	0.019
mcp500-4	500	500	1	500	4.30	0.009
qpG11	800	1,600	1	1,600	0.19	0.042
qpG51	1,000	2,000	1	2,000	0.35	0.014
thetaG11	2,401	801	1	801	0.87	0.113
thetaG51	6,910	1,001	1	1,001	1.48	0.053
truss8	496	628	34	19	100.00	0.270

Table 3.12: Problem statistics for SDPLIB problems.

instances have a nonchordal aggregate sparsity pattern. The problem `truss8` clearly has a block diagonal aggregate sparsity pattern with dense blocks (since $\rho_V = 1$). This implies that the chordal techniques are trivial and have no advantage over existing solvers that handle block diagonal structure. Notice also that $\bar{\rho}_{V,\text{rel}}$ is small for all problem instances, and as a consequence, SMCP-QR can be expected to be quite slow since it does not fully exploit the sparsity of the individual data matrices. Finally, we remark that all the problem instances in Table 3.12 have data matrices with very low rank.

Table 3.13 shows some statistics for two embeddings: a standard AMD embedding and an AMD embedding obtained via CHOLMOD [CDH08] which includes a post-processing of the clique tree. CHOLMOD's embedding clearly has fewer but larger cliques, and therefore the density ρ_V is somewhat larger. We will refer to Method 1 based on CHOLMOD's AMD embedding as SMCP-CM.

If we compare the density of the aggregate sparsity patterns (Table 3.12)

Instance	l	$w_{\max}$	$\mathcal{W}$	$\mathcal{U}$	ρ_V (%)
maxG11	598	24	4,552	3,752	2.48
maxG32	1,498	76	12,984	10,984	1.81
maxG51	674	326	14,286	13,286	13.41
maxG55	3,271	1,723	77,908	72,908	12.55
maxG60	5,004	1,990	100,163	93,163	8.51
mcp500-1	452	39	1,911	1,411	2.07
mcp500-2	363	138	4,222	3,722	10.74
mcp500-3	259	242	6,072	5,572	27.99
mcp500-4	161	340	8,420	7,920	52.64
qpG11	1,398	24	5,352	3,752	0.65
qpG51	1,674	326	15,286	13,286	3.38
thetaG11	598	25	5,150	4,349	2.72
thetaG51	676	324	14,883	13,882	13.41
truss8	34	19	628	0	100.00

(a) AMD embedding

Instance	l	$w_{\max}$	$\mathcal{W}$	$\mathcal{U}$	ρ_V (%)
maxG11	134	32	1,864	1,064	4.92
maxG32	253	79	5,348	3,348	3.12
maxG51	129	337	4,563	3,563	20.81
maxG55	728	1,776	28,421	23,421	15.30
maxG60	1,131	2,048	35,817	28,817	10.27
mcp500-1	127	51	860	360	5.55
mcp500-2	99	146	1,545	1,045	18.71
mcp500-3	58	251	2,196	1,696	42.08
mcp500-4	32	352	2,608	2,108	68.60
qpG11	934	32	2,664	1,064	1.26
qpG51	1,129	337	5,563	3,563	5.23
thetaG11	134	33	1,998	1,197	5.16
thetaG51	127	335	4,707	3,706	20.97
truss8	34	19	628	0	100.00

(b) CHOLMOD-AMD embedding

Table 3.13: Statistics for two chordal embeddings of selected sparsity patterns from SDPLIB.

with the density of the chordal embeddings (Table 3.13), we see that for some of the problems, the chordal embeddings are much more dense than the aggregate sparsity pattern. This means that the chordal embeddings are not very efficient. Indeed, this seems to be the case for many of the problems, in particular, `maxG55`, `maxG60`, `mcp500-3`, and `mcp500-4`. Notice also that the chordal embeddings of `maxG55` and `maxG60` have fairly large clique numbers.

From the results in Table 3.14, we see that the performance of the chordal techniques varies quite a bit when compared to the nonchordal solvers. As expected, the problems `maxG55`, `maxG60`, `mcp500-3`, and `mcp500-4` are not favorable to the chordal techniques. Problems with more efficient chordal embeddings and smaller maximum clique sizes, such as `maxG11` and `maxG32`, are solved more efficiently. If we compare SMCP-CH and SMCP-CM, it is readily seen that the chordal embedding obtained with CHOLMOD consistently results in faster iteration times. Other embedding techniques can be used and may further improve the speed. Notice that DSDP does quite well on most problems, and this may partially be due to DSDP's use of low rank techniques. Furthermore, DSDP solves the Schur complement equations using preconditioned conjugate gradient which is often more efficient, especially when m is large and so long as the condition number of the Schur complement is not too large. The chordal techniques do not exclude the use of low rank techniques or iterative solution of the Newton system, so implementing these techniques may also improve SMCP-CH/SMCP-CM.

Finally we remark that for the problem `thetaG51`, all the solvers stopped prematurely because of numerical problems, and for the problems `maxG55` and `maxG60`, SDPA-C spent more than 50% of the total CPU time outside the main loop.

Instance	SMCP-CH	SMCP-CM	DSDP	SDPA-C	SDPT3	SEDUMI
maxG11	0.74	0.47	0.24	0.48	0.98	13
maxG32	5.1	3.3	3.2	5.1	9.5	339
maxG51	8.3	5.4	0.64	2.8	2.0	31
maxG55	1169	650	73	852	149	m
maxG60	2301	1074	170	1883	428	m
mcp500-1	0.33	0.18	0.07	0.11	0.29	3.1
mcp500-2	1.3	0.77	0.10	0.33	0.37	3.3
mcp500-3	3.6	2.0	0.14	0.86	0.42	3.3
mcp500-4	13	4.4	0.20	1.7	0.47	3.4
qpG11	1.1	0.76	0.38	0.77	0.97	158
qpG51	12	8.4	0.97	4.6	1.9	308
thetaG11	4.2	3.2	3.2	3.4	2.7	21
thetaG51	59	52	94	85	33	301
truss8	0.60	0.60	0.14	2.6	0.17	0.13

(a) Time per iteration (sec.)

Instance	SMCP-CH	SMCP-CM	DSDP	SDPA-C	SDPT3	SEDUMI
maxG11	23	16	5.0	11	15	169
maxG32	154	118	70	133	142	4743
maxG51	282	196	19	76	34	498
maxG55	45598	28590	2618	22146	2539	m
maxG60	92029	49398	6131	45194	6848	m
mcp500-1	9.5	6.5	1.9	2.3	4.4	50
mcp500-2	32	19	2.2	6.9	5.9	50
mcp500-3	109	64	2.7	18	5.9	50
mcp500-4	349	124	4.2	37	6.1	47
qpG11	33	30	12	18	15	2214
qpG51	682	488	95	252	32	6775
thetaG11	248	178	200	67	49	317
thetaG51	2901	2676	5662	5726	1223	5422
truss8	26	26	3.7	87	2.8	3.1

(b) Total time (sec.)

Table 3.14: Average time per iteration and total time for selected SDPLIB problems. Failure due to insufficient memory is marked with an 'm'. SMCP-CH and SMCP-CM required a phase I for the problems thetaG11, thetaG51, and truss8 (time not included).

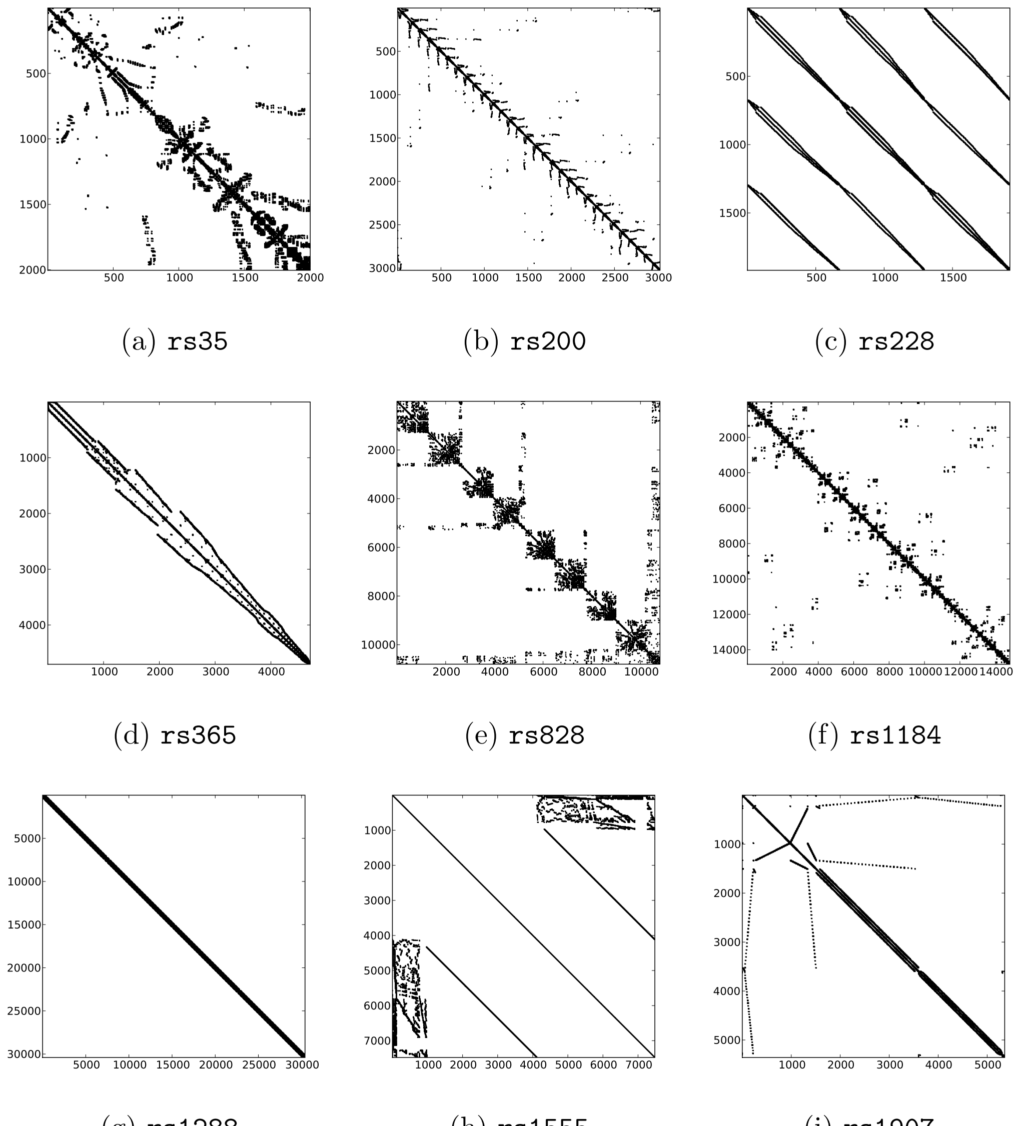
3.5.6 Nonchordal SDPs

In our last experiment we consider SDPs with nonchordal aggregate sparsity patterns and random data. Each problem is based on a sparsity pattern from the University of Florida Sparse Matrix Collection (UFSMC) [DV09b]. We will use as problem identifier the name `rsX`, where `X` is the ID number associated with the corresponding problem from UFSMC. Fig. 3.5 shows a selection of nine nonchordal sparsity patterns used in this experiment.

For each of the sparsity patterns, we generated a problem instance with average density $\bar{\rho}_{V,\mathrm{rel}} = 10^{-3}$ and $m = 200$ constraints. Table 3.15 lists problem statistics and Table 3.16 shows some statistics for two different chordal embeddings. All nine problems have a single block, and therefore, for the largest of the problems, handling the primal variable as a dense matrix is prohibitively expensive in terms of both computations and memory. However, for all nine problems, the chordal embeddings have clique numbers that are much smaller than n. Notice that CHOLMOD's AMD embedding generally has significantly fewer cliques than the AMD embedding, and its density is only slightly higher. The chordal embeddings have between 3-10 times as many nonzeros as the corresponding

	Dimensions				Sparsity (%)	
Instance	m	n	k	$n_{\max}$	ρ_V	$\bar{\rho}_{V,\mathrm{rel}}$
`rs35`	200	2,003	1	2,003	2.09	0.05
`rs200`	200	3,025	1	3,025	0.23	0.05
`rs228`	200	1,919	1	1,919	0.88	0.05
`rs365`	200	4,704	1	4,704	0.47	0.05
`rs828`	200	10,800	1	10,800	0.69	0.05
`rs1184`	200	14,822	1	14,822	0.33	0.05
`rs1288`	200	30,401	1	30,401	0.05	0.05
`rs1555`	200	7,479	1	7,479	0.12	0.05
`rs1907`	200	5,357	1	5,357	0.72	0.05

Table 3.15: Problem statistics for nonchordal problems.



(a) `rs35` (b) `rs200` (c) `rs228`

(d) `rs365` (e) `rs828` (f) `rs1184`

(g) `rs1288` (h) `rs1555` (i) `rs1907`

Figure 3.5: Aggregate sparsity patterns for nonchordal test problems.

Instance	l	$w_{\max}$	$\mathcal{W}$	$\mathcal{U}$	ρ_V (%)
`rs35`	589	343	27,881	25,878	13.21
`rs200`	1,632	95	20,063	17,038	1.51
`rs228`	790	88	17,244	15,325	3.60
`rs365`	1,230	296	36,136	31,432	2.54
`rs828`	841	480	74,256	63,456	2.19
`rs1184`	2,236	500	172,046	157,224	2.28
`rs1288`	10,394	412	222,706	192,305	0.32
`rs1555`	6,891	184	49,447	41,968	0.28
`rs1907`	577	261	30,537	25,180	2.97

(a) AMD embedding

Instance	l	$w_{\max}$	$\mathcal{W}$	$\mathcal{U}$	ρ_V (%)
`rs35`	141	394	11,819	9,816	14.66
`rs200`	314	95	8,711	5,686	2.12
`rs228`	207	88	7,333	5,414	4.35
`rs365`	396	296	19,417	14,713	2.87
`rs828`	605	480	56,364	45,564	2.25
`rs1184`	830	500	91,966	77,144	2.36
`rs1288`	2,295	412	108,865	78,464	0.38
`rs1555`	2,371	193	23,614	16,135	0.55
`rs1907`	400	261	24,974	19,617	3.10

(b) CHOLMOD-AMD embedding

Table 3.16: Statistics for AMD and CHOLMOD's AMD embeddings of nonchordal problems.

aggregate sparsity patterns.

The results are listed in Table 3.17. For three of the four smallest problem, DSDP is the fastest in terms of average time per iteration. The results show

Instance	SMCP-CH	SMCP-CM	DSDP	SDPA-C	SDPT3	SEDUMI
rs35	37	30	4.0	41	13	325
rs200	3.2	2.4	2.9	5.9	33	1139
rs228	3.9	3.0	1.4	5.7	11	282
rs365	30	28	15	59	100	m
rs828	406	392	482	1813	m	m
rs1184	729	677	791	2925	m	m
rs1288	386	362	m	2070	m	m
rs1555	14	11	22	23	m	m
rs1907	49	48	38	176	152	m

(a) Time per iteration (sec.)

Instance	SMCP-CH	SMCP-CM	DSDP	SDPA-C	SDPT3	SEDUMI
rs35	1632	1377	205	1245	234	5530
rs200	171	142	117	196	727	19367
rs228	187	139	71	181	214	5914
rs365	1473	1368	850	1940	1993	m
rs828	21107	20401	27000	63448	m	m
rs1184	43010	37897	47460	105313	m	m
rs1288	23548	20621	m	74528	m	m
rs1555	641	484	1062	648	m	m
rs1907	2531	2192	2004	5445	2888	m

(b) Total time (sec.)

Table 3.17: Average time per iteration and total time for nonchordal SDPs with random data. Failure due to insufficient memory is marked with an 'm'.

that the chordal techniques can be quite fast for large problems, even when the chordal embedding has many more nonzeros than the aggregate sparsity pattern. If we compare the results obtained with SMCP-CH and SMCP-CM, we see that CHOLMOD's chordal embedding is advantageous in terms of iteration time, but the difference in iteration time is typically smaller than for many of the SDPLIB problems. Both SMCP-CH and SMCP-CM used only T2 to compute the Schur complement matrix in all cases.

The general purpose primal–dual solvers SDPT3 and SEDUMI ran of memory while solving the largest of the problems, and while DSDP successfully solved the largest instance `rs1288`, it ultimately ran out of memory in an attempt to compute the dense primal variable.

3.6 Summary

We have described implementations of a primal and a dual path-following algorithm that take advantage of chordal sparsity when solving the Newton equations. Two different methods were implemented for solving the Newton equations. The first method forms and solves the Schur complement system via a Cholesky factorization. This method can exploit the sparsity of the individual data matrices more aggressively, but it is also less stable than the second method. The second method avoids the explicit construction of the Schur complement matrix, by solving the augmented system via a QR decomposition. Although the augmented system approach is rarely practical in general semidefinite programming because of its high cost (despite its better numerical stability), it can be viable for matrix cones of relatively low dimension. The two methods can be used in any interior-point method based on primal or dual scaling, including barrier and potential reduction methods, infeasible path-following methods, and Nesterov's nonsymmetric primal–dual methods [Nes06b].

The results of the experiments indicate that if the sparsity patterns are chordal, the nonsymmetric chordal methods are often significantly faster and can solve larger instances than the general-purpose semidefinite programming solvers.

For problems with a band or block-arrow sparsity, the complexity of the chordal methods increases linearly with the matrix dimension (for fixed bandwidth or block-width), while the complexity of the other interior-point SDP solvers increases quadratically or faster. For general (nonchordal) sparsity patterns, the performance depends on the efficiency of the chordal embedding. If no efficient chordal embedding can be found, the chordal techniques are generally not advantageous, although in many cases, their efficiency is still comparable with the primal–dual interior-point solvers.

CHAPTER 4

Applications in Nonlinear Programming

In this chapter, we investigate other applications of chordal matrix techniques in nonlinear optimization. We consider maximum determinant positive definite completion techniques as a means to approximate or compress large, dense, positive definite matrices. We know from §2.3.4 that this type of approximation yields a positive definite approximation (if it exists) whose inverse is sparse, and it can be computed efficiently when the associated sparsity pattern is chordal. Sparse inverse approximations have been used in numerical linear algebra and optimization as preconditioners for iterative methods. Our focus, however, is on kernel-based methods in machine learning that involve large, dense kernel matrices.

We start the chapter with a discussion of techniques that can be used to approximate the inverse of a matrix by a sparse matrix. To gain insight into the accuracy of the completion-based approximations, we will study the spectrum of a "preconditioned matrix" R^TAR where R^TR is the Cholesky factorization of a sparse approximate inverse of A. We then discuss an implementation of an interior-point method for nonlinear SVM training that relies on a sparse inverse approximation of a large, dense kernel matrix.

4.1 Sparse Approximate Inverses

A sparse inverse approximation of a positive definite matrix A is a matrix M that is close to A^{-1} in some sense. The work on sparse inverse approximations has mainly focused on preconditioners for iterative methods such as the conjugate gradient method. The convergence rate of the conjugate gradient method is largely determined by the spectral properties of the coefficient matrix, and improved efficiency and robustness can be obtained by transforming the system of equations $Ax = b$ into a preconditioned system of equations $MAx = Mb$ that has the same solution as the original problem. The matrix M, which is an inverse preconditioner of A, should ideally satisfy the following requirements: it should be close to A^{-1} in some sense, and matrix–vector products Mz as well as M itself should be cheap to compute. Clearly, these are conflicting requirements in all but trivial cases: if M is equal to A^{-1}, the cost of products with M is the same as that of solving the original system, and if Mz is easy to evaluate (e.g. if M is diagonal), M is typically a very poor approximation of A^{-1}. This means that some kind of trade-off is necessary when choosing the preconditioner M. Furthermore, some iterative methods (e.g., the conjugate gradient algorithm) require a positive definite preconditioner.

The sparsity pattern of the inverse of an irreducible matrix is structurally full. This can be verified using the Cayley-Hamilton theorem, which states that a square matrix A satisfies its own characteristic equation, i.e., $p_A(A) = 0$ where $p_A(t) = \det(tI - A)$ is the characteristic polynomial associated with A. This implies that [Che05, p. 192]

$$A^{-1} \in \operatorname{span}\{I, A, A^2, \ldots, A^{n-1}\} \tag{4.1}$$

and consequently, the sparsity pattern of A^{-1} is a subset of the sparsity pattern of $(I + A)^{n-1}$. It is therefore not obvious how a sparse approximate inverse can yield a good approximation. However, if A^{-1} has many small or numerical zero entries, a sparse inverse approximation may be justified in some cases. The above observations suggest that the sparsity pattern associated with M has a significant effect on the quality of M as an approximate inverse of A. For general matrices, however, the locations of the "important" entries in A^{-1} are usually unknown, and this makes it difficult to choose a suitable sparsity pattern for M.

As an example of an inverse type preconditioning technique, we highlight the pioneering approach proposed by Benson and Frederickson [Ben73, BF82] which seeks a matrix M that minimizes $\|I - AM\|_F^2$ subject to sparsity constraints on M. Many variations have since been proposed, including adaptive versions that try to automatically find a suitable sparsity pattern for M. The sparse approximate inverse algorithm [GH97], or SPAI, is an example of such an algorithm. Another variant of Benson's method is the factorized sparse approximate inverse [KY93], or FSAI, proposed by Kolotilina in 1993. Both the SPAI and the FSAI algorithm work with general nonsymmetric coefficient matrices, and when A is symmetric positive definite, an algorithm similar to FSAI can be used with the preconditioned conjugate gradient (PCG) algorithm [BMT96]. An overview of sparse inverse type preconditioners can be found in [CS94], and a comparative study is presented in [BT99].

4.1.1 Optimization Approach

Benson's idea to use $\|I - AM\|_F^2$ as the optimization objective in the search for an approximate inverse is both elegant and simple. In its simplest form, the problem is separable and can be solved as n independent least-squares problems, one for each column in M, i.e., the objective can be decomposed as

$$\|I - AM\|_F^2 = \sum_{i=1}^{n} \|e_i - Am_i\|_2^2$$

where e_i is the ith column of the identify matrix, and m_i is the ith column of M. Using the unitary invariance property of the Frobenius norm, it is easy to see that

$$\|I - AM\|_F^2 = \sum_{i=1}^{n} |1 - \lambda_i|^2 \tag{4.2}$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of the preconditioned matrix AM. Note that M may not be positive definite nor symmetric, even if A is symmetric positive definite.

Like Benson and Frederickson's method, the sparse inverse approximation that we obtain by solving the dual (2.7) of the maximum determinant positive definite completion problem can be interpreted as a matrix nearness problem. Instead of minimizing the objective $\|I - AM\|_F$, we use the so-called Burg entropy as a measure of the similarity of the two matrices M^{-1} and A. The Burg matrix entropy is defined as

$$D(X; Y) = X \bullet Y^{-1} - \log\det(XY^{-1}) - n \tag{4.3}$$

and it is a nonsymmetric measure of the difference between two symmetric positive definite matrices X and Y. (It is closely related to the Kullback-Leibler divergence, or *relative entropy*, of two zero-mean multivariate normal distributions.) The Burg entropy is zero if and only if $X = Y$ in (4.3). Rewriting $D(A; M^{-1})$ in terms of the eigenvalues $\lambda_1, \ldots, \lambda_n$ of $M^{1/2}AM^{1/2}$ yields

$$D(A; M^{-1}) = \sum_{i=1}^{n} (\lambda_i - \log(\lambda_i) - 1) \,. \tag{4.4}$$

If we compare (4.4) to (4.2), we see that the Burg entropy assigns a high penalty to small eigenvalues of $M^{1/2}AM^{1/2}$, and more importantly, the preconditioner is guaranteed to be symmetric positive definite. Furthermore, the completion preconditioner can be computed efficiently if V is a chordal sparsity pattern.

In related work, Yamashita [Yam08] proposes a technique for sparse quasi-Newton updates based on matrix completion.

4.1.2 Block-arrow Completion

We now study a special case of the completion approximation. If we restrict the sparsity pattern of M to be of block-arrow type with block-width p, we obtain a factorization $M = RR^T$ where R is of the form

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

with $R_{11} \in \mathbf{S}^p_{++}$, $R_{12} \in \mathbf{R}^{p \times (m-p)}$, and R_{22} is a positive diagonal matrix of order $m - p$. We will now show that the effect of this type of approximation is equivalent to implicit preconditioning of the Schur complement system. Gondzio [Gon09] has considered this type of preconditioner for the (regularized) Schur complement equations that arise in an interior-point method for convex quadratic optimization.

The inverse of R has the same sparsity pattern as R, and since the completion satisfies $(M^{-1})_{ij} = A_{ij}$ for all $(i,j) \in V$, we have

$$A - R^{-T}R^{-1} = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} - \begin{bmatrix} R_{11}^{-T} & 0 \\ F^T & R_{22}^{-1} \end{bmatrix} \begin{bmatrix} R_{11}^{-1} & F \\ 0 & R_{22}^{-1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & E \end{bmatrix} \qquad (4.5)$$

where $F = -R_{11}^{-1}R_{12}R_{22}^{-1}$ and $E = A_{22} - F^TF - R_{22}^{-2}$ with $\mathbf{diag}(E) = 0$. We immediately see from (4.5) that $A_{11} = R_{11}^{-T} R_{11}^{-1}$ and $F^T = A_{21} R_{11}$, and hence

$$A_{22} - F^T F = A_{22} - A_{21} A_{11}^{-1} A_{21}^T$$

where $S = A_{22} - A_{21} A_{11}^{-1} A_{21}^T$ is the Schur complement of A_{11} in A. Since the diagonal of E is zero, the diagonal matrix R_{22}^{-2} must be equal to the diagonal part of the Schur complement S, or equivalently,

$$\mathbf{diag}(R_{22}) = \mathbf{diag}(S)^{-1/2}.$$

The factorization $M = R^{-T} R^{-1}$ is closely related to the following explicit block factorization of A as

$$\begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21} A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{11}^{-T} A_{21}^T \\ 0 & I \end{bmatrix}.$$

To see this, let $A_{11} = L_{11} L_{11}^T$ be the Cholesky factorization of A_{11} and define $L_{21} = A_{21} L_{11}^{-T}$. It is easy to verify that $L_{11} = R_{11}^{-T}$ and $L_{21} = F^T$, and hence

$$\begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & B \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & B^{-1} S B^{-1} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & B \end{bmatrix} \tag{4.6}$$

for any nonsingular $B \in \mathbf{R}^{(m-p)\times(m-p)}$. The block lower triangular factor in (4.6) is therefore equal to R^{-T} if $B = R_{22}^{-1}$. This implies that the preconditioned matrix $R^T A R$ is of the form

$$R^T A R = \begin{bmatrix} I & 0 \\ 0 & R_{22} S R_{22} \end{bmatrix} \tag{4.7}$$

and moreover, $R^T A R$ has the eigenvalue 1 with multiplicity at least p. Notice that the $(2,2)$ block of the preconditioned matrix in (4.7) is the Schur complement S, preconditioned by its own diagonal. The block-arrow completion preconditioner can therefore be thought of as an implicit Jacobi preconditioner for the corresponding Schur complement equations.

We remark that the block-arrow completion approximation can be computed either by solving $P_V(M^{-1}) = P_V(A)$ using the recursive algorithm described previously, or alternatively, M^{-1} can be factored directly as $M = LL^T$, for example, using a partial Cholesky factorization. Generally this is not the case for any chordal sparsity pattern, but it is easy to show that in the block-arrow case, the sparsity pattern of R is identical to that of L^T.

To test the accuracy of the block-arrow completion approximation, we generate some random dense symmetric positive definite matrices with a prescribed spectrum as follows. First we generate a random orthogonal matrix Q. This can be done using a simple method based on the QR decomposition of an $m \times m$ matrix whose elements are drawn independently from a standard normal distribution [Ste80]. Second, we form $A = Q\,\mathbf{diag}(\lambda)Q^T$ where $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a vector with the desired eigenvalues.

In the first example, we generate a random matrix A of order 1000 whose eigenvalues decay at an exponential rate. Fig. 4.1 shows the spectrum and the condition number of $R^T AR$ for sparse inverse approximations with different block-widths p. It is easy to verify that the eigenvalue 1 has multiplicity p and that and the condition number decreases slowly as the block-width is increased.

In our second example we introduce a gap in the spectrum after the first 50 eigenvalues, and to keep the condition number fixed, we reduce the rate at which the eigenvalues decay. Fig. 4.2 shows the spectrum and condition number of $R^T AR$. Notice that the condition number increases initially for small block-widths ($p = 0$ corresponds to a diagonal preconditioner). However, when the block-width is larger than approximately 55, the condition number drops sharply. This example demonstrates that the block-width p is an important parameter. Choosing p too low may increase the condition number in some cases.

Not surprisingly, the value of p at which the condition number starts to drop is closely related to the gap in the spectrum of A.

4.1.3 Band Completion

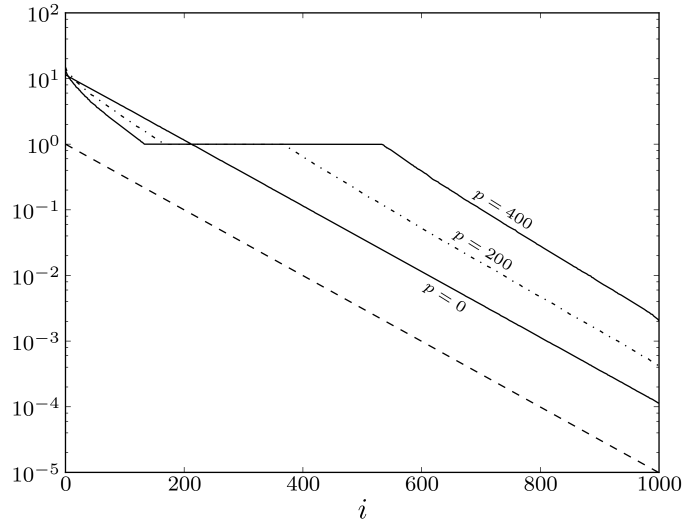
Another special case of the completion approximation is the band completion approximation, i.e., M is banded with bandwidth $2p + 1$. The band completion preconditioner reduces to a diagonal preconditioner when $p = 0$, so it can be thought of as a generalization of the diagonal preconditioner. Recall that since M is banded, the completion M^{-1} is a semiseparable matrix with semiseparability rank p, and hence the half-bandwidth parameter p can be seen as an upper bound on the rank of the submatrices in M^{-1} below its pth superdiagonal. It is easy to show that the preconditioned matrix $R^T AR$ satisfies

$$\mathbf{diag}(R^T AR) = \mathbf{1}$$

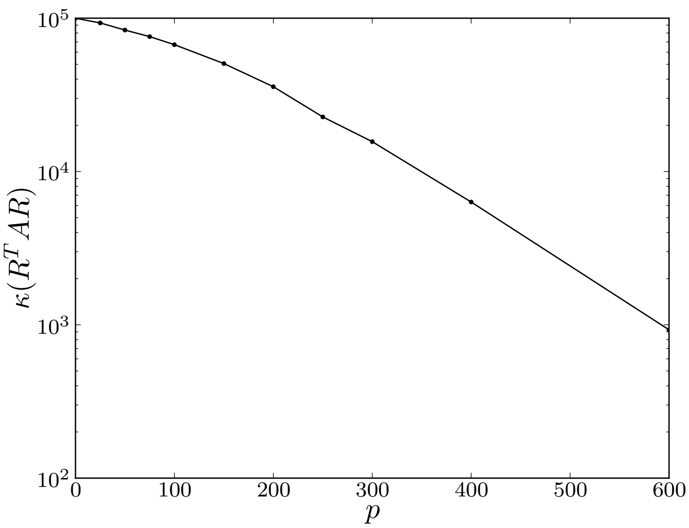
and the leading principal minor of order $p + 1$ of $R^T AR$ is the identity matrix.

We now repeat the simple experiments with random matrices to show the effect of the band completion preconditioner. The first experiment tests the band completion preconditioner on a random matrix A of order 1000 and whose eigenvalues decay at an exponential rate. The spectrum and the condition number of the preconditioned matrix $R^T AR$ are shown in Fig. 4.3 for different values of p. Notice that the condition number drops very slowly as p is increased, but the preconditioner does not introduce p repeated eigenvalues like the block-arrow preconditioner (compare with the plot in Fig. 4.1).

In the second experiment we reintroduce a gap in the spectrum of A. Fig. 4.4 shows the spectrum and the condition number of $R^T AR$ for different bandwidths. For small bandwidths, the condition number increases steadily and it peaks near

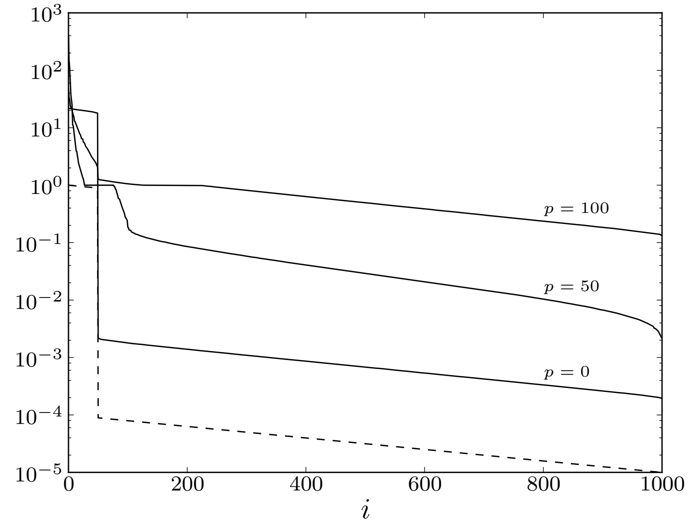


(a)

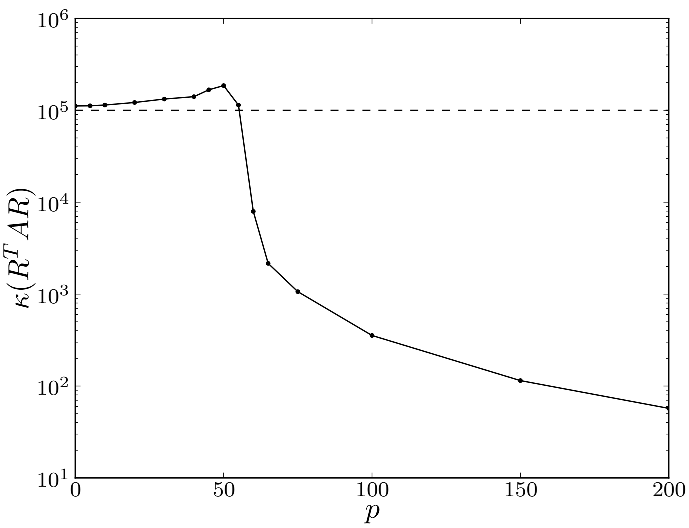


(b)

Figure 4.1: Spectrum (a) and condition number (b) of preconditioned matrix $R^T AR$ where RR^T is a positive definite block-arrow approximation of A^{-1} with block-width p. The spectrum of A is shown with a dashed line.



(a)



(b)

Figure 4.2: Spectrum (a) and condition number (b) of preconditioned matrix $R^T AR$ where RR^T is a positive definite block-arrow approximation of A^{-1} with block-width p. The spectrum of A is shown with a dashed line.

$p = 45$. For p greater than approximately 50, the condition number decreases rapidly. The effectiveness of the band completion preconditioner clearly depends on p, and p must be chosen sufficiently large in order for the preconditioner have the desired effect.

4.2 Applications in Machine Learning

Kernel methods typically require the solution of large, dense optimization problems. Perhaps the best known example of this is the quadratic program (QP) that arises in the dual formulation of the SVM training problem:

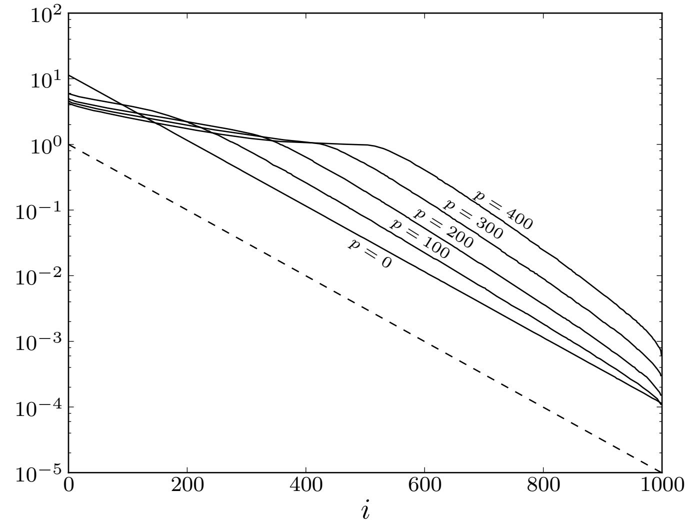
$$\begin{array}{ll} \text{maximize} & -(1/2)z^T Q z + d^T z \\ \text{subject to} & 0 \preceq \mathbf{diag}(d) z \preceq \gamma \mathbf{1} \\ & \mathbf{1}^T z = 0. \end{array} \tag{4.8}$$

The variable in this problem is $z \in \mathbf{R}^m$ where m is the number of training examples. The vector $d \in \{-1, 1\}^m$ contains the labels of the training examples, $\mathbf{1}$ is the m-vector with elements equal to one, and $\gamma > 0$ is a positive parameter. The inequalities denote componentwise inequality. The matrix Q in the objective is called the *kernel matrix*, and it is defined as

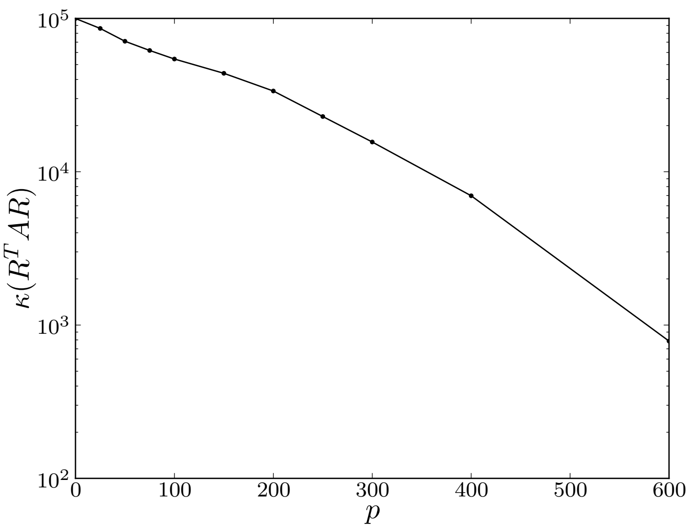
$$Q_{ij} = h(x_i, x_j), \quad i, j = 1, \ldots, m,$$

where h is a positive semidefinite kernel function and $x_1, \ldots, x_m$ are the training examples. Important kernel functions on $\mathbf{R}^n \times \mathbf{R}^n$ are the *linear* kernel function $h(u, v) = u^T v$, the *radial basis function* (RBF) kernel $h(u, v) = \exp(-\|u - v\|_2^2/(2\sigma))$, and the *polynomial* kernel $h(u, v) = (u^T v)^\delta$.

The QP (4.8) is a convex optimization problem and can be solved by standard methods for convex optimization, such as interior-point methods [NW06, ch.16]. This requires knowledge of the entire kernel matrix and, at each iteration,

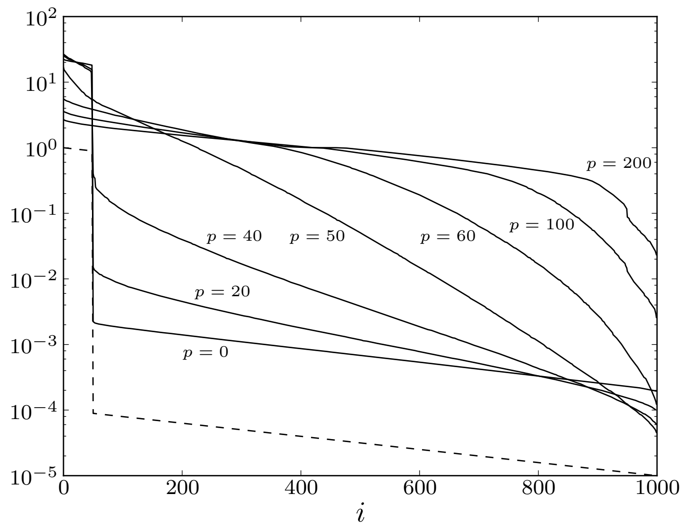


(a)

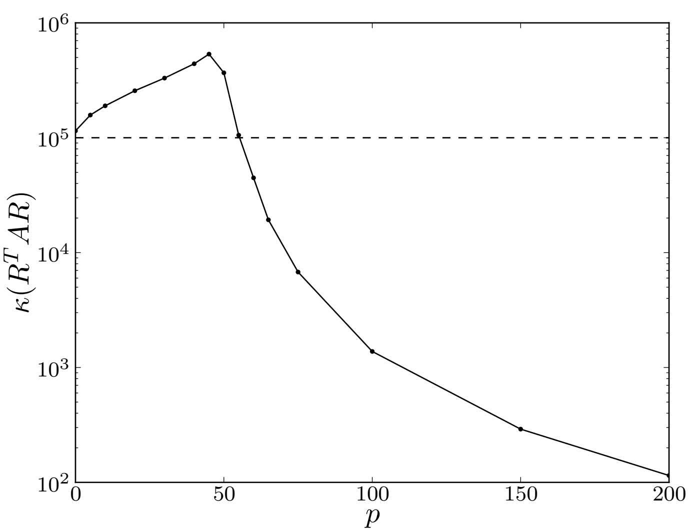


(b)

Figure 4.3: Spectrum (a) and condition number (b) of preconditioned matrix R^TAR where RR^T is a positive definite band approximation of A^{-1} with half-bandwidth p. The spectrum of A is shown with a dashed line. The condition number $\kappa(R^TAR)$ decreases slowly as the half-bandwidth p increases.



(a)



(b)

Figure 4.4: Spectrum (a) and condition number (b) of preconditioned matrix $R^T AR$ where RR^T is a positive definite band approximation of A^{-1} with half-bandwidth p. The spectrum A is shown with a dashed line. Notice that there is a gap in the spectrum of A after the first 50 eigenvalues. Notice also that $\kappa(R^T AR)$ is larger than $\kappa(A)$ (dashed line) for small p whereas $\kappa(R^T AR)$ drops quite fast when p is larger than approximately 50.

a factorization of the sum of Q and a positive diagonal matrix. Forming the (generally dense) kernel matrix Q requires $O(m^2n)$ time for the three standard kernel functions mentioned above and $O(m^2)$ storage, and factorizing it costs $O(m^3)$ operations. When the number of training vectors is large (say, greater than 10000), the QP therefore becomes prohibitively expensive to solve by general-purpose QP interior-point solvers.

Efforts to improve the efficiency of large-scale SVM algorithms have been most successful for the linear kernel $h(u, v) = u^T v$. If the linear kernel is used, $Q = XX^T$ if X is the $m \times n$-matrix with rows x_i^T. Therefore $\mathbf{rank}(Q) \leq n$ if $m \geq n$. This property can be exploited to reduce the complexity of an interior-point method from $O(m^3)$ to $O(nm^2)$ per iteration [FS02, FM03]. Research on fast first-order algorithms, such as projected gradient or cutting-plane algorithms, has also focused largely on the linear kernel [HCL08, FCH08, Joa06, JY09] by taking advantage of the fact that the gradient of the objective $-Qz + d$ can be evaluated in $O(mn)$ operations if Q has rank n.

If a nonlinear kernel is used, the matrix Q is generally dense and possibly full rank[1], and this complicates the implementation of quadratic programming algorithms. When m is large, it therefore makes sense to approximate Q by a simpler matrix which is easy to compute, requires less storage, and makes the QP easier to solve. Examples are low-rank [FS02] or diagonal-plus-low-rank [FM03] approximations. If the rank is much lower than m, the cost per iteration of an interior-point method can be reduced to $O(mn^2)$ by using the Sherman-Morrison-Woodbury formula or the product-form Cholesky factorization algorithm. Low-rank or diagonal-plus-low-rank approximations also simplify the implementation of first-order methods because matrix–vector products are simplified. Finding a

[1]Some kernel functions (such as the RBF kernel) yield a positive definite kernel matrix in theory. In practice, however, it can happen that the kernel matrix is numerically rank deficient.

suitable low-rank approximation, however, is a nontrivial task when m is large, since optimal approximations based on an eigenvalue decomposition are very expensive. Fine and Scheinberg [FS02] discuss a partial Cholesky factorization algorithm with pivoting for computing a low-rank approximation of a kernel matrix. In related work, Smola and Schölkopf [SS00] discuss greedy algorithms for low-rank approximations of the kernel matrix. Low-rank approximation also plays a role in the reduced SVM [LM01, LH06] formulation.

In this chapter we explore an idea similar to that of low-rank kernel matrix approximation. Instead of making a low-rank approximation, we approximate the kernel matrix by a matrix with a sparse inverse. The approximation is obtained by computing the maximum determinant positive definite completion of a partial kernel matrix. The approximated kernel matrix is dense and full rank but has the property that its inverse is sparse. This makes the QP very easy to solve. An added advantage is that only a subset of the entries of the kernel matrix are needed to compute the approximation. Recall that the Cholesky factor of the inverse of the completion can be computed very efficiently if the positions of the specified entries in the partial kernel matrix, which are also the nonzero positions in its inverse, form a *chordal* pattern, for example, a band pattern. Our goal is to evaluate the performance of an SVM training algorithm based on this sparse inverse kernel approximation. We focus on interior-point methods, but the approximation should be useful in first-order methods as well. It can also be combined with chunking, decomposition, and active set methods that are based on solving a sequence of lower-dimensional subproblems [OFG97, Pla99, Joa99, CL01].

4.2.1 Approximate Support Vector Machine Training

The matrix completion theory from §2.3.4 can be applied to compute an optimal (i.e., having minimum relative entropy) approximation of a partial kernel matrix. The approximation is a positive definite matrix with a sparse inverse. In this section, we examine the computational advantages of a sparse inverse kernel approximations when solving the SVM training problem (4.8).

Suppose $\bar{Q}$ is the maximum determinant extension of a partial kernel matrix $Q_{ij} = h(x_i, x_j)$, $(i, j) \in V$, where V is a chordal pattern. (In our experiments we will select V randomly, by choosing a band pattern after applying a random permutation of the training vectors.) We will substitute $\bar{Q}$ for Q in the training problem (4.8) and consider

$$\begin{array}{ll} \text{maximize} & -(1/2)z^T\bar{Q}z + d^Tz \\ \text{subject to} & 0 \preceq \mathbf{diag}(d)z \preceq \gamma\mathbf{1} \\ & \mathbf{1}^Tz = 0. \end{array} \tag{4.9}$$

In this section we first describe in detail how one can take advantage of the sparsity of $\bar{Q}^{-1}$ in an interior-point method for the QP (4.9) (see §4.2.1.1 and §4.2.1.2). We then discuss two possible ways of using the solution of (4.9). First, we can interpret the optimal z as the *exact* dual solution of an SVM training problem for an unknown kernel function $\bar{h}$ with values $\bar{h}(x_i, x_j) = \bar{Q}_{ij}$ on the training set. We will discuss two choices for $\bar{h}$ in §4.2.1.3 and compare the performance of the resulting classifiers in §4.2.2. Second, we can view the optimal z as an *approximation* of the solution of the QP (4.8) associated with the original kernel h and use the values of z to select a subset of training vectors for which we then solve a smaller dense QP exactly. This idea is investigated in §4.2.2.

4.2.1.1 Quadratic Programming Duality

We first review the quadratic programming duality theory for the SVM training problem (4.8). The dual of the problem can be expressed as

$$\begin{array}{ll} \text{minimize} & (1/2)y^T Q^\dagger y + \gamma \mathbf{1}^T v \\ \text{subject to} & y \in \text{range}(Q) \\ & \mathbf{diag}(d)(y + b\mathbf{1}) + v \succeq \mathbf{1} \\ & v \succeq 0, \end{array} \tag{4.10}$$

with variables $y \in \mathbf{R}^m$, $b \in \mathbf{R}$, and $v \in \mathbf{R}^m$. Here $Q^\dagger$ denotes the pseudo-inverse of Q (for now we do not make any assumptions on the invertibility of Q). The vector b is the multiplier associated with the equality constraint in (4.8); the vector v is the multiplier for the inequality $\mathbf{diag}(d)z \preceq \gamma\mathbf{1}$. The multiplier for the constraint $\mathbf{diag}(d)z \succeq 0$ has been eliminated from (4.10) and is equal to $\mathbf{diag}(d)(y + b\mathbf{1}) + v - \mathbf{1}$. If we make a substitution $y = Qu$ we can simplify the dual problem as

$$\begin{array}{ll} \text{minimize} & (1/2)u^T Q u + \gamma \mathbf{1}^T v \\ \text{subject to} & \mathbf{diag}(d)(Qu + b\mathbf{1}) + v \succeq \mathbf{1} \\ & v \succeq 0 \end{array} \tag{4.11}$$

with variables u, b. It can be shown that if z is optimal in (4.8), then $y = Qz$ is optimal in (4.10) and hence $u = z$ is optimal in (4.11).

We can eliminate v and write the dual QP (4.11) as an unconstrained nondifferentiable problem

$$\text{minimize} \quad \frac{1}{2}u^T Q u + \gamma \sum_{i=1}^m \max\{0, 1 - d_i(\sum_{j=1}^m Q_{ij}u_j + b)\}. \tag{4.12}$$

Since $Q_{ij} = h(x_i, x_j)$ the second term is

$$\sum_{i=1}^m \max\{0, 1 - d_i(\sum_{j=1}^m h(x_i, x_j)u_j + b)\},$$

i.e., the familiar hinge-loss penalty for the classifier

$$g(x) = \mathbf{sign}(\sum_{j=1}^{m} h(x, x_j)u_j + b) \tag{4.13}$$

evaluated on the training set defined by x_i, d_i, $i = 1, \ldots, m$. If the linear kernel is used ($h(x_i, x_j) = x_i^T x_j$ and $Q = X^T X$, where X is the matrix with rows x_i^T), then (4.12) can be written as

$$\text{minimize} \quad \frac{1}{2} w^T w + \gamma \sum_{i=1}^{m} \max\left\{0, 1 - d_i(w^T x_i + b)\right\}$$

with $w = X^T u$. The classifier (4.13) reduces to $g(x) = \mathbf{sign}(w^T x + b)$.

4.2.1.2 Interior-point Method

We now discuss the implementation of interior-point algorithms for solving the QP (4.9) and its dual, when $\bar{Q}$ is large and dense, but also invertible with a sparse inverse. For invertible $\bar{Q}$ the dual reduces to

$$\begin{array}{ll} \text{minimize} & (1/2) y^T \bar{Q}^{-1} y + \gamma \mathbf{1}^T v \\ \text{subject to} & \mathbf{diag}(d)(y + b\mathbf{1}) + v \succeq \mathbf{1} \\ & v \succeq 0. \end{array} \tag{4.14}$$

The main step in an iteration of an interior-point method applied to (4.9) and (4.14) is the solution of a linear equation of the form

$$\begin{bmatrix} \bar{Q}^{-1} & 0 & 0 & -\mathbf{diag}(d) & 0 \\ 0 & 0 & 0 & -d^T & 0 \\ 0 & 0 & 0 & -I & -I \\ -\mathbf{diag}(d) & -d & -I & -D_1 & 0 \\ 0 & 0 & -I & 0 & -D_2 \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta b \\ \Delta v \\ \Delta z \\ \Delta w \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{bmatrix}, \tag{4.15}$$

where D_1, D_2 are positive diagonal matrices that change at each iteration. We will refer to (4.15) as the Newton system because it can be interpreted as the linearization of the nonlinear equations that define the primal–dual central path.

Interior-point methods are known to reach a high accuracy after a small number of iterations (in the range 10–30), almost independent of problem size, so to get an idea of the overall complexity, it is fair to focus on the cost of solving one Newton system (4.15). (Some common interior-point algorithms solve multiple Newton systems (usually two or three) per iteration, with different right-hand sides but with the same values for D_1 and D_2. The cost of solving the multiple systems is therefore essentially the same as the cost of solving one Newton system [Wri97a, NW06].) We now describe an efficient algorithm for solving (4.15) by taking advantage of the sparsity of $\bar{Q}^{-1}$.

We start by eliminating Δv, Δz, Δw. Let $\widehat{D} = (D_1 + D_2)^{-1}$ which is diagonal and positive. From the bottom three block rows in (4.15) and the identity

$$\begin{bmatrix} 0 & I & I \\ I & D_1 & 0 \\ I & 0 & D_2 \end{bmatrix}^{-1} = \begin{bmatrix} -D_1 D_2 & D_2 & D_1 \\ D_2 & I & -I \\ D_1 & -I & I \end{bmatrix} \begin{bmatrix} \widehat{D} & 0 & 0 \\ 0 & \widehat{D} & 0 \\ 0 & 0 & \widehat{D} \end{bmatrix}$$

we can express Δv, Δz, Δw, as a function of Δy, Δb:

$$\begin{aligned} \begin{bmatrix} \Delta v \\ \Delta z \\ \Delta w \end{bmatrix} &= - \begin{bmatrix} 0 & I & I \\ I & D_1 & 0 \\ I & 0 & D_2 \end{bmatrix}^{-1} \left(\begin{bmatrix} 0 & 0 \\ \mathbf{diag}(d) & d \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta b \end{bmatrix} + \begin{bmatrix} r_3 \\ r_4 \\ r_5 \end{bmatrix} \right) \\ &= - \begin{bmatrix} D_2 \\ I \\ -I \end{bmatrix} \mathbf{diag}(d) \widehat{D} \begin{bmatrix} I & \mathbf{1} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta b \end{bmatrix} \\ &\quad - \begin{bmatrix} -D_1 D_2 & D_2 & D_1 \\ D_2 & I & -I \\ D_1 & -I & I \end{bmatrix} \begin{bmatrix} \widehat{D} r_3 \\ \widehat{D} r_4 \\ \widehat{D} r_5 \end{bmatrix}. \end{aligned} \tag{4.16}$$

Substituting this in (4.15) gives a smaller equation in Δy, Δb:

$$\begin{bmatrix} \bar{Q}^{-1} + \widehat{D} & \widehat{D}\mathbf{1} \\ \mathbf{1}^T\widehat{D} & \mathbf{1}^T\widehat{D}\mathbf{1} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta b \end{bmatrix} = \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} \tag{4.17}$$

where

$$\begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} - \begin{bmatrix} \mathbf{diag}(d) \\ d^T \end{bmatrix} \widehat{D}(D_2 r_3 + r_4 - r_5).$$

The coefficient matrix in (4.17) is positive definite since $\widehat{D} - (\mathbf{1}^T\widehat{D}\mathbf{1})^{-1}\widehat{D}\mathbf{1}\mathbf{1}^T\widehat{D} \succeq 0$ for positive diagonal $\widehat{D}$. To solve (4.17), we can solve two equations

$$(\bar{Q}^{-1} + \widehat{D})y^{(1)} = \hat{r}_1, \qquad (\bar{Q}^{-1} + \widehat{D})y^{(2)} = -\widehat{D}\mathbf{1}, \tag{4.18}$$

and make a linear combination $\Delta y = y^{(1)} + \Delta b\, y^{(2)}$, with Δb chosen to satisfy the last equation in (4.17), i.e.,

$$\Delta b = \frac{\hat{r}_2 - \mathbf{1}^T\widehat{D}y^{(1)}}{\mathbf{1}^T\widehat{D}(y^{(2)} + \mathbf{1})}. \tag{4.19}$$

In summary, we can solve (4.15) by first solving the two equations (4.18), then computing Δb and $\Delta y = y^{(1)} + \Delta b\, y^{(2)}$, and then Δv, Δz, Δw from (4.16). If $\bar{Q}^{-1}$ is sparse then the matrix $\bar{Q}^{-1} + \widehat{D}$ is sparse, with the same sparsity pattern as $\bar{Q}^{-1}$. In particular, as we saw in Section 2.4, if the sparsity pattern of $\bar{Q}^{-1}$ is chordal, then we can factor $\bar{Q}^{-1} + \widehat{D}$ with a zero-fill Cholesky factorization. For a band pattern with constant bandwidth, for example, the cost of solving (4.15), and hence the cost of the interior-point algorithm itself, is *linear* in m.

4.2.1.3 Completion Kernel Classifier

Every positive definite matrix can be interpreted as a kernel matrix for some positive definite kernel function [SS02, p. 44]. Replacing the kernel matrix Q with a positive definite completion $\bar{Q}$ of a subset of the entries of Q can therefore be thought of as applying a modified positive definite kernel function $\bar{h}$. The value of the modified kernel function $\bar{h}$ is known and equal to $\bar{Q}_{ij}$ at pairs of training points (x_i, x_j), $i, j = 1, \ldots, m$, but it is not uniquely defined for other points. To evaluate the decision function

$$\bar{g}(x) = \mathbf{sign}(\sum_{i=1}^{m} \bar{h}(x, x_i) z_i + b), \tag{4.20}$$

at a test point x, we therefore need to assign a value $\bar{h}(x, x_i)$.

A first choice is simply to use $\bar{h}(x, x_i) = h(x, x_i)$. While our results below indicate that this works well in some cases (e.g., if the bandwidth is chosen sufficiently large), there is no guarantee that $\bar{h}$ is a positive definite kernel, as it can happen that the bordered kernel matrix

$$\bar{Q}' = \begin{bmatrix} \bar{Q} & \bar{q} \\ \bar{q}^T & h(x, x) \end{bmatrix} \tag{4.21}$$

is not positive definite if we take $\bar{q}_i = h(x, x_i)$. We will refer to the classifier (4.20) with $\bar{h}(x, x_i) = h(x, x_i)$ as the *standard kernel classifier*. Note that this classifier is defined in terms of the solution to the approximation problem (4.9), not the solution to the original problem (4.8).

A second choice is to take the chordal pattern V used to define the completion $\bar{Q}$ and extend it to a chordal pattern V' for the bordered kernel matrix $\bar{Q}'$ in (4.21). We define $\bar{Q}'_{ij}$ for $(i, j) \in V'$ as

$$\bar{Q}_{ij} = h(x_i, x_j) \quad \forall (i, j) \in V, \qquad \bar{q}_i = h(x_i, x) \quad \forall (i, m + 1) \in V',$$

and use the maximum determinant completion of these entries to define $\bar{q}_i = \bar{h}(x_i, x)$ at the other values of i. Specifically, suppose V is a band pattern of bandwidth $2w + 1$ and let V' be a band pattern of the same bandwidth. The $w + 1$ nonzero entries ρ and r_i, $i > m - w$, in the last column of the Cholesky factorization of

$$\begin{bmatrix} \bar{Q} & \bar{q} \\ \bar{q}^T & h(x,x) \end{bmatrix}^{-1} = \begin{bmatrix} R & r \\ 0 & \rho \end{bmatrix} \begin{bmatrix} R^T & 0 \\ r^T & \rho \end{bmatrix}$$

can be obtained from R using the algorithm described in Section 2.4. This requires $w+1$ kernel evaluations and $O(w^2)$ operations. The factorization $\bar{Q}' = (R')^{-T}(R')^{-1}$ provides a method for evaluating $\bar{h}(x_i, x)$, i.e.,

$$\bar{h}(x_i, x) = e_i^T \bar{Q}' e_{m+1} = u_i \tag{4.22}$$

where u is obtained by solving $R'(R')^T u = e_{m+1}$. The cost of computing u is $O(wm)$. If $w \ll m$, the cost of evaluating the decision function (4.20) is therefore $O(wm)$. We will refer to this classifier as the *completion kernel classifier*.

4.2.2 Numerical Experiments

To evaluate the performance and accuracy of SVMs obtained with the completion-based kernel matrix approximations, we have conducted a series of experiments based on the MNIST database of handwritten digits [LC98]. The training set consists of 60000 patterns while the test set consists of 10000 patterns. We scale the data by 1/256; no other preprocessing is used. All experiments were conducted on a desktop computer with an Intel Core2 Quad Q6600 CPU (2.4 GHz), 4 GB RAM, and running Ubuntu 9.10 (64 bit). The algorithm was implemented in Python as a custom KKT-solver for the QP solver in CVXOPT 1.1.2 [DV08], and we used CHOMPACK 1.1 [DV09a] for chordal matrix computations. The software package LIBSVM 2.9 [CL01] was used for comparison.

We use the RBF kernel function $h(x_i, x_j) = \exp(-\|x_i - x_j\|^2/(2\sigma))$. The kernel matrix completions $\bar{Q}$ are computed for band patterns V with bandwidth $2w+1$, after applying a random permutation of the training examples.

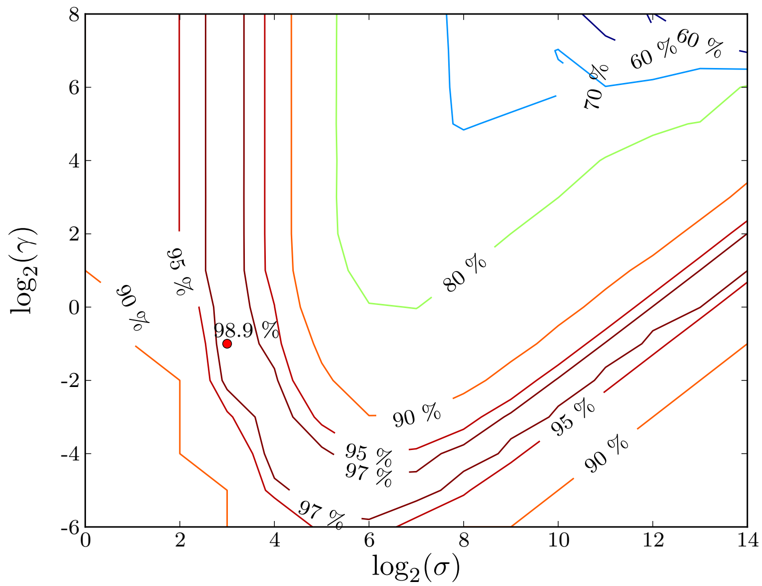
4.2.2.1 Cross-validation Accuracy

In the first experiment we compare the cross-validation accuracy for the two classifiers defined in §4.2.1.3. We use a training set consisting of 10000 randomly chosen examples from the MNIST database (1000 examples of digit 0 and 9000 examples of digits 1–9), and for each pair of parameters ($\gamma = 2^p, \sigma = 2^q$), where p and q are integers, we compute the 10-fold cross-validation accuracy. The half-bandwidth is $w = 100$. The results are shown in Fig. 4.5. We see that the two classifiers have similar cross-validation accuracies when the best combination of parameters is chosen. The completion kernel classifier appears to be less sensitive to parameter changes.

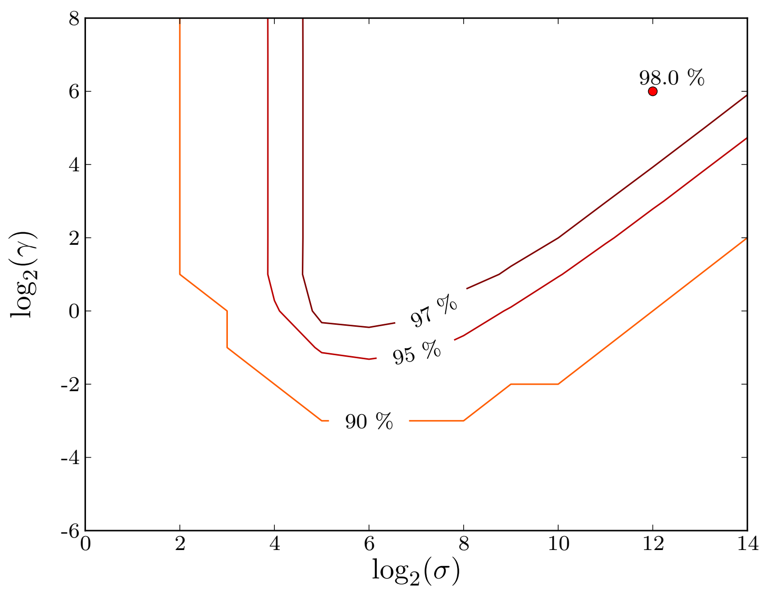
The optimal values of the parameters obviously depend on the bandwidth used in the approximation problem as well as the number of training vectors. In the next experiment we fix the parameters γ and σ and examine the test error rate as a function of bandwidth w.

4.2.2.2 Bandwidth versus Test Error Rate

We consider a single binary classification problem (digit 0 versus digits 1–9) and again we use as training set a subset of 10000 training vectors from the MNIST training set (1000 randomly chosen examples of digit 0 and 9000 randomly chosen examples digits 1–9). In addition to the standard kernel classifier and the completion kernel classifier, we also compute a third classifier by training an SVM with the exact RBF kernel function $h(x_i, x_j)$ on the set of support vectors obtained from the approximation problem. We use $\gamma = 4$ and $\sigma = 32$. Since the elements in the completed kernel matrix depend on the permutation of the training set, we repeated the experiment 10 times with different pseudo-randomly generated permutations. The average of these results is shown in Table 4.1. As the bandwidth



(a) Standard kernel classifier.



(b) Completion kernel classifier.

Figure 4.5: 10-fold cross-validation accuracy using the RBF kernel. The dots mark the best parameter pairs. The completion kernel classifier performs well on a large set of parameters, and it appears to be less sensitive to the choice of parameters than the standard kernel classifier.

w	Approx. problem ($m = 10000$) time	SC error (%)	CC error (%)	SV subset m	time	#SVs	error (%)
10	2	6.22	8.77	7786	567	654	0.30
20	4	7.95	7.39	5985	244	654	0.30
30	5	10.06	6.55	5100	146	649	0.31
40	5	12.10	5.52	4563	104	645	0.31
50	6	13.53	4.65	4152	79	644	0.32
75	10	15.79	3.84	3456	45	642	0.32
100	15	16.57	3.10	2988	28	640	0.32
150	26	16.18	2.28	2407	15	631	0.32
200	41	14.29	1.89	2057	10	630	0.31
250	59	12.12	1.54	1805	7	627	0.32
300	81	9.86	1.34	1623	5	626	0.31
400	138	6.25	1.01	1382	3	622	0.31
500	218	3.85	0.90	1240	2	616	0.31
750	643	1.39	0.67	1024	1	608	0.32
1000	1193	0.76	0.57	925	1	615	0.31

Table 4.1: Average training time (in seconds) and test error rates for the approximation problem and the dense subproblem, parameterized by the half bandwidth w and with parameters $\gamma = 4$ and $\sigma = 32$. SC refers to the standard kernel classifier and CC refers to the completion kernel classifier. For the dense subproblem, m is the number of support vectors obtained from the approximation problem. We remark that solving the full QP ($m = 10000$) took 1194 seconds and produced 659 support vectors and a test error of 0.30 %.

increases and the approximation gets better, the CPU time increases rapidly (as w^2), as expected. In this example, the completion classifier performs better than the approximation classifier except when the smallest bandwidth ($w = 10$) is used. Notice also that the third classifier (obtained by solving a subproblem with the support vectors from the approximation problem) consistently performs well, and the problem size (and hence also the training time) decreases quite fast with increasing bandwidths. The total training time therefore involves a trade-off between the complexity of the approximation problem and the dense subproblem. In this example the fastest total training time (41 seconds) is obtained for $w = 150$.

In column 2 we notice that the solution time for the approximation problem with $w = 1000$ is roughly the same as the time for the full dense QP with $m = 10000$. We believe that this is due to overhead in the current implementation of CHOMPACK (in which it is assumed that $w \ll m$). In an ideal implementation, the cost for the banded case should approach the cost for the dense case only when $w \approx m$.

Extrapolating the values in columns 4 and 5, we also note that the error rate in the completion classifier and the support vectors stop decreasing substantially after a certain value of $w < m$.

4.2.2.3 Multistage Method

In the previous experiment we were able to obtain a low error rate by solving a smaller, dense SVM QP with a reduced training set consisting of the support vectors from the approximation problem. This suggests using the approximated QP as a heuristic for working set selection. In our next experiment, we solve a sequence of approximation problems with increasing bandwidth and decreasing training set size. We consider ten binary classification problems (each digit versus the nine other digits). For each instance, we first solve the approximation QP (4.9) using the full MNIST data set as training data and with half-bandwidth w_1. We refer to this as stage 1. At a subsequent stage i we solve (4.9) with half-bandwidth w_i and use as training data the support vectors from stage $i-1$. If the number of support vectors from stage j is below $\bar{m} = 6000$, we proceed to a final stage and solve the SVM QP with the exact kernel matrix Q. At most five stages are used before the final stage, regardless of the number of support vectors in stage 5. If the number of support vectors at stage 5 exceeds the threshold $\bar{m}$, we keep only the $\bar{m}$ training vectors with the largest values of z_i in the solution of the stage 5 problem. We choose $w_1 = 100$ and set $w_i \approx \sqrt{2}w_{i-1}$ at subsequent stages.

Table 4.2 lists the number of support vectors and the time required to solve the QP in each stage. Table 4.3 shows the total CPU time for all stages and the error rate for the resulting classifier. We can note that, on average, the total CPU time for the multiple stages is less than half the CPU time for LIBSVM. Except for digit 9, the test error rate obtained in the final stage is comparable to that of LIBSVM. Furthermore, the final stage produces slightly fewer support vectors than LIBSVM. Whether the smaller number of support vectors is due to the suboptimality of the approximation problem or is simply related to thresholding is not clear. On average, about nine interior-point iterations were required at each stage, and it is possible that this number can be reduced by using a warm-start technique. Finally, we notice that the number of support vectors for each of the ten classifiers varies by more than a factor of three.

Digit	Stage 1 ($w = 100$)		Stage 2 ($w = 141$)		Stage 3 ($w = 200$)	
	time	#SVs	time	#SVs	time	#SVs
0	94	15558	35	6290	22	3757
1	94	8410	19	2688	-	-
2	100	21417	48	11976	44	8175
3	100	19525	47	10932	41	8097
4	95	19395	44	10553	39	7179
5	100	24613	56	14841	55	10311
6	100	17550	39	7564	27	4662
7	99	16752	40	7354	26	5126
8	95	23122	52	13367	50	9861
9	100	19293	44	11802	44	8931

Digit	Stage 4 ($w = 282$)		Stage 5 ($w = 400$)		Final stage (dense)	
	time	#SVs	time	#SVs	time	#SVs
0	-	-	-	-	50	1584
1	-	-	-	-	21	1161
2	47	6392	66	5102	120	2869
3	47	6636	69	5477	150	3149
4	41	5636	-	-	175	2713
5	61	8059	86	6421	192	3192
6	-	-	-	-	92	2002
7	-	-	-	-	122	2511
8	58	8110	86	6789	192	3633
9	52	7585	80	6481	193	3722

Table 4.2: CPU time (seconds) and number of support vectors at each stage. The parameters $\gamma = 0.667$ and $\sigma = 32$ were used at all stages. The full training set ($m = 60000$) is used at stage 1, and at the following stages, the support vectors from the previous stage are used as training set. We skip to the final stage, where we solve a dense subproblem, when the number of support vectors is 6000 or fewer. If there are more than 6000 support vectors at stage 5, we truncate the set based on the magnitude of the variables z_i. On average 9.2 interior-point iterations were needed to solve a subproblem.

Digit	Multistage approximation			LIBSVM		
	#SVs	time	error (%)	#SVs	time	error (%)
0	1584	203	0.21	1650	560	0.23
1	1161	135	0.24	1254	266	0.20
2	2869	430	0.52	3051	936	0.54
3	3149	456	0.65	3492	1100	0.51
4	2713	395	0.40	2869	804	0.49
5	3192	553	0.38	3412	1066	0.51
6	2002	260	0.32	2015	472	0.33
7	2511	289	0.71	2540	672	0.76
8	3633	536	0.72	4072	1205	0.67
9	3722	515	3.53	4138	1072	0.89

Table 4.3: Number of support vectors, total CPU time (in seconds), and test error rate for the multistage method as well as LIBSVM. We used the parameters $\gamma = 0.667$ and $\sigma = 32$ for all classifiers. The multistage method yields fewer support vectors in all cases. For the digits 0–8, the test error rates obtained with the approximation method are similar to those obtained with LIBSVM. The last classifier (9 versus the rest), however, has an error rate of nearly four times that of LIBSVM. The approximation algorithm required a total of 3777 seconds whereas the total for LIBSVM was 8156 seconds.

m	Stage 1 ($w = 100$)		Stage 2 ($w = 200$)		Stage 3 (dense)	
	#SVs	error (%)	#SVs	error (%)	#SVs	error (%)
2000	613	2.89	294	0.88	244	0.55
4000	1221	8.74	525	0.75	380	0.32
8000	2289	4.66	916	0.62	508	0.31
16000	4686	2.34	1687	0.84	712	0.28
32000	8628	1.97	2950	0.71	1075	0.24
50000	13023	4.76	4188	0.81	1398	0.21

m	LIBSVM	
	#SVs	error (%)
2000	277	0.56
4000	405	0.33
8000	543	0.33
16000	764	0.28
32000	1114	0.25
50000	1435	0.23

Table 4.4: Test error rates and number of support vectors at the three stages and for LIBSVM. Here m is the number of training vectors used at stage 1 and with LIBSVM, and the error rates reported at stage 1 and 2 were obtained using the completion kernel classifier. The parameters $\gamma = 40000/m$ and $\sigma = 32$ were used.

4.2.2.4 Number of Training Vectors versus Test Error Rate

In the last example, we consider the test error rate as a function of the number of training vectors m. As in the previous experiment, we use a multi-stage approach where the last stage is a dense QP, but in this experiment we do not use a threshold before the final dense SVM QP. We train a single classifier with 0 as class 1 and 1–9 as class 2. The training set consists of m_+ randomly chosen examples from class 1 and $m_- = 9m_+$ randomly chosen examples from class 2. The test set is the full MNIST test set. Table 4.4 shows the test error rate at each stage as well as the test error rate obtained with LIBSVM. As can be seen, the number of support vectors at stage 1 grows roughly linearly with the number of training vectors. As a result, the overall CPU time grows faster than

Stage 1		Stage 2		Stage 3		Stage 1+2+3	LIBSVM
m	time	m	time	m	time	time	time
2000	3	613	1	294	0.09	5	2
4000	6	1221	4	525	0.3	11	6
8000	13	2289	8	916	1	23	15
16000	26	4686	18	1687	5	50	41
32000	57	8628	32	2950	25	115	128
50000	85	13023	52	4188	74	212	424

Table 4.5: CPU time (in seconds) for the three stages and for LIBSVM. The parameters $\gamma = 40000/m$ and $\sigma = 32$ were used. The time at stages 1 and 2 grows linearly with m whereas at stage 3, which involves a dense QP, the time grows faster than linear.

linearly (more or less quadratically) with the training set size. This growth rate is comparable with LIBSVM.

4.2.3 Other Applications

Sparse inverse approximations of dense positive definite matrices are useful for a variety of optimization problems. In this section, we mention two interesting further examples from machine learning.

4.2.3.1 Gaussian Process Classification

We first consider Gaussian process (GP) classification [RW06, Section 3.3]. In two-class GP classification, it is assumed that the probability of observing a binary outcome ± 1 at a point $x \in \mathbf{R}^n$ depends on the value of a *latent variable* $f(x)$ via the formulas

$$\mathbf{prob}\,(\text{outcome at } x \text{ is } 1 \mid f(x)) = \kappa(f(x))$$

and

$$\begin{aligned}\mathbf{prob}\,(\text{outcome at } x \text{ is } -1 \mid f(x)) &= 1 - \kappa(f(x)) \\ &= \kappa(-f(x)),\end{aligned}$$

where $\kappa : \mathbf{R} \to \mathbf{R}$ is a symmetric function (i.e., satisfying $\kappa(-u) = 1 - \kappa(u)$) with values in $[0, 1]$. The latent variable $f(x)$ is assumed to be random with a zero-mean Gaussian process with covariance function $h(x, y) = \mathbf{E}\, f(x)f(y)$ as prior distribution. Common choices for κ are the logistic function $\kappa(u) = 1/(1 + \exp(-u))$ and the probit function (the cumulative density of a zero-mean unit-variance Gaussian). We note that these two functions $\kappa(u)$ are log-concave.

Suppose we have a training set of observed outcomes $d_i \in \{-1, 1\}$ at m training points x_i. To simplify the notation we denote by $F = (f(x_1), \ldots, f(x_m))$ the random vector of latent variables at the training points. The first step in deriving the GP classifier is a maximum a posteriori (MAP) estimation of F, given the observed outcomes $d = (d_1, \ldots, d_m)$. The MAP estimate of F is the solution of the optimization problem

$$\text{maximize} \quad L(u) = \sum_{i=1}^{m} \log \kappa(d_i u_i) - \frac{1}{2} u^T Q^{-1} u - \frac{1}{2} \log \det Q \tag{4.23}$$

with variable $u \in \mathbf{R}^m$. The matrix Q has elements $Q_{ij} = h(x_i, x_j)$ and is the covariance of the prior distribution $\mathcal{N}(0, Q)$ of F. The function $L(u)$ is, up to a constant, the logarithm of the posterior density $p_{F|d}(u)$ of the latent variables F, given the observed outcomes d_i, $i = 1, \ldots, m$. The MAP estimate of the latent variables at the training points is the solution of (4.23). Problem (4.23) is a convex optimization problems if the function κ is log-concave. We refer the reader to [RW06, §3.4.2] for the details on how the MAP estimate is applied for classifying test points.

The key step in GP classification is the solution of the unconstrained optimization problem (4.23). If the function κ is log-concave, this problem is convex and can be solved by Newton's method. Each step requires the solution of an equation $\nabla^2 L(u)\Delta u = \nabla L(u)$, or

$$\left(Q^{-1} + D\right)\Delta u = -\nabla L(u) \tag{4.24}$$

where D is diagonal with diagonal elements

$$D_{ii} = \frac{\kappa'(d_i u_i)^2 - \kappa''(d_i u_i)\kappa(d_i u_i)}{\kappa(d_i u_i)^2}.$$

For most common kernel functions the covariance Q and its inverse Q^{-1} are dense, so the Newton equation may be expensive to solve when n is large. The complexity can be improved by making low-rank or low-rank-plus-diagonal approximations, as described in Section 4.2 (see also [RW06, §3.4.3]). Clearly, the Newton equation (4.24) becomes much easier to solve if we can replace Q^{-1} by a sparse approximation $\bar{Q}^{-1}$, obtained via a maximum determinant positive definite completion $\bar{Q}$ of $Q_{ij} = h(x_i, x_j)$, $(i,j) \in V$. This is equivalent to replacing the Gaussian prior $\mathcal{N}(0, Q)$ in (4.23) with the maximum entropy distribution that matches the moments $\mathbf{E}\, f(x_i) f(x_j) = h(x_i, x_j)$, $(i,j) \in V$.

4.2.3.2 Kernel PCA

In the two applications discussed so far, we have used sparse (zero-fill) Cholesky factorizations of the inverse completion kernel to reduce the linear algebra cost of interior-point algorithms and Newton's method. Another benefit of the sparse inverse is that it simplifies matrix–vector products. As an example, we discuss how the matrix completion techniques can be exploited in kernel principal component analysis (PCA).

The main computational effort in kernel PCA lies in computing an eigenvalue decomposition of the centered kernel matrix [SSM98]

$$\begin{aligned} Q_c &= (I - (1/m)\mathbf{1}\mathbf{1}^T)Q(I - (1/m)\mathbf{1}\mathbf{1}^T) \\ &= Q - (1/m)\mathbf{1}\mathbf{1}^T Q - (1/m)Q\mathbf{1}\mathbf{1}^T + (1/m^2)\mathbf{1}\mathbf{1}^T Q\mathbf{1}\mathbf{1}^T \end{aligned} \tag{4.25}$$

where $Q_{ij} = h(x_i, x_j)$. Clearly, computing the entire eigenvalue decomposition is not feasible when the number of observations m is large, so in practice some approximation is often used, for example, sparse greedy methods [SS00]. If matrix–vector products with Q_c are cheap, the dominant eigenvalues and eigenvectors can also be computed iteratively, using the Lanczos method [GV96].

Suppose we replace the dense kernel matrix Q with the maximum determinant positive definite completion $\bar{Q}$ of a partially specified kernel matrix. The matrix–vector product $\bar{Q}_c v$ can then be evaluated efficiently given the sparse factorization $\bar{Q}^{-1} = RR^T$, i.e., using (4.25),

$$\bar{Q}_c v = R^{-T}(R^{-1}v - z\mathbf{1}^T v) - \mathbf{1}z^T R^{-1}v + z^T z\mathbf{1}\mathbf{1}^T v \tag{4.26}$$

where z is the solution of $Rz = \mathbf{1}$. The cost of evaluating (4.26) is *linear* in m for band patterns with fixed bandwidth. Hence the dominant eigenvalues of $\bar{Q}_c$ and the corresponding eigenvectors can then be computed efficiently using the Lanczos method.

4.3 Summary

In this chapter we have investigated the use of maximum determinant positive definite matrix completion as a means to compute a sparse approximation of a symmetric positive definite matrix. The theory, as well as our preliminary experiments, indicate that the accuracy of the approximation is determined largely by the choice of sparsity pattern. While the experiments were limited to band and block-arrow structured approximations, we intend to investigate the use of more general chordal sparsity patterns in the future. Future work may also include a study of the use of drop strategies to further improve the speed of the iterations.

Interior-point methods for support vector machine training provide robust and accurate solutions, but their scope is known to be limited by the high demands of computation time and storage, as a consequence of the density of the kernel matrix. Scaling interior-point methods to training set sizes above, say, 10000 therefore requires approximations of the dense kernel matrix. Examples of such approximations that have been studied in the literature include low-rank, diagonal-plus-low-rank, and sparse approximations. By approximating the positive definite kernel matrix by the maximum determinant positive definite completion of a partially specified kernel matrix with chordal sparsity, we obtain an approximate QP where the inverse of the kernel matrix is sparse. Exploiting the sparsity of the inverse in an interior-point method leads to a dramatic improvement in solution time and memory requirements. As a consequence, very large problems can be solved approximately on a standard desktop PC using an interior-point method. Numerical results with band sparsity indicate that the method can be faster than LIBSVM while the test error rates are comparable. However, more experimentation is needed to evaluate the robustness of the method across different kernel functions and data sets. The positive definite matrix completion techniques should also be of interest in other applications that involve large dense convex optimization problems. As two examples in machine learning, we have mentioned Gaussian process classification and kernel PCA.

CHAPTER 5

Conclusions

We have discussed interior-point methods for linear cone programs involving two types of sparse matrix cones: cones of positive semidefinite matrices with a given chordal sparsity pattern, and their associated dual cones, which consist of the chordal sparse matrices that have a positive semidefinite completion. These cones include as special cases the familiar cones in linear, second-order cone, and semidefinite programming, i.e., the nonnegative orthant (equivalent to a matrix cone with diagonal sparsity pattern), the second-order cone (equivalent to a matrix cone with arrow pattern), and the positive semidefinite cone. They also include a variety of useful nonsymmetric cones, for example, cones of sparse matrices with band or block-arrow patterns, as well as chordal embeddings of general (nonchordal) sparse matrix cones.

Sparse matrix cone programs can be solved as SDPs, i.e., by embedding the cones into dense positive semidefinite cones. Sparsity in the coefficient matrices is exploited, to the extent possible, when solving the Newton equations. The advantage of this approach is that symmetric primal–dual methods can be applied. An alternative approach is to solve the sparse matrix cone programs directly via a nonsymmetric (primal or dual) interior-point method. The advantage of this approach is that we work with cones in a lower-dimensional space, and it is possible to take advantage of efficient algorithms for computing the values, and first and second derivatives of the logarithmic barriers for the chordal sparse matrix cones. However, a theoretical comparison of the complexity is difficult because the techniques for exploiting sparsity in SDP solvers are quite involved and their efficiency depends greatly on the sparsity pattern. In this dissertation, we have developed, implemented, and tested nonsymmetric interior-point methods for conic optimization with chordal sparse matrix cone constraints. Our results show that implementations based on the chordal sparse matrix cone approach are competitive or better than general purpose SDP solvers for many sparse problems. For problems with band or block-arrow structure, the cost per interior-point iteration grows only linearly in the order of the matrix variable, instead of quadratically or worse.

Symmetric primal–dual interior-point methods for LP are known to be numerically more stable than purely primal or dual algorithms [Wri95, Wri97b], and the same benefit is believed to hold for interior-point methods for second-order cone programming and semidefinite programming as well. Despite forcing us to give up primal–dual symmetry, however, the sparse matrix cone formulation allows us to work in a lower-dimensional subspace, and this often makes the augmented system approach (method 2) a feasible alternative to methods that form and solve the Schur complement equations. It is well-known that the augmented system approach is the preferred method from a numerical point-of-view, but in practice, it is rarely used in general-purpose semidefinite programming solvers because of the large size of the Newton system. We believe that using the augmented system approach, at least to some extent, makes up for the loss of primal–dual symmetry.

Another conclusion of the experiments is that the chordal techniques can serve as a complement to, but not a replacement of, the various techniques used in existing SDP solvers. In our implementation of method 1, for example, we found it useful to exploit sparsity in the data matrices using a technique (T2) that does not rely on chordality and is related to techniques common in sparse semidefinite programming. As another example, the good results for DSDP show the importance of exploiting low-rank structure in many applications. It may therefore be advantageous to implement and choose between or combine several techniques in a more sophisticated code.

We can mention a few possible directions for continuing this research. As we have already pointed out, the efficiency and robustness of the solver can be improved in many respects, for example, by a better selection of starting point, the use of a self-dual embedding for initialization, or by using more sophisticated predictor steps. Furthermore, we have seen that the aggregate sparsity pattern V plays an important role in the sparse matrix cone formulation. As a result, a single dense coefficient matrix will destroy the aggregate sparsity and force us to work with dense matrix cones. However, even if only a few of the coefficient matrices are dense, it may still be possible to exploit chordal structure if the dense coefficient matrices are low-rank. The dual slack variable S is then the sum of a sparse term and a low-rank term, and it can be factored efficiently with a product-form Cholesky factorization if the sparse part of S is positive definite. Unfortunately, the sparse part of S may be indefinite, and this makes it more challenging to exploit structure when some of the coefficient matrices are dense.

As another possible continuation of the research on sparse matrix cone programs, we mention first-order interior-point methods in combination with decomposition methods. This combination may also provide a way to handle a few dense low-rank coefficient matrices. Related work on first-order methods for large-scale semidefinite programming includes [LNM07, WGY10].

We have also explored other applications of chordal matrix techniques in nonlinear optimization. We discussed a technique for computing sparse inverse approximations based on maximum determinant positive definite completion theory. This technique can be used to approximate large, dense kernel matrices that arise in applications in machine learning. The kernel matrix approximation depends on the sparsity pattern of a partial positive kernel matrix, and it is cheaper to compute and requires less storage than the full kernel matrix. Furthermore, the completion-based approximation technique provides an alternative to existing approximation methods that are typically based on randomized low-rank approximation. We have implemented and tested the technique in combination with an interior-point method. Our results show that the sparse inverse approximation makes it possible to solve very large SVM training problems using interior-point methods, and the method can perform well in terms of test error rate and speed of training.

A useful extension of the completion-based sparse inverse approximation technique would be a method for automatically selecting the sparsity pattern of the approximate inverse. Such a method would potentially improve the approximation accuracy.

The techniques for chordal sparse matrices should also be useful for improving the efficiency of numerical methods for large-scale convex optimization problems involving Euclidean distance matrix (EDM) cone constraints. A nonnegative symmetric matrix $X \in \mathbf{S}^n$ with zero diagonal is called an EDM if there exist vectors $p_1, \ldots, p_n$, such that $X_{ij} = \|p_i - p_j\|_2^2$. EDMs have applications in multidimensional scaling in statistics [CC01], machine learning [WS06, XSB06], and node localization in sensor networks [WZB07, CY07, KW10]. The cone of EDMs is a convex cone, and it can be represented in terms of the positive semidefinite cone. Hence many problems involving EDM cone constraints can be solved via semidefinite programming. Furthermore, a result by Bakonyi and

Johnson [BJ95] relates EDM completion problems and chordal graphs: if V is chordal, then $X \in \mathbf{S}_V^n$ has an EDM completion if and only if every fully specified principal submatrix of X is an EDM, i.e., if $X_{W_k W_k}$ is an EDM for all the cliques W_k of V. Moreover, if X has an EDM completion, such a completion is easily constructed. However, for general nonchordal V, numerical optimization methods must be used to construct EDM completions [AKW99, Tro00]. Chordal matrix techinques may therefore provide a way to solve large, sparse conic optimization problems with EDM cone constraints efficiently.

APPENDIX A

Chordal Matrix Algorithms

This appendix provides a summary of the chordal matrix algorithms described in Chapter 2. These algorithms are described in detail in [DVR08]. The notation is the same as in Chapter 2. We assume a clique tree of a chordal pattern V is given, with l cliques W_i numbered in reverse topological order. Each clique is partitioned in two sets U_i, V_i, defined in (2.5). Although it is not essential, we can assume that the matrices are permuted so that

$$\begin{aligned} W_1 &= \{1, \ldots, |W_1|\}, \\ W_k &= \left\{ \sum_{j=1}^{k-1} |W_j| + 1, \quad \sum_{j=1}^{k-1} |W_j| + 2, \quad \ldots, \quad \sum_{j=1}^{k} |W_j| \right\} \text{ for } k > 1. \end{aligned} \tag{A.1}$$

As an example, Fig. 2.8 shows the sparsity pattern and the clique tree of the sparsity pattern in Fig. 2.7, after applying a symmetric reordering of rows and columns to satisfy (A.1).

To simplify the notation, we define the Cholesky factorization of $S \in \mathbf{S}^n_{V,++}$ as a factorization

$$S = RDR^T \tag{A.2}$$

where D is block diagonal with dense diagonal blocks $D_{V_kV_k}$, and R is a square matrix with unit diagonal and off-diagonal entries equal to zero, except for the submatrices $R_{U_kV_k}$, $k = 1, \ldots, l$, which are dense. The matrix $R^T + D + R$ then has the same sparsity pattern as S, and if the nodes are numbered as in (A.1), the matrix R is unit upper triangular. The ordering (A.1) is therefore referred to as a *perfect elimination ordering* for the factorization (A.2). By factoring the diagonal blocks of D and absorbing the factors in R, and then applying a symmetric permutation to R, we obtain from (A.2) a standard Cholesky factorization of the form

$$P^T S P = L L^T,$$

with L lower triangular.

A.1 Cholesky Factorization

The following recursion overwrites a positive definite matrix $S \in \mathbf{S}^n_{V,++}$ with its Cholesky factors.

Algorithm A.1. Cholesky factorization ($S = RDR^T$)

Input: $S \in \mathbf{S}^n_{V,++}$

for $k = l$ **down to** 1 **do**

Factor $S_{V_k V_k} = L_k L_k^T$, for example, using a dense Cholesky factorization.
Compute

$$S_{U_k V_k} := S_{U_k V_k} L_k^{-T}, \quad S_{U_k U_k} := S_{U_k U_k} - S_{U_k V_k} S_{U_k V_k}^T, \quad S_{U_k V_k} := S_{U_k V_k} L_k^{-1}.$$

end for

On completion, $S_{U_k V_k} = R_{U_k V_k}$, $S_{V_k V_k} = D_{V_k V_k}$ for $k = 1, \ldots, l$.

A.2 Gradient of Dual Barrier

The following algorithm computes $P_V(S^{-1}) = -\nabla \phi(S)$ from the Cholesky factorization $S = RDR^T$. The recursion follows from a careful examination of the equation $DR^T S^{-1} = R^{-1}$ [DVR08].

Algorithm A.2. Projected inverse.

Input: Cholesky factorization $S = RDR^T$ of $S \in \mathbf{S}^n_{V,++}$.

Initialize $X \in \mathbf{S}^n_V$ as $X_{U_kV_k} := R_{U_kV_k}$ and $X_{V_kV_k} := D_{V_kV_k}$.

for $k = 1$ **to** l **do**

Factor $X_{U_kU_k} = L_kL_k^T$ (for example, using a dense Cholesky factorization).

Compute

$$X_{U_kV_k} := L_k^T X_{U_kV_k}, \quad X_{V_kV_k} := X^{-1}_{V_kV_k} + X^T_{U_kV_k} X_{U_kV_k}, \quad X_{U_kV_k} := -L_k X_{U_kV_k}.$$

end for

On completion, $X = P_V(S^{-1})$.

The factorization in the first step can be obtained efficiently by updating and downdating the factorization of $X_{U_iU_i}$, where i is the index of the parent of clique W_k.

A.3 Hessian of Dual Barrier

Since

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

we can compute $\nabla^2\phi(S)[Y] = P_V(S^{-1}YS^{-1})$ by linearizing the expressions for the gradient, using the chain rule. The resulting algorithm can be written as follows.

Algorithm A.3. Evaluate Hessian of dual barrier.

Input: Cholesky factorization $S = RDR^T$ of $S \in \mathbf{S}^n_{V,++}$, projected gradient $P_V(S^{-1})$, and a matrix $Y \in \mathbf{S}^n_V$.

for $k = l$ **down to** 1 **do**
Compute

$$Y_{W_kW_k} := R^{-1}_{W_kW_k} Y_{W_kW_k} R^{-T}_{W_kW_k}. \tag{A.3}$$

end for
for $k = 1$ **to** l **do**
Compute

$$Y_{V_kV_k} := D^{-1}_{V_kV_k} Y_{V_kV_k} D^{-1}_{V_kV_k}, \qquad Y_{U_kV_k} := (S^{-1})_{U_kU_k} Y_{U_kV_k} D^{-1}_{V_kV_k}.$$

end for
for $k = 1$ **to** l **do**
Compute

$$Y_{W_kW_k} := R^{-T}_{W_kW_k} Y_{W_kW_k} R^{-1}_{W_kW_k}.$$

end for

On completion, Y is overwritten with $P_V(S^{-1}YS^{-1})$.

Here we use multiplications with $R^{-1}_{W_kW_k}$ to simplify the notation. In a practical implementation, these operations can be expressed in terms of multiplications with the nonzero blocks $R_{U_kV_k}$ of R (see [DVR08]). The update (A.3), for example, can be written as

$$\begin{bmatrix} Y_{U_kU_k} & Y_{U_kV_k} \\ Y_{V_kU_k} & Y_{V_kV_k} \end{bmatrix} := \begin{bmatrix} I & -R_{U_kV_k} \\ 0 & I \end{bmatrix} \begin{bmatrix} Y_{U_kU_k} & Y_{U_kV_k} \\ Y_{V_kU_k} & Y_{V_kV_k} \end{bmatrix} \begin{bmatrix} I & 0 \\ -R^T_{U_kV_k} & I \end{bmatrix}.$$

A.4 Factors of Hessian of Dual Barrier

The first and third steps of the Hessian evaluation perform adjoint operations, and the second step is self-adjoint. We can therefore interpret the algorithm as evaluating the Hessian in factored form:

$$\nabla^2\phi(S)[Y] = \mathcal{L}_{\mathrm{adj}}(\mathcal{L}(Y)).$$

The forward mapping $\mathcal{L}$ is evaluated as follows. Note that the factorization of $(S^{-1})_{U_kU_k}$ is part of the computation of $P_V(S^{-1})$ in Section A.2, so we assume here it is given. We also assume that $D_{V_kV_k}$ is stored in factored form.

Algorithm A.4. Evaluate factor of Hessian of dual barrier.

Input: Cholesky factorization $S = RDR^T$ of $S \in \mathbf{S}^n_{V,++}$, factorizations $D_{V_kV_k} = P_kP_k^T$ and $(S^{-1})_{U_kU_k} = L_kL_k^T$, and a matrix $Y \in \mathbf{S}^n_V$.
for $k = l$ **down to** 1 **do**
Compute $Y_{W_kW_k} := R^{-1}_{W_kW_k} Y_{W_kW_k} R^{-T}_{W_kW_k}$.
end for
for $k = 1$ **to** l **do**
Compute $Y_{V_kV_k} = P_k^{-1} Y_{V_kV_k} P_k^{-T}$ and $Y_{U_kV_k} = L_k^T Y_{U_kV_k} P_k^{-T}$.
end for

On completion, Y is overwritten with $\mathcal{L}_S(Y)$.

The adjoint $\mathcal{L}_{\mathrm{adj}}$ can be evaluated by a similar recursion, by reversing the order of the iteration and replacing the matrix operations by their adjoints.

Algorithm A.5. Evaluate adjoint factor of Hessian of dual barrier.

Input: Cholesky factorization $S = RDR^T$ of $S \in \mathbf{S}^n_{V,++}$, factorizations $D_{V_kV_k} = P_kP_k^T$ and $(S^{-1})_{U_kU_k} = L_kL_k^T$, and a matrix $Y \in \mathbf{S}^n_V$.
for $k = 1$ **to** l **do**
Compute $Y_{V_kV_k} = P_k^{-T} Y_{V_kV_k} P_k^{-1}$, and $Y_{U_kV_k} = L_k Y_{U_kV_k} P_k^{-1}$.
end for
for $k = 1$ **to** l **do**
Compute $Y_{W_kW_k} := R^{-T}_{W_kW_k} Y_{W_kW_k} R^{-1}_{W_kW_k}$.
end for

On completion, Y is overwritten with $\mathcal{L}_{\mathrm{adj}}(Y)$.

The calculations in the algorithms above are also easily inverted, so we can evaluate the inverse Hessian $\nabla^2\phi(S)^{-1}$, or the inverse factors $\mathcal{L}^{-1}$ and $(\mathcal{L}^{\mathrm{adj}})^{-1}$, in a very similar way.

A.5 Gradient of Primal Barrier

The following algorithm computes the solution $\hat{S} \in \mathbf{S}^n_{V,++}$ of the nonlinear equation $P_V(\hat{S}^{-1}) = X$ where $X \in \mathbf{S}^n_{V,c++}$. The matrix $\hat{S}$ is the negative gradient of the primal barrier function at X (i.e., $\hat{S} = -\nabla\phi_c(X)$). It also defines the Hessian of the primal barrier function via the identity $\nabla^2\phi_c(X) = \nabla^2\phi(\hat{S})^{-1}$.

Algorithm A.6. Cholesky factorization of $\hat{S}$ where $P_V(\hat{S}^{-1}) = X$.

Input: $X \in \mathbf{S}^n_{V,c++}$
  for $k = 1$ **to** l **do**
    Factor $X_{U_kU_k} = L_kL_k^T$, for example, using a dense Cholesky factorization.
    Compute

$$
\begin{aligned}
R_{U_kV_k} &:= L_k^{-1} X_{U_kV_k}, \\
D_{V_kV_k} &:= \left(X_{V_kV_k} - R^T_{U_kV_k} R_{U_kV_k}\right)^{-1}, \\
R_{U_kV_k} &:= -L_k^{-T} R_{U_kV_k}.
\end{aligned}
$$

  end for

On exit, R and D contain the Cholesky factorization of $\hat{S}$.

As in the algorithm for the dual gradient, the factorization in the first step can be obtained by updating and downdating the factorization of $X_{U_iU_i}$, where W_i is the parent of clique W_k.

A.6 Complexity

To gain some insight in the complexity of the algorithms, we examine in more detail the case of a band or block-arrow pattern. A band pattern with half bandwidth w has $n - w$ cliques

$$W_k = \{k, k+1, \ldots, k+w\}, \quad k = 1, \ldots, n-w.$$

These cliques can be arranged in a clique tree with $U_1 = \emptyset$, $V_1 = \{1, 2, \ldots, w+1\}$, and

$$U_k = \{k, k+1, \ldots, w+k-1\}, \quad V_k = \{w+k\}, \quad k = 2, \ldots, n-w.$$

A block-arrow pattern with w dense leading columns and rows has $n-w$ cliques

$$W_k = \{1, 2, \ldots, w, w+k\}, \quad k = 1, \ldots, n-w,$$

which can be arranged in a clique tree with $U_1 = \emptyset$, $V_1 = \{1, 2, \ldots, w+1\}$, and

$$U_k = \{1, 2, \ldots, w\}, \quad V_k = \{w+k\}, \quad k = 2, \ldots, n-w.$$

For these two patterns, $|U_1| = 0$, $|V_1| = w+1$, and $|U_k| = w$, $|V_k| = 1$ for $k = 2, \ldots, n-w$, so they have maximal overlap between cliques. Note also that for a band pattern U_k has $w-1$ elements in common with U_{k-1} if $k \geq 2$, and that for a block-arrow pattern the sets U_k are constant for $k = 2, \ldots, n-w$.

The dominant terms in the flop count of the Cholesky factorization are $(1/3)w^3 + (n-w)w^2$. This is always less than the cost $(1/3)n^3$ of a dense Cholesky factorization of order n, and linear in n for fixed w.

The algorithm for the dual gradient in §A.2 requires a Cholesky factorization of the dense matrix $X_{U_kU_k}$ at each step. For a block-arrow pattern this matrix is the same for all cliques, so only one factorization is needed. For a band pattern, the matrix has a submatrix of order $w-1$ in common with $X_{U_{k-1}U_{k-1}}$, and its factorization can be computed in $O(w^2)$ operations by updating the factorization of $X_{U_{k-1}U_{k-1}}$. The complexity of the rest of the algorithm is dominated by matrix multiplications with a complexity of w^2. The total cost is therefore $O((n-w)w^2)$. Similar observations apply to the algorithm for the primal gradient.

The dominant terms in each step of the evaluation of the dual Hessian (§A.3) or its factors (§A.4) are of order w^2 (for the updates of $Y_{U_kU_k}$). The resulting total cost is $O((n-w)w^2)$.

References

[ADD96] P. Amestoy, T. Davis, and I. Duff. "An approximate minimum degree ordering." *SIAM Journal on Matrix Analysis and Applications*, **17**(4):886–905, 1996.

[AG03] F. Alizadeh and D. Goldfarb. "Second-order cone programming." *Mathematical Programming Series B*, **95**:3–51, 2003.

[AHM88] 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.

[AHO98] 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.

[AKW99] A. Y. Alfakih, A. Khandani, and H. Wolkowicz. "Solving Euclidean distance matrix completion problems via semidefinite programming." *Computational Optimization and Applications*, **12**(1-3):13–30, 1999.

[ART03] E. D. Andersen, C. Roos, and T. Terlaky. "On implementing a primal-dual interior-point method for conic quadratic optimization." *Mathematical Programming*, **95**(2):249–277, 2003.

[AY98] E. D. Andersen and Y. Ye. "A computational study of the homogeneous algorithm for large-scale convex optimization." *Computational Optimization and Applications*, **10**:243–269, 1998.

[BBH04] 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.

[Ben73] M. W. Benson. "Iterative Solutions of Large Scale Linear Systems." Master's thesis, Lakehead University, Thunder Bay, Ontario, 1973.

[BF82] M. W. Benson and P. O. Frederickson. "Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems." *Utilitas Math*, **22**(127), 1982.

[BJ95] M. Bakonyi and C. R. Johnson. "The Euclidian distance matrix completion problem." *SIAM Journal on Matrix Analysis and Applications*, **16**(2):646–654, 1995.

[BJ05] F. R. Bach and M. I. Jordan. “Predictive low-rank decomposition for kernel methods.” In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 33–40. ACM Press, 2005.

[BJL89] 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.

[BMT96] M. Benzi, C. D. Meyer, and M. Tuma. “A Sparse Approximate Inverse Preconditioner for the Conjugate Gradient Method.” *SIAM Journal on Scientific Computing*, **17**(5):1135–1149, 1996.

[BN98] A. Ben-Tal and A. Nemirovski. “Robust convex optimization.” *Mathematics of Operations Research*, **23**:769–805, 1998.

[BN01] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization. Analysis, Algorithms, and Engineering Applications.* SIAM, 2001.

[Bor99a] B. Borchers. “CSDP, a C library for semidefinite programming.” *Optimization Methods and Software*, **11**(1):613–623, 1999.

[Bor99b] B. Borchers. “SDPLIB 1.2, A Library of Semidefinite Programming Test Problems.” *Optimization Methods and Software*, **11**(1):683–690, 1999.

[BP93] 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.

[BT99] M. Benzi and M. Tuma. “A comparative study of sparse approximate inverse preconditioners.” *Applied Numerical Mathematics*, **30**(2–3):305–340, 1999.

[Bur03] S. Burer. “Semidefinite programming in the space of partial positive semidefinite matrices.” *SIAM Journal on Optimization*, **14**(1):139–172, 2003.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, 2004.

[BY05] S. J. Benson and Y. Ye. “DSDP5: Software For Semidefinite Programming.” Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. Submitted to ACM Transactions on Mathematical Software.

[CC01] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Chapman & Hall/CRC, second edition, 2001.

[CDH08] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate." *ACM Transactions on Mathematical Software*, **35**(3):1–14, 2008.

[Che05] K. Chen. *Matrix Preconditioning Techniques and Applications*. Cambridge University Press, 2005.

[CL01] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CS94] E. Chow and Y. Saad. "Approximate Inverse Preconditioners for General Sparse Matrices." Technical Report UMSI 94/101, University of Minnesota Supercomputer Institute, Minneapolis, MN 55415, 1994.

[CW02] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Theory*. Society for Industrial and Applied Mathematics, 2002.

[CY07] A. M.-C. Cho and Y. Ye. "Theory of semidefinite programming for sensor network localization." *Mathematical Programming, Series B*, **109**:367–384, 2007.

[DGO10] 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*, pp. 111–141. Birkhäuser Basel, 2010.

[Die10] R. Diestel. *Graph Gheory*. Springer, fourth edition, 2010.

[DM05] P. Drineas and M. W. Mahoney. "On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning." *Journal of Machine Learning Research*, **6**:2153–2175, 2005.

[DSW88] P. M. Dearing, D. R. Shier, and D. D. Warner. "Maximal chordal subgraphs." *Discrete Applied Mathematics*, **20**(3):181–190, 1988.

[DV08] J. Dahl and L. Vandenberghe. *CVXOPT: A Python Package for Convex Optimization*. `abel.ee.ucla.edu/cvxopt`, 2008.

[DV09a] J. Dahl and L. Vandenberghe. *CHOMPACK: Chordal Matrix Package*. `abel.ee.ucla.edu/chompack`, 2009.

[DV09b] J. Dahl and L. Vandenberghe. *CVXOPT: A Python Package for Convex Optimization*. `abel.ee.ucla.edu/cvxopt`, 2009.

[DVR08] 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.

[EL97] L. El Ghaoui and H. Lebret. "Robust solutions to least-squares problems with uncertain data." *SIAM Journal of Matrix Analysis and Applications*, **18**(4):1035–1064, 1997.

[FCH08] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. "LIBLINEAR: A Library for Large Linear Classification." *Journal of Machine Learning Research*, **9**:1871–1874, 2008.

[FFN06] K. Fujisawa, M. Fukuda, and K. Nakata. "Preprocessing sparse semidefinite programs via matrix completion." *Optimization Methods and Software*, **21**:17–39, 2006.

[FG65] D. R. Fulkerson and O. Gross. "Incidence matrices and interval graphs." *Pacific Journal of Mathematics*, **15**(3):835–855, 1965.

[FKM00] 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.

[FKN97] K. Fujisawa, M. Kojima, and K. Nakata. "Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming." *Mathematical Programming*, **79**(1-3):235–253, October 1997.

[FM68] A. Fiacco and G. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. Wiley, 1968. Reprinted 1990 in the SIAM Classics in Applied Mathematics series.

[FM93] R. Fourer and S. Mehrotra. "Solving symmetric indefinite systems in an interior-point approach for linear programming." *Mathematical Programming*, **62**:15–39, 1993.

[FM03] M. C. Ferris and T. S. Munson. "Interior-Point Methods for Massive Support Vector Machines." *SIAM Journal on Optimization*, **13**(3):783–804, 2003.

[FS02] S. Fine and K. Scheinberg. "Efficient SVM training using low-rank kernel representations." *Journal of Machine Learning Research*, **2**:243–264, 2002.

[Gav74] F. Gavril. "The intersection graphs of subtrees in trees are exactly the chordal graphs." *Journal of Combinatorial Theory, Series B*, **16**(1):47–56, 1974.

[GB12] M. Grant and S. Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.0 (beta)*. `cvxr.com`, 2012.

[Geo73] A. George. "Nested Dissection of a Regular Finite Element Mesh." *SIAM Journal on Numerical Analysis*, **10**(2):345–363, 1973.

[GH97] M. J. Grote and T. Huckle. "Parallel Preconditioning with Sparse Approximate Inverses." *SIAM Journal on Scientific Computing*, **18**(3):838–853, 1997.

[GI03] D. Goldfarb and G. Iyengar. "Robust convex quadratically constrained programs." *Mathematical Programming Series B*, **97**:495–515, 2003.

[GJS84] 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.

[Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, second edition, 2004.

[Gon09] J. Gondzio. "Matrix-Free Interior Point Method." Technical Report ERGO-2009-012, School of Mathematics and Maxwell Institute for Mathematical Sciences, The University of Edinburgh, 2009.

[GS04] D. Goldfarb and K. Scheinberg. "A product-form Colesky factorization method for handling dense columns in interior point methods for linear programming." *Mathematical Programming, Series A*, **99**:1–34, 2004.

[GS05] D. Goldfarb and K. Scheinberg. "Product-form Cholesky factorization in interior point methods for second-order cone programming." *Mathematical Programming Series A*, **103**:153–179, 2005.

[Gul96] O. Güler. "Barrier functions in interior point methods." *Mathematics of Operations Research*, **21**:860–865, 1996.

[GV96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[HCL08] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. Keerthi, and S. Sundararajan. "A dual coordinate descent method for large-scale linear SVM." In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pp. 408–415, New York, NY, USA, 2008. ACM.

[Heg06] P. Heggernes. "Minimal triangulation of graphs: a survey." *Discrete Mathematics*, **306**:297–317, 2006.

[HRV96] 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.

[Joa99] T. Joachims. "Making large-Scale SVM Learning Practical." In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pp. 169–184. MIT Press, Cambridge, MA, 1999.

[Joa06] T. Joachims. "Training Linear SVMs in Linear Time." In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pp. 217–226, 2006.

[JPA00] D. Johnson, G. Pataki, and F. Alizadeh. "Seventh DIMACS implementation challenge: Semidefinite and related problems.", 2000. `http://dimacs.rutgers.edu/Challenges/Seventh`.

[JY09] T. Joachims and C.-N. J. Yu. "Sparse kernel SVMs via cutting-plane training." *Machine learning*, **76**(2):179–193, 2009.

[Kak10] N. Kakimura. "A direct proof for the matrix decomposition of chordal-structured positive semidefinite matrices." *Linear Algebra and Its Applications*, **433**:819–823, 2010.

[Kar84] N. Karmarkar. "A New Polynomial-Time Algorithm for Linear Programming." *Combinatorica*, **4**(4):373–395, 1984.

[KBM96] M. V. Kothare, V. Balakrishnan, and M. Morari. "Robust constrained model predictive control using linear matrix inequalities." *Automatica*, **32**(10):1361–1379, 1996.

[KKK08] 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.

[KKM10] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. "Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion." *Mathematical Programming*, pp. 1–36, 2010.

[KMT09a] S. Kumar, M. Mohri, and A. Talwalkar. "Ensemble Nyström Method." In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pp. 1060–1068. Nips Foundation, 2009.

[KMT09b] S. Kumar, M. Mohri, and A. Talwalkar. "Sampling Techniques for the Nyström Method." In D. van Dyk and M. Welling, editors, *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*. JMLR, 2009.

[KRT97] E. de Klerk, C. Roos, and T. Terlaky. "Initialization in semidefinite programming via a self-dual skew-symmetric embedding." *Operations Research Letters*, **20**(5):213–221, 1997.

[KSH97] 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.

[KW10] N. Krislock and H. Wolkowicz. "Euclidean Distance Matrices and Applications." Technical Report CORR 2010-06, Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada, 2010.

[KY93] L. Yu. Kolotilina and A. Yu. Yeremin. "Factorized sparse approximate inverse preconditionings I: theory." *SIAM Journal on Matrix Analysis and Application*, **14**(1):45–58, 1993.

[Lau96] S. L. Lauritzen. *Graphical Models*. Oxford University Press, Oxford, 1996.

[LC98] Y. LeCun and C. Cortes. "The MNIST Database of Handwritten Digits." Available at `http://yann.lecun.com/exdb/mnist/`, 1998.

[LH06] Y. J. Lee and S. Y. Huang. "Reduced support vector machines: A statistical theory." *IEEE Transactions on Neural Networks*, **18**(1):1–13, 2006.

[LM01] Y. J. Lee and O. L. Mangasarian. "RSVM: Reduced support vector machines." In *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

[LNM07] Z. Lu, A. Nemirovski, and R. Monteiro. "Large-scale semidefinite programming via a saddle point Mirror-Prox algorithm." *Mathematical Programming*, **109**:211–237, 2007.

[Lof04] J. Löfberg. *YALMIP : A Toolbox for Modeling and Optimization in MATLAB*, 2004.

[LS83] R. Laskar and D. Shier. "On powers and centers of chordal graphs." *Discrete Applied Mathematics*, **6**(2):139–147, 1983.

[LSZ00] Z.-Q. Luo, J. F. Sturm, and S. Zhang. "Conic convex programming and self-dual embedding." *Optimization Methods and Software*, **14**:169–218, 2000.

[Mar57] H. M. Markowitz. "The Elimination Form of the Inverse and Its Application to Linear Programming." *Management Science*, **3**(3):255–269, 1957.

[MM65] J. Moon and L. Moser. "On cliques in graphs." *Israel Journal of Mathematics*, **3**:23–28, 1965.

[Mon95] R. D. C. Monteiro. "Primal-dual path following algorithms for semidefinite programming." *SIAM J. on Optimization*, **7**:663–678, 1995.

[Mon98] R. D. C. Monteiro. "Polynomial convergence of primal-dual algorithms for semidefinite programming based on Monteiro and Zhang family of directions." *SIAM J. on Optimization*, **8**(3):797–812, 1998.

[Nes96] Yu. Nesterov. "Long-step strategies in interior point potential reduction methods." *Mathematical Programming*, **76**:47–94, 1996.

[Nes06a] Yu. Nesterov. "Nonsymmetric potential-reduction methods for general cones." Technical Report 2006/34, CORE Discussion Paper, Université catholique de Louvain, 2006.

[Nes06b] Yu. Nesterov. "Towards nonsymmetric conic optimization." Technical Report 2006/28, CORE Discussion Paper, Université catholique de Louvain, 2006.

[NFF03] 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.

[NN94] Yu. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Methods in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[NT97] 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.

[NT98] Yu. E. Nesterov and M. J. Todd. "Primal-dual interior-point methods for self-scaled cones." *SIAM Journal on Optimization*, **8**(2):324–364, May 1998.

[NW06] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[OFG97] E. Osuna, R. Freund, and F. Girosi. "An improved training algorithm for support vector machines." In *Proceedings of the 7th IEEE Workshop on Neural Networks for Signal Processing*, pp. 276–285, 1997.

[Par61] S. Parter. "The use of linear graphs in Gauss elimination." *SIAM Review*, **3**(2):119–130, 1961.

[Pla99] J. C. Platt. "Fast training of support vector machines using sequential minimal optimization." In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 185–208. MIT Press, Cambridge, MA, 1999.

[PS95] F. A. Potra and R. Sheng. "Homogeneous interior-point algorithms for semidefinite programming." Reports on Computational Mathematics 82/1995, Department of Mathematics, The University of Iowa, 1995.

[Ren01] J. Renegar. *A Mathematical View of Interior-Point Methods in Convex Optimization*. SIAM, 2001.

[Ros70] D. J. Rose. "Triangulated graphs and the elimination process." *Journal of Mathematical Analysis and Applications*, **32**:597–609, 1970.

[Ros72] D. J. Rose. "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations." In R. C. Read, editor, *Graph Theory and Computing*, pp. 183–217. Academic Press, 1972.

[Ros74] D. J. Rose. "On simple characterizations of k-trees." *Discrete Mathematics*, **7**:317–322, 1974.

[RTL76] 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.

[RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[SS00] A. J. Smola and B. Schölkopf. "Sparse greedy matrix approximation for machine learning." In *Proceedings of the 17th International Conference on Machine Learning*, pp. 911–918. Morgan Kaufmann, 2000.

[SS02] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[SSM98] B. Schölkopf, A. Smola, and K.-R. Müller. "Nonlinear component analysis as a kernel eigenvalue problem." *Neural Computation*, **10**(5):1299–1319, 1998.

[Ste80] G. W. Stewart. "The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators." *SIAM Journal on Numerical Analysis*, **17**(3):403–409, 1980.

[Stu99] J. F. Sturm. "Using SEDUMI 1.02, a Matlab Toolbox for Optimization Over Symmetric Cones." *Optimization Methods and Software*, **11-12**:625–653, 1999.

[Stu02] J. F. Sturm. "Implementation of interior point methods for mixed semidefinite and second order cone optimization problems." *Optimization Methods and Software*, **17**(6):1105–1154, 2002.

[Stu03] J. F. Sturm. "Avoiding numerical cancellation in the interior point method for solving semidefinite programs." *Mathematical Programming Series B*, **95**:219–247, 2003.

[SV04] G. Srijuntongsiri and S. A. Vavasis. "A Fully Sparse Implementation of a Primal-Dual Interior-Point Potential Reduction Method for Semidefinite Programming.", 2004.

[Tro00] M. W. Trosset. "Distance matrix completion by numerical optimization." *Computational Optimization and Applications*, **17**(1):11–22, 2000.

[TTT98] M. J. Todd, K. C. Toh, and R. H. Tütüncü. "On the Nesterov-Todd direction in semidefinite programming." *SIAM J. on Optimization*, **8**(3):769–796, 1998.

[TTT03] 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.

[TW67] W. F. Tinney and J. W. Walker. "Direct solutions of sparse network equations by optimally ordered triangular factorization." *Proceedings of the IEEE*, **55**(11):1801–1809, 1967.

[TY84] 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.

[VB96] L. Vandenberghe and S. Boyd. "Semidefinite programming." *SIAM Review*, **38**(1):49–95, 1996.

[VBM07] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices: Linear Systems*. Johns Hopkins University Press, 2007.

[VBM09] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices: Eigenvalue and Singular Value Methods*. Johns Hopkins University Press, 2009.

[Wer80] N. Wermuth. "Linear recursive equations, covariance selection, and path analysis." *Journal of the American Statistical Association*, **75**(372):963–972, 1980.

[WGY10] Z. Wen, D. Goldfarb, and W. Yin. "Alternating direction augmented Lagrangian methods for semidefinite programming." *Mathematical Programming Computation*, **2**(3-4):203–230, 2010.

[WKK06] 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.

[Wri95] S. J. Wright. “Stability of linear equations solvers in interior-point methods.” *SIAM Journal on Matrix Analysis and Applications*, **16**(4):1287–1307, 1995.

[Wri97a] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, 1997.

[Wri97b] S. J. Wright. “Stability of augmented system factorizations in interior-point methods.” *SIAM Journal on Matrix Analysis and Applications*, **18**(1):191–222, 1997.

[WS01] C. K. I. Williams and M. Seeger. “Using the Nyström method to speed up kernel machines.” *Advances in neural information processing systems*, pp. 682–688, 2001.

[WS06] K. Q. Weinberger and L. K. Saul. “Unsupervised learning of image manifolds by semidefinite programming.” *International Journal of Computer Vision*, **70**:77–90, 2006.

[WZB07] Z. Wang, S. Zheng, S. Boyd, and Y. Ye. “Further relaxations of the SDP approach to sensor network localization.” Technical report, Department of Management Science and Engineering, Stanford University, 2007.

[XSB06] 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.

[Yam08] N. Yamashita. “Sparse quasi-Newton updates with positive definite matrix completion.” *Mathematical Programming, Series A*, **115**(1):1–30, 2008.

[Yan81] M. Yannakakis. “Computing the minimum fill-in is NP-complete.” *SIAM Journal on Algebraic and Discrete Methods*, **2**(1):77–79, 1981.

[YFK03] 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.

[YTM94] Y. Ye, M. J. Todd, and S. Mizuno. "An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm." *Mathematics of Operations Research*, **19**(1):53–67, 1994.