Noname manuscript No.
(will be inserted by the editor)

Sparse Semidefinite Programs with Guaranteed Near-Linear Time Complexity via Dualized Clique Tree Conversion

Richard Y. Zhang · Javad Lavaei

Abstract Some of the strongest polynomial-time relaxations to NP-hard combinatorial optimization problems are semidefinite programs (SDPs), but their solution complexity of up to $O(n^{6.5}L)$ time and $O(n^4)$ memory for L accurate digits limits their use in all but the smallest problems. Given that combinatorial SDP relaxations are often sparse, a technique known as clique tree conversion can sometimes reduce complexity substantially. In this paper, we describe a dualized verison of clique tree conversion, and prove that the technique allows any general-purpose interior-point method to solve a large class of sparse SDPs with a guaranteed complexity of $O(n^{1.5}L)$ time and $O(n)$ memory. To illustrate the use of this technique, we solve the MAX k-CUT relaxation, the Lovasz Theta problem, and the AC optimal power flow relaxation, on power system models with up to $n = 13659$ nodes.

1 Introduction

Given $n \times n$ real symmetric matrices $C, A_1, \ldots, A_m$ and m-dimensional real vector b, we consider the semidefinite program

$$\begin{aligned} \text{minimize} \quad & C \bullet X & \\ \text{subject to} \quad & A_i \bullet X = b_i \qquad \forall i \in \{1, \ldots, m\} \\ & X \succeq 0 \end{aligned} \tag{SDP}$$

This work was supported by the ONR YIP Award, DARPA YFA Award, AFOSR YIP Award, NSF CAREER Award, and ONR N000141712933.

R.Y. Zhang and J. Lavaei
Department of Industrial Engineering and Operations Research
University of California, Berkeley
Berkeley, CA 94720
United States of America
E-mail: {ryz,lavaei}@berkeley.edu

The notation $A_i \bullet X \equiv \operatorname{tr} A_i X$ refers to the usual matrix inner product, and the constraint $X \succeq 0$ restricts the matrix X to be symmetric positive semidefinite.

Semidefinite programs arise as some of the best convex relaxations to non-convex problems like graph optimization [1,2], integer programming [3,4,5,6], and polynomial optimization [7,8], but their high complexity severely limits their use in practice. Typically, (SDP) is solved using an interior-point method [9,10], to L accurate digits in $O(n^{6.5}L)$ time and $O(n^4)$ space. While these complexity figures are formally polynomial, their high exponents nevertheless limit n to the order of a few hundreds.

1.1 Sparsity graph with bounded treewidth

Sparsity-exploiting techniques [11,12,13,14] can accommodate much larger instances of (SDP), under the assumption that the data matrices C, A_1,..., A_m have a *sparsity graph* with *bounded treewidth.*

Definition 1 The *sparsity graph* $G = (V, E)$ of an $n \times n$ matrix M (resp. a collection of matrices $M_1, \ldots, M_m$) is an undirected graph on n vertices $V = \{1, \ldots, n\}$ with an edge $(i, j) \in E$ if and only if $M[i, j] \neq 0$ (resp. there exists a choice of $M \in \{M_1, \ldots, M_m\}$ such that $M[i, j] \neq 0$).

Definition 2 A *tree decomposition* $\mathcal{T}$ of a graph $G = (V, E)$ is a pair $(\mathcal{J}, T)$, where $\mathcal{J} = \{J_1, \ldots, J_\ell\}$ contains subsets of V, and T is a tree on ℓ vertices, such that:

1. (Vertex cover) For every $v \in V$, there exists $J_k \in \mathcal{J}$ such that $v \in J_k$;
2. (Edge cover) For every $(u, v) \in E$, there exists $J_k \in J$ such that $u \in J_k$ and $v \in J_k$; and
3. (Running intersection) If $v \in J_i$ and $v \in J_j$, then we also have $v \in J_k$ for every J_k that lies on the path from J_i to J_j in the tree T.

The *width* of the tree decomposition $\mathcal{T} = (\mathcal{J}, T)$, denoted $\operatorname{wid}(\mathcal{T})$, is one less than the maximum number of elements in any subset $J_k \in \mathcal{J}$. The *treewidth* of G, denoted $\operatorname{tw}(G)$, is the minimum width amongst all tree decompositions $\mathcal{T}$.

The bounded treewidth assumption is widely satisfied by real-life sparsity graphs. As we review in Section 3, the sparsity graph G of $C, A_1, \ldots, A_m$ has bounded treewidth if and only if every matrix of the form $S = C - \sum_i y_i A_i \succeq 0$ factors into a Cholesky factor R satisfying $RR^T = S$ in *linear time* (possibly after reordering the rows and columns of S). Sparsity graphs that factor in linear time are ubiquitous in real-life applications, including those arising from physical networks like VLSI, communication systems, transportation networks, and electric power systems [15,16,17], as well as abstract graphical structures in statistical and machine learning [18]. All of these sparsity graphs have bounded treewidth by virtue of the claim described above.

If a given graph G is known *a priori* to have bounded treewidth, then a minimum-width tree decomposition $\mathcal{T}$ can be explicitly computed using the

linear-time algorithm of Bodlaender [19]. In practice, it is far more efficient to compute a choice of $\mathcal{T}$ with a small but suboptimal width using a fill-reducing heuristic from numerical linear algebra (see Section 3.3 and the references therein). Our numerical results in Section 9 use MATLAB's in-built approximate minimum degree heuristic to find choices of $\mathcal{T}$ with $\mathrm{wid}(\mathcal{T}) \leq 34$ in less than 0.01 seconds on a standard desktop, for real-world graphs containing as many as $n = 13659$ vertices.

1.2 Clique tree conversion

One popular way to exploit the bounded treewidth property is via the *clique tree conversion* technique of Fukuda and Nakata et al. [12,13]. Given a tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ with small width for the sparsity graph G of $C, A_1, \ldots, A_m$, the technique reformulates (SDP) into an *equivalent* problem of reduced complexity

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{\ell} C_j \bullet X_j && && \text{(CTC)} \\ \text{subject to} \quad & \sum_{j=1}^{\ell} A_{i,j} \bullet X_j = b_i && \forall i \in \{1,\ldots,m\} \\ & N_{J_i J_j}(X_j) = N_{J_j J_i}(X_i) && \forall (i,j) \in E(T) \\ & X_j \succeq 0 && \forall j \in \{1,\ldots,\ell\}, \end{aligned}$$

over semidefinite variables $X_1, X_2, \ldots, X_\ell$ of appropriate dimensions. Here, $E(T)$ refers to the edges of the tree graph T, each new $A_{i,j}$ (resp. C_j) is constructed from a submatrix of the i-th data matrix A_i (resp. C), and the linear operator $N_{J_i J_j}(\cdot)$ outputs the overlapping elements of two principal submatrices, given the latter as the argument[1]

$$N_{J_i J_j}(Z[J_j, J_j]) = Z[J_i \cap J_j,\ J_i \cap J_j] = N_{J_j J_i}(Z[J_i, J_i]).$$

Essentially, clique tree conversion splits the large $n \times n$ semidefinite variable $X \succeq 0$ in (SDP) into $\ell \leq n$ smaller semidefinite variables $X_1, X_2, \ldots, X_\ell$ in (CTC), each of size at most $\omega \equiv 1 + \mathrm{wid}(\mathcal{T})$. Given a solution to (CTC), we rapidly recover a corresponding solution to (SDP) by applying a closed-form formula. (The details of clique tree conversion are reviewed in Section 4, including its derivation and implementation considerations.)

The most expensive part of clique tree conversion is the solution of (CTC). To give a rigorous complexity bound, we state some nondegeneracy assumptions, which are standard for interior-point methods.

Assumption 1 (Nondegeneracy) *We assume in (SDP):*

[1] We use the notation $Z[I, J]$ to index the (I, J)-th submatrix of Z.

1. *(Linear independence) The matrix* $\mathbf{A} = [\operatorname{vec} A_1, \ldots, \operatorname{vec} A_m]$ *has full column-rank, meaning that* $\mathbf{A}^T\mathbf{A}$ *is invertible.*
2. *(Slater's condition) There exist* $X_1, \ldots, X_\ell \succ 0$*,* y*, and* $S_1, \ldots, S_\ell \succ 0$*, such that* $\sum_j A_{i,j} \bullet X_j = b_i$ *and* $\sum_i y_i A_{i,j} + S_j = C_j$ *for all* $i \in \{1, \ldots, m\}$ *and* $j \in \{1, \ldots, m\}$*.*

Note that Slater's condition is usually satisfied in solvers like SeDuMi [20] and MOSEK [21] using the homogenous self-dual embedding technique [22].

Proposition 1 *Let* $\omega \equiv 1 + \operatorname{wid}(\mathcal{T})$*. Then, there exists an interior-point method that solves (CTC) to* L *accurate digits in*

$$O(\omega^{6.5} \cdot n^{3.5} \cdot L) \text{ time and } O(\omega^4 \cdot n^2) \text{ space.} \tag{1}$$

Proof It is a seminal result by Nesterov and Nemirovski [9] that an interior-point method solves every order-θ linear conic program to L accurate digits in $k = O(\sqrt{\theta}L)$ iterations assuming Slater's condition. Problem (CTC) optimizes over a convex cone of order $\theta \leq \omega n$. We show in Section 6 that each interior-point iteration takes $O(\omega^6 n^3)$ time and $O(\omega^4 n^2)$ memory. □

According to Proposition 1, the reduced-complexity problem (CTC) requires up to $O(n^{3.5})$ time and $O(n^2)$ memory to solve. These worst-case figures are sharp. For example, they are attained by dense instances of (CTC) whose constraint matrices $A_{i,j}$ are nonzero for all (i, j).

1.3 Main results

Empirical results [12,13,23,24,16] find that many instances of (CTC) can be actually solved in near-linear time using an interior-point method. Unfortunately, other problem instances attain the worst-case cubic complexity quoted in Proposition 1 above. The key issue is the large number of *overlap constraints*

$$N_{J_i J_j}(X_j) = N_{J_j J_i}(X_i) \qquad \forall (i, j) \in E(T) \tag{2}$$

which are imposed in addition to the constraints already present in the original problem [25, Section 14.2]. These overlap constraints can significantly increase the size and density of the linear system solved at each interior-point iteration, known as the *normal equations* (or *Schur complement equations*). Specifically, the overlap constraints may contribute up to $O(\omega^4 n^2)$ nonzero elements to the normal matrix, thereby pushing the per-iteration cost of an interior-point method to cubic $O(n^3)$ time and quadratic $O(n^2)$ memory. In [26], omitting some of the overlap constraints made (CTC) easier to solve, but at the cost of also making the reformulation from (SDP) inexact.

In this paper, we apply the *dualization* technique of Löfberg [27] to (CTC) before solving the problem using a general-purpose interior-point solver. As we explain in Section 7, the overlap constraints contribute exactly $\Theta(\omega^4 n)$ nonzero elements to the normal matrix of the dualized problem—a figure that is *linear*

with respect to the number of unknowns. Put more simply, dualization makes it possible to *guarantee sparsity* in the normal equations.

The first main result of this paper, given as Theorem 3 in Section 7, states the overall complexity for the dualized version of clique tree conversion as

$$O(\tau^2 \cdot \omega^{6.5} \cdot n^{1.5} \cdot L) \text{ time and } O(\tau \cdot \omega^4 \cdot n) \text{ memory,}$$

where ω, n and L are the same as in Proposition 1, and τ is related to the treewidth of a certain *intersection graph* G_d (following the terminology of Fulkerson and Gross [28]). Our key insight is to note that, after clique tree conversion and dualization, the intersection graph G_d coincides with the (block) sparsity graph of the normal matrix. If G_d has bounded treewidth, then the normal matrix can be formed and factored in linear time, so the cost of an interior-point iteration is linear time and memory.

One special case where the parameter τ is guaranteed to be bounded is a class of semidefinite programs that we name *decoupled* SDPs, relevant to semidefinite relaxations of combinatorial optimization problems.

Definition 3 (Decoupled) Given index sets $\mathcal{J} = \{J_1, \ldots, J_\ell\}$, we say that the linear constraint $A_i \bullet X = b_i$ is *decoupled* if there exists $J_j \in \mathcal{J}$ and some choice of $A_{i,j} \in \mathbb{S}^{|J_j|}$ such that

$$A_i \bullet X = A_{i,j} \bullet X[J_j, J_j] \qquad \forall X \in \mathbb{S}^n.$$

We say that an instance of (SDP) is *decoupled* if every constraint matrix $A_1, \ldots, A_m$ is decoupled.

Theorem 1 *Let (SDP) be a decoupled SDP, and let $\mathcal{T} = (\mathcal{J}, T)$ be a tree decomposition for the sparsity graph G of $C, A_1, \ldots, A_m$. Then, there exists an algorithm that: 1) converts (SDP) into an instance of (CTC); 2) solves the dualized version of (CTC) to L accurate digits; and 3) recovers a corresponding solution of (SDP) in*

$$O(\omega^{6.5} \cdot n^{1.5} \cdot L) \text{ time and } O(\omega^4 \cdot n) \text{ space} \tag{3}$$

where $\omega \equiv 1 + \mathrm{wid}(\mathcal{T})$ is the clique number of $\mathcal{T}$.

Proof The exact algorithm and detailed proof are both given in Section 7. □

Remark 1 The original, non-dualized version of clique tree conversion is not guaranteed to achieve near-linear time complexity. Indeed, we give an explicit example in Section 6 of a decoupled SDP that forces the original clique tree conversion to attain its worst-case cubic time complexity.

Problems that do not satisfy the decoupled assumption in Theorem 1 can be systematically decoupled by introducing *auxillary variables*, and grouping them together with the existing variables $X_1, \ldots, X_\ell$. Our second main result, given as Theorem 4 in Section 8, states the overall complexity of this auxillary-variables-based approach as

$$O(\gamma_{\max}^3 \cdot \omega^{6.5} \cdot n^{1.5} \cdot L) \text{ time and } O(\gamma_{\max}^2 \cdot \omega^4 \cdot n) \text{ memory,}$$

where ω, n and L are again the same as in Proposition 1, and $\gamma_{\max}$ is the maximum number of auxillary variables added to a single variable group.

In a class of graph-based semidefinite programs known as *network flow SDPs*, the value of $\gamma_{\max}$ can be bounded in terms of parameters of the graph. Such problems frequently arise on physical networks subject to Kirchoff's conservation laws, such as electrical circuits and hydraulic networks.

Definition 4 (Network flow) Given a graph $G = (V, E)$ on n vertices $V = \{1, \dots, n\}$, we say that the linear constraint $A \bullet X = b$ is a *network flow constraint* (at vertex k) if the $n \times n$ constraint matrix A can be rewritten

$$A = \alpha_k e_k e_k^T + \frac{1}{2} \sum_{(j,k) \in E} \alpha_j (e_j e_k^T + e_k e_j^T),$$

in which e_k is the k-th column of the identity matrix and $\{\alpha_j\}$ are scalars. We say that an instance of (SDP) is a *network flow SDP* if every constraint matrix $A_1, \dots, A_m$ is a network flow constraint, and G is the sparsity graph for the objective matrix C.

Theorem 2 *Let (SDP) be network flow SDP on a graph $G = (V, E)$ with n vertices $V = \{1, \dots, n\}$, and let $\mathcal{T} = (\mathcal{J}, T)$ be a tree decomposition for G. Then, there exists an algorithm that solves (CTC) to L accurate digits in*

$$O((\omega + d_{\max} m_k)^{3.5} \cdot \omega^{3.5} \cdot n^{1.5} L) \text{ time and } O((\omega + d_{\max} m_k)^2 \cdot \omega^2 \cdot n) \text{ memory}$$

where:

- $\omega \equiv 1 + \text{wid}(\mathcal{T})$ *is the clique number of* $\mathcal{T}$
- $d_{\max}$ *is the maximum degree of the tree* T,
- m_k *is the maxmimum number of network flow constraints at any vertex* $k \in V$.

Proof The actual algorithm and detailed proof are both given in Section 8. □

1.4 Related work

Conversion methods. Conversion methods like clique tree conversion, which split a size-n semidefinite cone into numerous smaller cones, were first proposed in [12,13], and have been refined by [23] and implemented as the SparseCoLO MATLAB package [29]. They have found extensive use for the ACOPF relaxation problem arising in power systems applications (with n on the order of several thousands), reducing interior-point solution time from tens of hours to tens of minutes [30,15,24,16]. Further speed-ups were obtained by dropping some of the overlap constraints, though the reformulation may no longer be exact [26].

All of these previous works have been empirical in nature, justifying efficiency through exhaustive numerical simulations. Our work contributes to the existing literature by providing a *theoretical guarantee* for efficiency. We

show that if a given SDP satisfies certain sparsity assumptions, then the dualized version of clique tree conversion is guaranteed to solve the problem in $O(n^{1.5}L)$ time and $O(n)$ memory—linear in the number of accurate digits L and near-linear in the problem size n. In particular, we guarantee this efficiency figure for the ACOPF relaxation problem (posed on a network with bounded treewidth and bounded degree) using a modified version of dualized clique tree conversion.

Our proposed method is also highly efficient in practice. In Section 9, we use dualized clique tree conversion and MOSEK [21] to solve the MAX 3-CUT and Lovas Theta SDPs on a suite of 40 power system graphs (the largest of which contains $n = 13659$ nodes) to $L \geq 6$ accurate digits in less than a minute (including the pre- and post-processing steps). The empirical time complexity for these two problems is just $T \approx 10^{-4} \cdot n^{1.1} \cdot L$ seconds. We also use the modified verison of dualized clique tree conversion to solve the ACOPF SDP in less than four minutes, with an empirical time complexity of $T \approx 2.3 \times 10^{-4} \cdot n^{1.3} \cdot L$.

Auxillary variables. A key idea in the proof of Theorem 2 is to systematically reduce the treewidth of a certain intersection graph by introducing auxillary variables. The technique we use in Section 8 is inspired by [31,32], and is chosen primarily for its simplicity. A more sophisticated technique by Bienstock and Munoz [33] introduces auxillary variables by splitting existing variables corresponding to high-degree nodes in the intersection graph. This latter technique uses fewer auxillary variables for high-degree intersection graphs, and can be used to improve the constants in Theorem 4.

First-order methods. Conversion methods can be used alongside first-order methods like ADMM [32,31] or alternating projections [34], in lieu of interior-point methods. In this context, the main advantage of first-order methods is that it is relatively easy to guarantee a low per-iteration cost of linear $O(n)$ time and memory. Unfortunately, convergence guarantees are significantly weakened. Of the methods with global convergence bounds, only sublinear error rates of $O(1/k^{\alpha})$ can be established [35,36]; an approximation with L accurate digits would require exponential $O(\exp(L))$ iterations to compute.

Nonsymmetric interior-point method. Another popular way to exploit the bounded treewidth property is to optimize directly on the nonsymmetric cone of sparse positive semidefinite matrices using the nonsymmetric interior-point method of Andersen, Dahl, and Vandenberghe [14,37]; an implementation is available as the CVXOPT python package [38]. Compared to conversion methods, the main advantage of the nonsymmetric interior-point method is that it avoids introducing a possibly large number of overlap constraints. On the other hand, the algorithm works by explicitly forming and factorizing the $m \times m$ fully-dense normal matrix. Consequently, its near-linear time complexity of $O(m^3 n^{0.5} + n^{1.5} m^2)$ becomes cubic time when the number of constraints m is on the same order of magnitude as n.

2 Applications

2.1 Decoupled SDPs

Several standard SDP problems are decoupled SDPs, particularly those that arise as the convex relaxation of combinatorial optimization problems. According to Theorem 1, all of these problems can all be solved in near-linear time and linear memory.

MAXCUT and MAX k-CUT relaxations. Let C be the (weighted) Laplacian matrix for a graph $G = (V, E)$ with n vertices. Frieze and Jerrum [39] proposed a randomized algorithm to solve MAX k-CUT with an approximation ratio of $1 - 1/k$ based on solving

$$\begin{aligned} \text{maximize} \quad & \frac{k-1}{2k} C \bullet X & & \text{(MkC)} \\ \text{subject to} \quad & X[i,i] = 1 & \forall i \in \{1,\dots,n\} & \\ & X[i,j] \geq \frac{-1}{k-1} & \forall (i,j) \in E & \\ & X \succeq 0 & & \end{aligned}$$

based on the Goemans and Williamson 0.878 algorithm [2], which is recovered by setting $k = 2$ and removing the redundant constraint $X[i,j] \geq -1$. In both cases, observe that each constraint in (MkC) affects just a single matrix element in X, so the problem is trivially decoupled.

Lovasz Theta. Let $G = (V, E)$ be a graph with n vertices. The Lovasz number $\vartheta(G)$ [1] is the solution to the following dual semidefinite program (here, e_j is the j-th column of the size-n identity matrix and $\mathbf{1}$ is the size-n vector-of-ones) with variables $\lambda \in \mathbb{R}$ and $y_{i,j} \in \mathbb{R}$ for $(i,j) \in E$:

$$\begin{aligned} \text{minimize} \quad & \lambda & & \text{(LT)} \\ \text{subject to} \quad & \mathbf{1}\mathbf{1}^T - \sum_{(i,j)\in E} y_{i,j}(e_i e_j^T + e_j e_i^T) \preceq \lambda I & & \end{aligned}$$

and serves as a bound for a number of graph theoretical quantities that are NP-hard to compute. Problem (LT) is not decoupled. However, given that $\vartheta(G) \geq 1$ holds for all graphs G, we may divide the linear matrix inequality through by λ, redefine $y \leftarrow y/\lambda$, apply the Schur complement lemma, and take the Lagrangian dual to yield a sparse formulation

$$\begin{aligned} \text{minimize} \quad & \begin{bmatrix} I & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \bullet X & & \text{(LT}') \\ \text{subject to} \quad & X[i,j] = 0 & \forall (i,j) \in E & \\ & X[n+1,n+1] = 1 & & \\ & X \succeq 0. & & \end{aligned}$$

Like in (MkC), each constraint contains just a single matrix element in X, so the problem is again trivially decoupled.

MAX BISECTION. Again, let C be the (weighted) Laplacian matrix for a graph $G = (V, E)$ with n vertices. The MAX BISECTION relaxation of Frieze and Jerrum [39] solves

$$\begin{aligned} \text{minimize} \quad & \frac{1}{4} C \bullet X \\ \text{subject to} \quad & X[i,i] = 1 \quad \forall i \in \{1, \ldots, n\} \\ & \mathbf{1}\mathbf{1}^T \bullet X \leq 0, \\ & X \succeq 0, \end{aligned} \tag{MB}$$

where $\mathbf{1}$ is the size-n vector-of-ones. The problem is not decoupled due to the presence of the constraint $\mathbf{1}\mathbf{1}^T \bullet X \leq 0$. However, noting that the Lagrangian multiplier for this constraint is always positive (otherwise, we may simply delete the constraint to yield the usual MINCUT relaxation), we can apply the Schur complement lemma to the dual of (MB) to yield

$$\begin{aligned} \text{minimize} \quad & \begin{bmatrix} C/4 & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \bullet X \\ \text{subject to} \quad & X[i,i] = 1 \quad \forall i \in \{1, \ldots, n\} \\ & X[n+1, n+1] = 0, \\ & X \succeq 0, \end{aligned} \tag{MB$'$}$$

Each constraint in (MB$'$) affects X only at a single element along its diagonal, so the problem is again trivially decoupled.

2.2 Network flow SDPs

The most prominent example of a network flow SDP is the semidefinite relaxation of the nonconvex AC optimal power flow (ACOPF) problem [40], which plays a vital role in the operations of an electric power system. The problem is formulated to impose a small number of network flow constraints on all the vertices in a network. Power systems generally have bounded treewidth and bounded degree [16], so we would expect near-linear time complexity via the algorithm in Theorem 2. Our numerical results in Section 9 find this to indeed be the case.

Optimal power flow (OPF). The OPF problem is a large-scale optimization problem that plays a vital role in the operation of an electric power system. Variants of this problem is solved by the system operator from every few minutes to every several months, in order to ensure a reliable and inexpensive supply of electricity. The OPF problem is well-known to be nonlinear, nonconvex, and NP-hard in general, but can be relaxed into an SDP problem using standard techniques:

$$\begin{aligned} \text{minimize} \quad & C \bullet X \\ \text{subject to} \quad & b_i^\ell \leq A_i \bullet X \leq b_i^u \quad \forall i \in \{1, \ldots, m\} \\ & X \succeq 0. \end{aligned} \tag{OPF}$$

Here, X is a Hermitian positive semidefinite matrix, the constraint matrices $A_1, \ldots, A_m$ measure physical quantities within the system (e.g. voltage magnitudes, power flows, and power injections), and $b^\ell, b^u \in \mathbb{R}^m$ provide a corresponding list of upper- and lower-bounds. If the solution matrix $\hat{X}$ is rank-one, then the relaxation (OPF) is exact, and a globally-optimal solution to the original NP-hard problem can be extracted by factoring $\hat{X} = \hat{v}\hat{v}^T$.

A surprising feature of the OPF problem is that the SDP relaxation (OPF) is often exact [41]. Even when this is not the case, the problem can be slightly perturbed to guarantee an exact relaxation; a solution of the perturbed problem can then serve as a near-globally-optimal solution to the original problem. This perturbation technique, known as penalized SDP, was tested on IEEE benchmarks, Polish, Ontario and New York Grids, over 7000 settings, where it always obtained a solution with global optimality guarantee exceeding 99% [24, 16]. It is therefore safe to say that the practical cost of solving a size-n OPF is usually about the same as that of solving a size-n SDP.

3 Preliminaries

3.1 Definitions, notations, basic results

Matrix sets. The sets $\mathbb{R}^{n\times n} \supset \mathbb{S}^n \supset \mathbb{S}^n_+ \supset \mathbb{S}^n_{++}$ refer to the $n\times n$ real matrices, real-symmetric matrices, positive semidefinite matrices, and positive definite matrices, respectively. These four spaces are endowed with the trace inner product $X \bullet Y \equiv \operatorname{tr} X^T Y$ and Euclidean (Frobenius) norm $\|X\|_F^2 = X \bullet X$. We will frequently write $X \succeq Y$ (resp. $X \succ Y$) to mean $X - Y \in \mathbb{S}^n_+$ (resp. $X - Y \in \mathbb{S}^n_{++}$).

Vectorization and Kronecker. The vectorization of a matrix is the column-stacking operation

$$\operatorname{vec} X = [X_{1,1}, \ldots, X_{n,1}, X_{1,2}, \ldots, X_{n,2}, \ldots, X_{n,n}]^T,$$

and the Kronecker product

$$A \otimes B = \begin{bmatrix} A_{1,1}B & \cdots & A_{1,n}B \\ \vdots & \ddots & \vdots \\ A_{n,1}B & \cdots & A_{n,n}B \end{bmatrix},$$

is defined to satisfy the Kronecker identity

$$\operatorname{vec}(AXB) = (B^T \otimes A)\operatorname{vec} X.$$

Matrix elements and submatrices. Given $X \in \mathbb{R}^{n\times n}$ and two indices $i, j \in \{1, \ldots, n\}$, the notation $X[i, j]$ indexes the (i, j)-th element of X. Likewise, given two index *sets* $I, J \subseteq \{1, \ldots, n\}$, the notation $X[I, J]$ indexes the (I, J)-th *submatrix* of X. We use $\operatorname{nnz}(X)$ to count the total number of nonzero elements $X[i, j] \neq 0$ in X.

Sparsity pattern. A sparsity pattern is a set of indices $E = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ referring to the positions of its strictly lower-triangular nonzeros (i.e. excluding the diagonal elements). We denote $\mathbb{R}^{n\times n}_E \subset \mathbb{R}^{n\times n}$ (resp. $\mathbb{S}^n_E \subset \mathbb{S}^n$) as the space of $n \times n$ real matrices (resp. real symmetric matrices) with sparsity pattern E. Every matrix $X \in \mathbb{R}^{n\times n}_E$ has zero elements $X[i, j] = X[j, i] = 0$ for all off-diagonal (i, j) satisfying $i \neq j$ and $(i, j) \notin E$, though the structurally nonzero elements $X[i, j]$ and $X[j, i]$ for $(i, j) \in E$ are allowed to be numerically zero. (The diagonal elements of X are unconstrained.)

Graph theory. A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$. The elements of V are known as the *vertices* of G, and the elements of E are known as the *edges* of G. Given a graph G, we will refer to its *vertex set* as $V(G)$ and *edge set* as $E(G)$. The *adjacency matrix* of G is a $|V| \times |V|$ binary matrix $A = A^T$ with $A[i, j] = 1$ iff $(i, j) \in E$ and $A[i, j] = 0$ otherwise. The *line graph* $\mathcal{L}(G)$ of G is a graph on $|E|$ vertices such that: 1) each vertex of $\mathcal{L}(G)$ represents an edge of G; and 2) two vertices of $\mathcal{L}(G)$ are adjacent if and only if their corresponding edges share a vertex.

Sparsity graph. We identify each sparsity pattern E with an undirected graph $G = (V, E)$ on n vertices $V = \{1, 2, \ldots, n\}$. Per Definition 1, the sparsity graph of a given $n \times n$ matrix M (resp. a collection of matrices $M_1, \ldots, M_m$) is the graph corresponding to the *minimum* sparsity pattern for M (resp. $M_1, \ldots, M_m$). That is, the sparsity pattern E satisfying $M \in \mathbb{R}^{n\times n}_E$ (resp. $M_1, \ldots, M_m \in \mathbb{R}^{n\times n}_E$) with the fewest number of elements $|E|$. Note that due to the minimum sparsity requirement, it is possible for M to have sparsity pattern E without also having sparsity graph $G = (V, E)$.

Chordal graphs and completions. The graph $G = (V, E)$ is said to be *chordal* if it does not contain an induced cycle with length greater than three. If G is not chordal, then we may add edges to it until it becomes chordal; the resulting edge set $E^* \supseteq E$ is called a *chordal completion* (or *triangulation*) of E, and the associated graph $G^* = (V, E^*)$ is the chordal completion of G.

It is a well-known result that G is chordal if and only if it has a *perfect elimination ordering*: an ordering of the vertices of the graph such that, for each vertex $v \in V$, v and the neighbors of v that occur after v in the order form a clique [28]. The ordering can be explicitly computed in $O(|E|)$ time and space using a breadth-first search [42].

Tree decompositions. Tree decompositions (Definition 2) are also known as *junction trees* in statistics and machine learning. They are slightly more general than *clique trees*, which further imposes a *maximality* clause: no two subsets $J_i, J_j \in \mathcal{J}$ satisfy $J_i \subseteq J_j$ or $J_i \supseteq J_j$.

Tree decompositions are closely related to chordal completions. Given a tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ on G, we obtain a chordal completion G^* of G by interconnecting all the vertices in each $J_j \in \mathcal{J}$ [43]; see also [25, Theorem 3.8]. The maximum clique size (i.e. the size of the largest clique) of G^* is $\omega = 1+\mathrm{wid}(\mathcal{T})$. In the other direction, given a chordal completion G^* of G with maximum clique size ω, we recover a (possibly different) tree decomposition

$\mathcal{T} = (\mathcal{J}, T)$ with $\text{wid}(\mathcal{T}) = \omega - 1$ by eliminating G using the perfect elimination ordering of $G*$ [44]; see also [25, Section 4.3] and Section 3.3.

3.2 Sparse Cholesky factorization

Given an $n \times n$ positive definite matrix S, the standard approach for solving the linear system of equations

$$Sx = b, \tag{4}$$

is to compute the unique *Cholesky factor* R of S satisfying

$$RR^T = S, \quad R \text{ is lower-triangular}, \quad R_{i,i} > 0 \quad \forall i, \tag{5}$$

in $O(n^3)$ time and $O(n^2)$ memory, and to backsolve two triangular systems $Ru = r$ and $R^T x = u$ in $O(n^2)$ time and memory.

In the case that S is sparse, sparse Cholesky algorithms can solve (4) at significantly reduced costs by avoiding arithmetic operations with, and the explicit storarge of, the zero elements in R.

Proposition 2 *There exists a sparse Cholesky factorization algorithm that:*

- *factors S into RR^T in $O(\omega^2 n)$ time and $O(\omega n)$ memory; and*
- *solves $Ru = r$ for u and $R^T x = u$ for x in $O(\omega n)$ time and memory*

where $\omega = \max_j |J_j|$ is the maximum number of nonzero elements in a single column of R.

In general, the process of factoring S into R creates new nonzeros in positions where S is zero, known as *fill-in*, which adversely affect complexity. Fill-in can be reduced by symmetrically reordering the rows and columns of S, i.e. to solve the symmetrically permutated problem

$$(\Pi S \Pi^T)(\Pi x) = \Pi r$$

in which Π is a *fill-reducing* permutation matrix.

The problem of computing a choice of Π to minimize the parameter ω in Proposition 2 coincides with the *treewidth* problem [45]. (This link becomes more obvious after we introduce the elimination tree in Section 3.3.) The treewidth problem is also NP-complete [46], but has a linear time fixed parameter algorithm due to Bodlaender [19]. Combining this algorithm with Proposition 2 yields the following folklore result.

Proposition 3 *Let the sparsity graph G of S have bounded treewidth. Then, there exists an algorithm that solves $Sx = b$ for x in $O(n)$ time and memory.*

Proof We use the linear time algorithm of Bodlaender [19] to compute a tree decomposition $\mathcal{T}$ for G satisfying $\text{wid}(\mathcal{T}) = \text{tw}(G) = O(1)$. Next, we compute the chordal completion G^* associated with $\mathcal{T}$, and the perfect elimination ordering Π associated with G^*, both in linear time. This particular choice of Π yields up to $\omega = 1 + \text{wid}(\mathcal{T}) = O(1)$ nonzero elements per column of R. Substituting this ω into Proposition 2 yields the desired result. □

In practice, it is far more efficient to compute Π using one of the many heuristics developed for the minimum fill-in problem [45]. In fact, for bounded-degree graphs, the nested dissection heuristic generates a provably good choice of Π that minimizes ω to a factor of $O(\log(n))$ of the optimal [47,48,45].

3.3 The elimination tree as a tree decomposition

The problem of selecting a good fill-reducing permutation Π was studied in detail using a graph theoretical framework by Rose [49]. We identify S with its sparsity graph $G = (V, E)$, and a permuted sparsity graph $G_\pi = (V, E_\pi)$ for each permutation matrix Π. Observe that every G_π is *isomorphic* to G, meaning that we obtain G_π from G by relabeling its vertices.

Now, fix the choice of Π, and compute the Cholesky factor R satisfying $\Pi S \Pi^T = RR^T$. The sparsity pattern E_π^* of R is related to the sparsity pattern E_π of $\Pi S \Pi^T$ by two fundamental properties (excluding accidental cancellation):

$$(i,j) \in E_\pi \quad \Longrightarrow \quad (i,j) \in E_\pi^* \qquad \forall i > j, \tag{6}$$

$$(i,k) \in E_\pi^*, \quad (j,k) \in E_\pi^* \quad \Longrightarrow \quad (i,j) \in E_\pi^* \qquad \forall i > j > k. \tag{7}$$

The two properties characterize $G_\pi^* = (V, E_\pi)$ as a *chordal completion* of G_π, and hence also of G up to isomorphism [49,50].

We define the *column index sets* $J_1, \ldots, J_\ell \subseteq \{1, \ldots, n\}$ of R as

$$J_j \equiv \{i \in \{1, \ldots, n\} : R[i,j] \neq 0\}, \tag{8}$$

and the *elimination tree* $T_\pi = (V, F_\pi)$ associated with R via the parent pointers $p : \{1, \ldots, n\} \to \{1, \ldots, n\}$

$$p(j) = \begin{cases} j & |J_j| = 1, \\ \min_i\{i > j : i \in J_j\} & |J_j| > 1, \end{cases} \tag{9}$$

(Our convention assigns each root node to be its own parent.) More concretely, we define the edges F_π to link each node to its parent

$$F_\pi = \{(1, p(1)), (2, p(2)), \ldots, (n, p(n))\}.$$

Then, the column index sets $\mathcal{J}_\pi = \{J_1, \ldots, J_n\}$ and the elimination tree $T_\pi = (\{1, \ldots, n\}, F_\pi)$ constitute a *tree decomposition* $\mathcal{T}_\pi = (\mathcal{J}_\pi, T_\pi)$ for G [44].

Here, observe that the clique number $\omega \equiv 1 + \mathrm{wid}(\mathcal{T})$ coincides with the maximum number of nonzeros in each column of R, as in

$$\omega = \max\{|J_1|, |J_2|, \ldots, |J_n|\}.$$

Hence, if ω is bounded, then $\mathcal{T}_\pi = (\mathcal{J}_\pi, T_\pi)$ is a bounded width tree decomposition for the sparsity graph G of S. The following is an immediate consequence.

Proposition 4 *Given permutation matrix Π, let the Cholesky factor R of $\Pi S \Pi^T$ contain at most ω nonzero elements per column. Then, the sparsity graph G of S has treewidth bounded* $\mathrm{tw}(G) \leq \omega - 1$.

3.4 Interior-point methods on symmetric cones

Given a data matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, data vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and a prescribed convex cone $\mathcal{K} \subseteq \mathbb{R}^n$, we may define a *cone linear program* (CLP) written in *standard canonical form* as the primal-dual pair

$$\begin{aligned} \text{(P)}\quad &\text{minimize} && c^T x \\ &\text{subject to} && \mathbf{A}x = b, \\ & && x \in \mathcal{K}. \end{aligned} \qquad \begin{aligned} \text{(D)}\quad &\text{maximize} && b^T y \\ &\text{subject to} && \mathbf{A}^T y + s = c, \\ & && s \in \mathcal{K}_*. \end{aligned} \tag{10}$$

Here, $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are the vectors of primal and dual decision variables, and $\mathcal{K}_*$ is the dual cone of $\mathcal{K}$. In this paper, we restrict $\mathcal{K}$ to the following three standard cones (and their Cartesian products):

$$\begin{aligned} \mathcal{K}^{\text{LP}} &= \{x \in \mathbb{R}^n : x_i \geq 0 \quad \text{for all } i \in \{1,\dots,n\}\} \equiv \mathbb{R}^n_+, \\ \mathcal{K}^{\text{SOCP}} &= \{x \in \mathbb{R}^n : x_1 \geq \sqrt{x_2^2 + x_3^2 + \cdots + x_n^2}\}, \\ \mathcal{K}^{\text{SDP}} &= \{\text{vec}\,(X) : X \in \mathbb{R}^{p\times p} \text{ is symmetric positive semidefinite}\} \equiv \text{vec}\,(\mathbb{S}^p_+), \end{aligned} \tag{11}$$

where $p \approx \sqrt{2n}$ is the integer satisfying $\dim(\mathbb{S}^p) = p(p+1)/2 = n = \dim(\mathbb{R}^n)$. Setting $\mathcal{K}$ in (10) as one of (11) yields a linear program (LP), a second-order cone program (SOCP), and a semidefinite program (SDP) respectively. All three cones are *symmetric* [51] (also known by Nesterov and Todd as *self-scaled* [52]), meaning that they satisfy the self-adjoint property $\mathcal{K} = \mathcal{K}_*$ by construction.

The *interior-point method* is the standard approach for solving a CLP on a symmetric cone. In practice, primal-dual methods [53,52,54,20] are preferred for their superior numerical stability and their ability to converge at a super-linear rate. For simplicity and conciseness, however, we will focus our attention on the *primal barrier method* from Nesterov and Nemirovski's original monograph [9], while noting that all of these methods share the same iteration bounds and per-iteration costs.

The primal barrier method begins with an initial point satisfying $\mathbf{A}x = b$ and lying in the strict interior of the cone $x \in \text{Int}(\mathcal{K})$. Using this point as an initial point, the method proceeds to solve the following problem

$$\begin{aligned} &\text{minimize } t \cdot c^T x + F(x) \\ &\text{subject to } \mathbf{A}x = b, \end{aligned} \tag{12}$$

using Newton's method. Here, $t \geq 1$ is a fixed barrier parameter, and F is a self-concordant barrier function for the cone $\mathcal{K}$. The standard choice for the

three primitive cones (11) are respectively

$$F^{\mathrm{LP}}(x) = -\sum_{i=1}^{n} \log x_i, \tag{13}$$
$$F^{\mathrm{SOCP}}(x) = -\log\left(x_1^2 - \sum_{i=2}^{n} x_i^2\right),$$
$$F^{\mathrm{SDP}}(\mathrm{vec}\,(X)) = -\log\det(X).$$

The standard choice for the Cartesian product $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ of two cones $\mathcal{K}_1$ and $\mathcal{K}_2$ with barrier functions $F^{(1)}$ and $F^{(2)}$ is their sum $F(x_1, x_2) = F^{(1)}(x_1) + F^{(2)}(x_2)$. After (12) is solved to sufficient accuracy, t is increased by a fixed factor, and the method proceeds to solve (12) using Newton's method, with the previous solution as the new initial point.

Using the homogenous self-dual embedding technique [22], a suitable initial point satisfying $\mathbf{A}x = b$ and $x \in \mathrm{Int}(\mathcal{K})$ can always be found. Moreover, if t is increased by a factor of $1 + 1/\sqrt{\nu(\mathcal{K})}$, then Newton's method requires just $O(1)$ iterations to converge to sufficient accuracy. Here, $\nu(\mathcal{K})$ denotes the *order* of the cone $\mathcal{K}$. The three cones in (11) each have order

$$\nu(\mathcal{K}^{\mathrm{LP}}) = n, \qquad \nu(\mathcal{K}^{\mathrm{SOCP}}) = 2, \qquad \nu(\mathcal{K}^{\mathrm{SDP}}) = p.$$

The order of a Cartesian product $\mathcal{K} = \mathcal{K}^{(1)} \times \mathcal{K}^{(2)}$ between two symmetric cones $\mathcal{K}^{(1)}$ and $\mathcal{K}^{(2)}$ is the sum of their orders $\nu(\mathcal{K}) = \nu(\mathcal{K}^{(1)}) + \nu(\mathcal{K}^{(2)})$. Combining the two facts above yields the following classic result first established by Nesterov and Nemirovski [9] and Alizadeh [10].

Lemma 1 (Iteration bound) *Given an instance of (10) with a full-rank $m \times n$ data matrix $\mathbf{A}$ (i.e. $\mathrm{rank}(\mathbf{A}) = m \leq n$), and where $\mathcal{K}$ is a Cartensian product of the three cones in (11), there exists an interior-point method that generates an iterate $(x, y, s) \in \mathcal{K} \times \mathbb{R}^m \times \mathcal{K}_*$ satisfying $\mathbf{A}^T x = b$, $\mathbf{A}y + s = c$ and $x^T s \leq \epsilon$ in*

$$O(\sqrt{\nu(\mathcal{K})} \log \epsilon^{-1}) \text{ iterations,}$$

where $\nu(\mathcal{K})$ denotes the order of the cone $\mathcal{K}$.

3.5 Exploiting sparsity in the normal equations

The cost of each interior-point iteration is dominated by the solution of the $m \times m$ system of linear equations (known as the *normal equations* or the *Schur complement equations*):

$$\mathbf{H}u = \mathbf{A}\nabla^2 F_*(w)\mathbf{A}^T u = r, \tag{14}$$

given iteration- and algorithm-specific vectors r and $w \in \mathcal{K}_*$. Here, F_* is the convex conjugate for F, the self-concordant barrier of the problem cone $\mathcal{K}$.

Lemma 2 (Per-iteration cost) *Under the same conditions as Lemma 1, let* $\omega \equiv \max_i \nu(\mathcal{K}_i^{\text{SDP}})$ *denote the size of the largest SDP subcone in* $\mathcal{K}$*, and let* ℓ *denote the number of SDP subcones in* $\mathcal{K}$*. Then, an interior-point method costs*

$$O(\ell\omega^3 + f(n)) \text{ time and } O(g(n)) \text{ memory,}$$

where $f(n)$ *and* $g(n)$ *are the time and memory costs required to solve (14) for* u*, given* $\mathbf{A} \in \mathbb{R}^{m\times n}$*,* $w \in \mathcal{K}_*$*, and* $r \in \mathbb{R}^m$*.*

In the case that the matrix $\mathbf{A}\mathbf{A}^T$ is sparse and $\mathcal{K} = \mathcal{K}^{\text{LP}}$, the per-iteration cost of the iterior-point method can be substantially reduced by factoring $\mathbf{H}$ using a sparse Cholesky algorithm. Here, the matrix $\nabla^2 F_*(w)$ is diagonal positive definite, so the sparsity graph of $\mathbf{H}$ coincides with the *primal intersection graph* [28] of the CLP (10)

$$G_p = (\{1, \dots, m\}, E_p) \tag{15}$$

$(i,j) \in E_p$ if there exists $k \in \{1,\dots,n\}$ such that $\mathbf{A}[i,k] \neq 0$ and $\mathbf{A}[j,k] \neq 0$.

If G_p has bounded treewidth, then the cost of solving (14) given $\mathbf{H}$ is $O(m)$ due to Proposition 3. The interior-point iteration costs linear $O(n)$ time and memory, and is dominated by the cost of forming the matrix $\mathbf{H}$.

Lemma 3 *Given* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and diagonal* $D \in \mathbb{R}^{n\times n}$*, the cost of forming* $\mathbf{H} = \mathbf{A}D\mathbf{A}^T$ *is* $O(\omega^2 n)$ *time and* $O(\omega n)$ *memory, where* $\omega = \text{tw}(G_p) + 1$ *and* G_p *is the primal intersection graph in (15).*

Proof By properties of the tree decomposition, the largest clique in G_p is size ω, and G_p contains at most ωm edges. Write a_i as the i-th column of $\mathbf{A}$. Since each $a_i a_i^T$ contributes a clique of size $\text{nnz}\,(a_i)$ to G_p, we must have $\text{nnz}\,(a_i) \leq \omega$. Summing up n columns of $\mathbf{A}$ yields $\text{nnz}\,(\mathbf{A}) \leq \omega n$. Adding up n sets of $D_{i,i}a_i a_i^T$ costs $\omega^2 n$ operations. The total complexity is $O(\omega^2 n)$ time and $\text{nnz}\,(\mathbf{A}) + \text{nnz}\,(\mathbf{H}) + \text{nnz}\,(D) = O(\omega n)$ memory. □

If the matrix $\mathbf{A}\mathbf{A}^T$ is sparse and $\mathcal{K} = \mathcal{K}^{\text{SOCP}}$, then the same efficiency result is obtained via a *low-rank update*. Here, the Hessian matrix $\nabla^2 F_*(w)$ is the low-rank perturbation qq^T of a diagonal positive definite matrix D, and the matrix $\mathbf{H}$ is the low-rank perturbation of a positive definite sparse matrix

$$\mathbf{H} = \mathbf{A}(D + qq^T)\mathbf{A}^T = \mathbf{A}D\mathbf{A}^T + (\mathbf{A}q)(\mathbf{A}q)^T$$

Such matrices are frequently inverted using the Sherman–Morrison–Woodbury formula[2]

$$u = \mathbf{H}^{-1}r = \tilde{u} - v(I + v^T\mathbf{A}q)^{-1}v^T r \tag{16}$$

where $\tilde{u}$ and v are the solutions to the following system of equations

$$(\mathbf{A}D\mathbf{A}^T)\begin{bmatrix}\tilde{u} & v\end{bmatrix} = \begin{bmatrix}r & \mathbf{A}q\end{bmatrix}. \tag{17}$$

[2] In practice, the product form Cholesky technique of Goldfarg and Scheinberg [55] is used in lieu of the naïve Sherman–Morrison–Woodbury procedure described above, in order to overcome issues with numerical stability [56,20]. These two techniques have the same complexity figures.

If G_p has bounded treewidth, then factoring $\mathbf{A}D\mathbf{A}^T$, solving (17) for $\tilde{u}$ and v, and applying the formula (16) all costs $O(m)$ time and space. Again, the interior-point iteration costs linear $O(n)$ time and memory, and is dominated by the cost of forming the sparse matrix $\mathbf{A}D\mathbf{A}^T$.

3.6 Dualization

In some problem instances, the matrix $\mathbf{A}^T\mathbf{A}$ may be sparse while $\mathbf{A}\mathbf{A}^T$ is guaranteed to be fully-dense. This is the case, for example, in problems where $\mathbf{A}$ contains many fully-dense columns but no dense rows. Sparsity in $\mathbf{A}^T\mathbf{A}$ may be exploited by the dualization technique of Löfberg [27]. Given a cone linear program in standard canonical form (10), we generate the following pair

$$\begin{array}{lllll} \text{(P)} & \text{minimize} & -b^T x_1 & \text{(D)} \quad \text{maximize} & -c^T y \\ & \text{subject to} & \mathbf{A}^T x_1 + x_2 = c, & \text{subject to} & \mathbf{A}y + s_1 = c, \\ & & x_1 \in \mathbb{R}^m, \quad x_2 \in \mathcal{K}_*. & & -y + s_2 = 0, \\ & & & & s_1 \in \{0\}^m \quad s_2 \in \mathcal{K}. \end{array} \tag{18}$$

Essentially, dualization swaps the roles of the primal problem with the dual problem and vice versa, at the cost of introducing m free variables x_1 in the primal and m equality constraints in the dual. The free variables $x_1 \in \mathbb{R}^m$ and fixed variables $s_1 \in \{0\}^m$ may be embedded into a second-order cone[3] as follows

$$\begin{array}{lllll} \text{(P)} & \text{minimize} & -b^T x_1 & \text{(D)} \quad \text{maximize} & -c^T y \\ & \text{subject to} & \mathbf{A}^T x_1 + x_2 = c, & \text{subject to} & 0 + s_0 = 0 \\ & & \|x_1\| \le x_0, \quad x_2 \in \mathcal{K}_*. & & \mathbf{A}y + s_1 = c, \\ & & & & -y + s_2 = 0, \\ & & & & \|s_1\| \le s_0 \quad s_2 \in \mathcal{K}, \end{array} \tag{19}$$

to result in a cone linear program in standard form, of order $\nu(\mathcal{K}^{\text{SOCP}} \times \mathcal{K}) = 2 + \nu(\mathcal{K})$. An interior-point method solves (19) in $O(\sqrt{\nu(\mathcal{K})})$ iterations. Each interior-point iteration is dominated by the cost of forming and factoring the

[3] An older technique embeds $x_1 \in \mathbb{R}^m$ and $s_1 \in \{0\}^m$ into the linear cone, by splitting $x_1 = x_1^+ - x_1^-$ where $x_1^+, x_1^- \ge 0$ and fixing $0 \le s_1 \le 0$; see [57,58,59]. We prefer the second-order cone embedding because it does not substantially increase the order of the dualized problem, and also because our experiments found it to be more numerically stable.

normal matrix

$$\begin{aligned}\mathbf{H} &= \begin{bmatrix}0 & \mathbf{A}^T\end{bmatrix} \nabla^2 F^{\text{SOCP}}(w_1) \begin{bmatrix}0 \\ \mathbf{A}\end{bmatrix} + \nabla^2 F^{\mathcal{K}}(w_2), \\ &= \begin{bmatrix}0 & \mathbf{A}^T\end{bmatrix} \left(\begin{bmatrix}d_0 & 0 \\ 0 & D_1\end{bmatrix} + \begin{bmatrix}q_0 \\ q_1\end{bmatrix} \begin{bmatrix}q_0 \\ q_1\end{bmatrix}^T \right) \begin{bmatrix}0 \\ \mathbf{A}\end{bmatrix} + \nabla^2 F^{\mathcal{K}}(w_2), \\ &= \left(\mathbf{A}^T D_1 \mathbf{A} + \nabla^2 F^{\mathcal{K}}(w_2)\right) + (\mathbf{A}^T q_1)(\mathbf{A}^T q_1)^T, \end{aligned} \tag{20}$$

where F^{SOCP} is the self-concordant barrier for the second-order cone used to embed the free and fixed variables, and $F^{\mathcal{K}}$ is the self-concordant barrier for problem cone $\mathcal{K}$ from before. In the case that $\nabla^2 F^{\mathcal{K}}$ is diagonal or the low-rank perturbation of a diagonal matrix (i.e. if $\mathcal{K} = \mathcal{K}^{\text{LP}}$ or $\mathcal{K} = \mathcal{K}^{\text{SOCP}}$), the matrix $\mathbf{H}$ is the low-rank perturbation of the sparse matrix $\mathbf{A}^T D_1 \mathbf{A}$, whose sparsity graph matches the *dual intersection graph* of the CLP:

$$G_d = (\{1, \dots, n\}, E) \tag{21}$$

$(i,j) \in E$ if there exists $k \in \{1, \dots, m\}$ such that $\mathbf{A}[k,i] \neq 0$ and $\mathbf{A}[k,j] \neq 0$.

If G_d has bounded treewidth, then the cost of solving $\mathbf{H}u = r$ is linear $O(n)$ time and memory, attained by factoring the sparse portion of the matrix and then making a low-rank update. The corresponding interior-point iteration is also linear $O(n)$ time and memory, and is dominated by the cost of forming the sparse portion of the matrix $\mathbf{H}$.

4 Clique tree conversion

In this section, we review the *clique tree conversion* technique of Fukuda and Nakata et al. [12,13], and the *positive semidefinite matrix completion* algorithm of Andersen, Dahl, and Vandenberghe [14,25]. It is helpful to view the former as the *preprocessor* that reformulates (SDP) into the reduced complexity problem (CTC), and the latter as the *postprocessor* that recovers a solution to (SDP) from a solution of (CTC). Our description here is given entirely using linear algebra, to facilitate implementation via high-level function calls to standard numerical linear algebra libraries.

4.1 Chordal conversion

Define $G = (\{1, \dots, n\}, E)$ as the sparsity graph associated with the data matrices $C, A_1, \dots, A_m \in \mathbb{S}^n_E$, and let $\mathcal{T} = (\mathcal{J}, T)$ with $\mathcal{J} = \{J_1, \dots, J_\ell\}$ be a tree decomposition for G. Recall from Section 3 that a good choice of $\mathcal{T} = (\mathcal{J}, T)$ with low width may be computed using standard linear algebra routines for sparse Cholesky factorization.

We begin by noting that (SDP) is a linear optimization over the matrix elements $X[i,j]$ for $(i,j) \in E$ that "directly interacts" with the problem data.

This observation is made explicit by defining Z to be the projection of X onto the sparsity pattern E:

$$Z[i,j] = P_E(X)[i,j] \equiv \begin{cases} X[i,j] & (i,j) \in E, \\ 0 & (i,j) \notin E, \end{cases}$$

and rewriting each linear objective and constraint in (SDP) in terms of Z:

$$C \bullet X \overset{\text{(a)}}{=} P_E(C) \bullet X \overset{\text{(b)}}{=} C \bullet P_E(X) \overset{\text{(c)}}{=} C \bullet Z. \tag{22}$$

Here, (a) follows $C \in \mathbb{S}^n_E$, (b) is due to the self-adjoint property of projection operators, and (c) is by definition. This same line of reasoning also allows us to rewrite $A_i \bullet X = A_i \bullet Z$ for all $i \in \{1, \ldots, m\}$.

The remaining matrix elements in X participate indirectly via the nonnegativity constraint $X \succeq 0$, insuring that Z can be made positive semidefinite by completing its zero elements with nonzero values:

$$\text{there exists } X \succeq 0 \text{ such that } X[i,j] = Z[i,j] \text{ for all } (i,j) \in E. \tag{23}$$

In other words, Z is constrained to have a *positive semidefinite matrix completion*. If the sparsity pattern E is *chordal*, then the condition can be enforced by constraining select principal submatrices to be positive semidefinite, due to a classic result by Grone et al. [60].

Proposition 5 *Let the sparsity pattern E be chordal. Then, we have*

$$Z = P_E(X), \quad X \succeq 0 \qquad \Longleftrightarrow \qquad Z[J_j, J_j] \succeq 0 \quad \forall j \in \{1, \ldots, \ell\}, \tag{24}$$

in which $J_1, \ldots, J_\ell$ are all of the cliques in the corresponding sparsity graph $G = (\{1, \ldots, n\}, E)$.

Proof The original proof is due to Grone et al. [60]. The reader is referred to [25, Theorem 10.1] for a simplified proof using linear algebra arguments. □

Let $G^* = (\{1, \ldots, n\}, E^*)$ be the chordal completion of G associated with $\mathcal{T}$, obtained by interconnecting the vertices within each node $J_j \in \mathcal{J}$ of our tree decomposition $\mathcal{T}$. Viewing (SDP) as a linear optimization over the matrix elements $X[i,j]$ for $(i,j) \in E^*$, we define $Z = P_{E^*}(X)$ and rewrite $C \bullet X = C \bullet Z$ and each equation $A_i \bullet X = A_i \bullet Z$, using the same reasoning as in (22). Then, substituting (24) into (SDP) yields the following reduced-complexity problem

$$\begin{array}{lll} \text{minimize} & C \bullet Z & \\ \text{subject to} & A_i \bullet Z = b_i & \forall i \in \{1, \ldots, m\}, \\ & Z[J_j, J_j] \succeq 0 & \forall J_j \in \mathcal{J}. \end{array} \tag{25}$$

Observe that the dense matrix variable $X \in \mathbb{S}^n$ has been eliminated and replaced by a sparse surrogate $Z \in \mathbb{S}^n_{E^*}$. The number of decision variables has been reduced from $\frac{1}{2}n(n+1) = O(n^2)$ to $|E^*| \leq \omega n = O(n)$.

4.2 Clique tree conversion

In order to induce sparsity in an interior-point solution of (25), we split the "global" matrix variable $Z \in \mathbb{S}^n_{E*}$ into "local" matrix variables $X_1, \dots, X_\ell$ satisfying

$$X_j = Z[J_j, J_j] \succeq 0 \qquad \forall J_j \in \mathcal{J}. \tag{26}$$

These local variables $X_1, \dots, X_\ell$ are constrained by the need for their overlapping elements over the global variable Z to agree. Naively enforcing this agreement without the global variable Z would incur $\frac{1}{2}\ell(\ell-1) = O(n^2)$ pairwise comparisons of the form

$$N_{J_i J_j}(X_j) = N_{J_j J_i}(X_i) \qquad \forall i, j \in \{1, 2, \dots, \ell\}$$

where we recall that the linear operator $N_{J_i J_j}(\cdot)$ outputs the overlapping elements of two principal submatrices, given the latter as the argument:

$$N_{J_i J_j}(Z[J_j, J_j]) = Z[J_i \cap J_j,\ J_i \cap J_j] = N_{J_j J_i}(Z[J_i, J_i]).$$

However, the running intersection property of the tree decomposition allows us to reduce the number of pairwise comparisons to $\ell - 1$.

Proposition 6 *Let $(\mathcal{J}, T)$ be a tree decomposition with tree nodes $\mathcal{J} = \{J_1, \dots, J_\ell\}$. Then, given $X_1, \dots X_\ell$ where $X_j \in \mathbb{S}^{|J_j|}$, we have*

$$X_j = Z[J_j, J_j] \quad \forall J_j \in \mathcal{J} \qquad \Longleftrightarrow \qquad N_{J_i J_j}(X_j) = N_{J_j J_i}(X_i) \quad \forall (i,j) \in E(T) \tag{27}$$

where $E(T)$ refers to the edges of the tree T.

Proof See Fukuda and Nakada et al. [12,13]. □

Finally, we declare "local" versions $\{C_j\}$ and $\{A_{i,j}\}$ of the objective C and constraint matrices $\{A_j\}$ satisfying

$$\begin{aligned} C_1 \bullet Z[J_1, J_1] + C_2 \bullet Z[J_2, J_2] + \cdots + C_\ell \bullet Z[J_\ell, J_\ell] &= C \bullet Z, \\ A_{i,1} \bullet Z[J_1, J_1] + A_{i,2} \bullet Z[J_2, J_2] + \cdots + A_{i,\ell} \bullet Z[J_\ell, J_\ell] &= A_i \bullet Z. \end{aligned} \tag{28}$$

Substituting (28), (27) and (26) allows us to eliminate the global variable Z in (25), thereby resulting in

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{\ell} C_j \bullet X_j \\ \text{subject to} \quad & \sum_{j=1}^{\ell} A_{i,j} \bullet X_j = b_i && \forall i \in \{1, \dots, m\}, \\ & N_{J_i J_j}(X_j) = N_{J_j J_i}(X_i) && \forall (i,j) \in E(T), \\ & X_j \succeq 0 && \forall J_j \in \mathcal{J}. \end{aligned}$$

which is exactly (CTC) as desired.

Algorithm 1 Positive semidefinite matrix completion

Input. Tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ with tree nodes $J_1, \ldots, J_\ell \subseteq \{1, \ldots, n\}$. Matrices $X_1, \ldots, X_\ell$ with each $X_j \succeq 0$ and of size $|J_j| \times |J_j|$.
Output. Sparse matrices R and $D \succeq 0$, which implicitly define a matrix $X = R^{-T} D R^{-1}$ satsifying $X[J_j, J_j] = X_j$.
Algorithm. Iterate over $j \in \{1, \ldots, \ell\}$ in any order. For each j with parent k on $\mathcal{T}$:

1. Define the unique set $U_j \equiv J_j \backslash J_k$ and the ancestor set $V_j \equiv J_j \backslash U_j$ (if j is a root node, then we set $U_j \equiv J_j$ and $V_j \equiv \varnothing$) and isolate the following three submatrices of $X_j \equiv X[J_j, J_j] \succeq 0$:

$$F_{1,1} = P_{J_j U_j}^T X_j P_{J_j U_j}, \qquad F_{2,1} = P_{J_j V_j}^T X_j P_{J_j U_j},$$
$$F_{2,2} = P_{J_j V_j}^T X_j P_{J_j V_j},$$

   where, given index sets $I, J \subseteq \{1, \ldots, n\}$, the $|I| \times |J|$ binary matrix P_{IJ} is defined with elements $P_{IJ}[i, j] = 1$ if $I(i) = J(j)$ and $P_{IJ}[i, j] = 0$ otherwise.
2. Solve a least-squares problem for $R[V_j, U_j]$

$$\text{minimize } \|R[V_j, U_j]\|_F^2 \text{ subject to } F_{2,1} + F_{2,2}\, R[V_j, U_j] = 0$$

   and set $R[U_j, U_j] = I_{|U_j|}$ and $D[U_j, U_j] = F_{1,1} - F_{2,1}^T R[V_j, U_j]$.

4.3 Recovery

Given a solution $X_1^\star, \ldots, X_\ell^\star$ to (CTC), we recover a corresponding solution $X^\star$ of (SDP) satisfying $X^\star[J_j, J_j] = X_j^\star$ by solving a positive semidefinite matrix completion problem in closed-form. Note that the matrix $X^\star$ is generally full-dense, so simply forming the matrix would push the overall complexity up to quadratic $\Theta(n^2)$ time and memory. Instead, we compute $X^\star$ implicitly in factorized form

$$X^\star = R^{-T} D R^{-1} \tag{29}$$

in which D is block-diagonal and possibly singular, and R is a psychologically[4] lower-triangular matrix with sparsity pattern E^*. In other words, we recover $X^\star$ by computing a data-sparse implicit representation of its psychologically upper-triangular Cholesky factorization.

Algorithm 1 is an adapted and simplified version of [37, Algorithm 7.3] and [25, Algorithm 10.1]. Its practical running time is comparable to the sparse Cholesky factorization algorithm, and its correctness is justified in [25, Section 10.2]. The following complexity figure is easily verified by inspection.

Proposition 7 (Postprocessing) *Given a solution $X_1^\star, \ldots, X_\ell^\star$ to (CTC), Algorithm 1 recovers a corresponding solution $X^\star$ of (SDP) in*

$$O(\omega^3 n) \text{ time and } O(\omega^2 n) \text{ memory.}$$

[4] The matrix R is said to be *psychologically* triangular if its rows can be reordered to leave the matrix triangular, i.e by sorting them according to the number of leading / trailing zeros. The etymology emphasizes the fact that solving $Ry = b$ for y is no more difficult than if R were actually triangular.

Proof Algorithm 1 performs some algorithm manipulations, and solves n linear systems of up to size $\omega \times \omega$ for $O(\omega^3 n)$ time and $O(\omega^2 n)$ memory. □

5 Optimal constraint splitting

The choice of matrices $\{C_j\}$ and $\{A_{i,j}\}$ in (CTC) is not unique, but has a significant impact on the sparsity of the reformulation, and hence the solution complexity. The problem of choosing the *sparsest* choice, i.e. the one with the fewest number of nonzero matrices, is a Set Cover problem, but can be efficiently solved in linear time using the tree decomposition $\mathcal{T}$. In this section, we describe an algorithm that minimizes the number of nonzero matrices $\{C_j\}$ and $\{A_{i,j}\}$ in linear time. Our algorithm is inspired by prior work on the tree-like Set Cover problem, but appears to be new within the context of clique tree conversion.

Let us state the exact problem to be solved. Given a sparse matrix M with sparsity pattern E, we wish to select a subset $\mathcal{S} \subseteq \mathcal{J}$ with the smallest number of elements $|\mathcal{S}|$ and define the "local" matrices $\{M_j\}$ to satisfy for all $X \in \mathbb{S}^n$:

$$M \bullet X = \sum_{J_k \in \mathcal{S}^\star} M_k \bullet X[J_k, J_k].$$

This can be posed as the following *Set Cover* problem

$$\mathcal{S}^\star = \underset{\mathcal{S} \subseteq \mathcal{J}^{(2)}}{\text{minimize}} \quad |\mathcal{S}| \quad \text{subject to} \quad \bigcup_{J_k^{(2)} \in \mathcal{S}} J_k^{(2)} \supseteq \mathcal{M}. \tag{30}$$

in which the set to be covered $\mathcal{M}$ are the indices for the nonzero elements in M,

$$\mathcal{M} = \{(i,j) \in \{1,\dots,n\}^2 : M[i,j] \neq 0\}$$

and the ℓ covering sets are constructed from the nodes $\mathcal{J} = \{J_1, \dots, J_\ell\}$ of our tree decomposition

$$\mathcal{J}^{(2)} = \{J_1^{(2)}, \dots, J_\ell^{(2)}\} \qquad \text{where } J_k^{(2)} = J_k \times J_k.$$

The general Set Cover problem is NP-Complete. However, (30) can be efficiently solved in polynomial time, due to the existence of the tree decomposition $\mathcal{T}$ associated with $\mathcal{J}^{(2)}$. More precisely, the covering sets inherit the *edge cover* and *running intersection* properties of $\mathcal{T}$:

$$\bigcup_{J_k^{(2)} \in \mathcal{J}^{(2)}} J_k^{(2)} \supseteq \mathcal{M} \qquad \text{for all possible choices of } \mathcal{M}, \tag{31}$$

$$J_i^{(2)} \cap J_j^{(2)} \subseteq J_k^{(2)} \qquad \text{for all } k \text{ on the path from } i \text{ to } j \text{ in } \mathcal{T}. \tag{32}$$

Set Cover endowed with (31)-(32) is known as *tree-like* Set Cover, and can be efficiently solved in polynomial time using a leaf-pruning algorithm [61]. The key insight is to note that every leaf node j with parent k in $\mathcal{T}$ contains a

Algorithm 2 Tree-like Set Cover for splitting constraints

Input. Rooted tree decomposition $\mathcal{T} = (\mathcal{J}, F)$ with tree nodes $\mathcal{J} = \{J_1, \ldots, J_\ell\}$. Sparse real symmetric matrix $M \subseteq \mathbb{S}^n$.
Output. Solution $\mathcal{S}^\star$ to Set Cover (30) and corresponding matrices $\{M_1, \ldots, M_\ell\}$ satisfying $M \bullet X = \sum_{J_k \in \mathcal{S}^\star} M_k \bullet X[J_k, J_k]$ for all $X \in \mathbb{S}^n$.
Algorithm.

1. (Precomputation) Define the inverse unique set map $u : \{1, \ldots, n\} \to \mathcal{J}$. Iterate over $J_j \in \mathcal{J}$ in any order. For each J_j with parent J_k, define $U_j \equiv J_j \backslash J_k$, and set $u(i) = J_j$ for all $i \in J_j \backslash J_k$. (If J_j has no parent, then define $U_j = J_j$.)
2. (Overestimation) Compute the overestimate $\mathcal{S}' \supseteq \mathcal{S}^\star$ using u:

$$\mathcal{S}' = \bigcup_{(i,j) \in \mathcal{M}} u(j), \qquad \mathcal{M} = \{(i,j) \in \{1, \ldots, n\}^2 : M[i,j] \neq 0\}.$$

3. (Leaf pruning on the overestimation) Iterate over $J_j \in \mathcal{S}'$ in topological order (children before parents). If $M[J_j, U_j] \neq \emptyset$ then add J_j to the set cover, define a new splitting, and set the submatrix to zero:

$$\mathcal{S} \leftarrow \mathcal{S} \cup J_j, \qquad M_j \leftarrow M[J_j, J_j], \qquad M[J_j, J_j] \leftarrow 0.$$

If $M = 0$, break and return $\mathcal{S} = \mathcal{S}^\star$and $\{M_k\}$.

unique set of elements $U_j^{(2)} \equiv J_j^{(2)} \backslash J_k^{(2)}$ due to (32). If $\mathcal{M} \cap U_j^{(2)} \neq \emptyset$, then $J_j^{(2)}$ must be included in the covering set; otherwise it can be safely ignored. Pruning the leaf node reveals new leaf nodes, and we repeat this process until $\mathcal{M}$ is covered.

Algorithm 2 is an adaptation of the leaf-pruning algorithm described above, with three important simplifications. First, it roots the tree (possibly arbitrarily) and uses a topological traversal to simulate the process of leaf pruning. Second, it notes that for every leaf node $J_j^{(2)}$ with parent $J_k^{(2)}$, the corresponding unique set $U_j^{(2)}$ can be written in terms of another unique set U_j:

$$U_j^{(2)} \equiv J_j^{(2)} \backslash J_k^{(2)} = (U_j \times J_j) \cup (J_j \times U_j) \qquad \text{where } U_j \equiv J_j \backslash J_k.$$

Third, it notes that the unique set $\{U_j\}$ generates a partitioning of $\{1, \ldots, n\}$, and as such, we may use the inverse map $u : \{1, \ldots, n\} \to \mathcal{J}$ satisfying

$$u(j) = \mathcal{J}_k \qquad \Longleftrightarrow \qquad j \in U_k.$$

to avoid explicitly pruning all $\ell = O(n)$ covering sets. This final simplification reduces the cost of processing a single M from $O(n)$ time to $O(\text{nnz}\,(M))$ time, after a precomputation step requiring $O(n)$ time and space.

In a practical implementation, dense indexing operations of the form $M[J, J]$ are expensive when M is stored using a sparse matrix storage format (typically, the compressed columns structure (CCS); see [62]). These operations can be avoided altogether in a multifrontal version of Algorithm 2, by incrementally assembling the elements of $M[J, J]$ while traversing the tree decomposition $\mathcal{T}$; see [37, Section 3] for details.

If we implement the preprocessing from (SDP) to (CTC) while using Algorithm 2 for the contraint splitting, then the complexity of the algorithm is linear time and space.

Proposition 8 (Preprocessing) *Given a tree decompositon $\mathcal{T} = (\mathcal{J}, F)$ for the sparsity graph G, the steps to transform (SDP) into (CTC) require*

$$O(\eta + \omega^2 n) \text{ time and memory.}$$

where $\omega \equiv 1 + \text{wid}(\mathcal{T})$ and $\eta = \text{nnz}\,(C) + \text{nnz}\,(A_1) + \cdots + \text{nnz}\,(A_m)$ is the total number of nonzeros in the data matrices.

Proof The first substep of converting (SDP) into (25) is free given a precomputed tree decomposition $\mathcal{T}$. The second substep of adding the overlap constraints (27) introduces $O(\omega^2 n)$ nonzeros, thereby requiring this much time and memory. Using Algorithm 2, the third substep of splitting (28) costs $O(n)$ time and space to set-up the unique sets $\{U_j\}$ and inverse map u, and $O(\text{nnz}\,(M))$ time and memory for each subsequent $M \in \{C, A_1, \ldots, A_m\}$. □

6 Block sparsity pattern of the overlap constraints

To solve the clique tree converted problem (CTC) using an interior-point method, we begin by putting the problem in standard canonical form (10) by vectoring the data:

$$\begin{aligned} \text{(P)} \quad & \text{minimize} \quad c^T x, \\ & \text{subject to} \quad \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} x = \begin{bmatrix} b \\ 0 \end{bmatrix}, \\ & \qquad\qquad x \in \mathcal{K}. \end{aligned} \qquad \begin{aligned} \text{(D)} \quad & \text{maximize} \quad \begin{bmatrix} b \\ 0 \end{bmatrix}^T y, \\ & \text{subject to} \quad \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T y + s = c, \\ & \qquad\qquad s \in \mathcal{K}_*. \end{aligned} \tag{33}$$

Here, the problem cone $\mathcal{K}$ is the Cartesian product of ℓ lower-dimensional semidefinite cones:

$$\mathcal{K} = \mathcal{K}_* = \mathbb{S}_+^{|J_1|} \times \mathbb{S}_+^{|J_2|} \times \cdots \times \mathbb{S}_+^{|J_\ell|}, \tag{34}$$

and the data c, $\mathbf{A}$, and $\mathbf{N}$ are

$$c = [\text{vec}\,(C_j)]_{j=1}^{\ell}, \quad \mathbf{A} = [\text{vec}\,(A_{i,j})^T]_{i,j=1}^{m,\ell}, \quad \mathbf{N} = [\mathbf{N}_{i,j}]_{i,j=1}^{\ell,\ell}, \tag{35}$$

where each i-th block-row of $\mathbf{N}$ implements the overlap constraint between the index set $J_i \in \mathcal{J}$ and its parent $J_j \in \mathcal{J}$ in the tree decomposition $(\mathcal{J}, T)$, as in

$$\sum_{j=1}^{\ell} \mathbf{N}_{i,j} \text{vec}\,(X_j) = \text{vec}\,\left(N_{J_i J_k}(X_k) - N_{J_k J_i}(X_i)\right) \qquad \text{where } k = p(i). \tag{36}$$

Note that by definition, we have $\mathbf{N}_{i,j} = 0$ for all j if $i = p(i)$ is a root node in T.

The vectorized problem (33) is a cone linear program in standard form of order $\nu(\mathcal{K}) = \sum_j |J_j| \leq \omega n$. An interior-point method is guaranteed to converge to an ϵ-accurate solution in $O(\sqrt{\omega n} \log \epsilon^{-1})$ iterations. The cost of each interior-point iteration is dominated by the cost of solving the $m \times m$ normal equations

$$\mathbf{H}u = \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} \nabla^2 F_{\mathcal{K}}(w) \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T u = r. \tag{37}$$

Here, $F_{\mathcal{K}}(w) = -\sum_j \log\det W_j$ for $w = [\mathrm{vec}\,(W_j)]_{j=1}^{\ell}$ is the usual log-determinant barrier for the Cartesian product of many semidefinite cones. The Hessian of $F_{\mathcal{K}}$ is block-diagonal:

$$\nabla^2 F_{\mathcal{K}}(w) = \mathrm{diag}(W_1^{-1} \otimes W_1^{-1}, W_2^{-1} \otimes W_2^{-1}, \ldots, W_\ell^{-1} \otimes W_\ell^{-1}), \tag{38}$$

so the $(1,1)$, $(2,1)$, and $(2,2)$ blocks of $\mathbf{H}$ must share their block-sparsity patterns with $\mathbf{A}\mathbf{A}^T$, $\mathbf{N}\mathbf{A}^T$, and $\mathbf{N}\mathbf{N}^T$.

The (1,1) block of $\mathbf{H}$ corresponds to the problem data $\mathbf{A}\mathbf{A}^T$. Its sparsity graph is a block generalization of the *primal intersection graph* previously defined in (15) for linear programs

$$G_p = (\{1, \ldots, m\}, E_p) \tag{39}$$

$(i,j) \in E_p$ if there exists $k \in \{1, \ldots, n\}$ such that $A_{i,k} \neq 0$ and $A_{j,k} \neq 0$.

This sparsity graph has been studied under the name of *correlative sparsity* [63]. In particular, it is known that many classes of SDPs convert into (CTC) for which G_p is the trivial empty graph [34].

The $(2,2)$ block of $\mathbf{H}$ corresponds to the overlap constraints $\mathbf{N}\mathbf{N}^T$. To study its sparsity graph, we make a crucial observation: the matrix $\mathbf{N}$ has the same block sparsity pattern as the incidence matrix for the tree T. More specifically, each nonzero i-th block-row of $\mathbf{N}$ contains exactly two nonzero sub-blocks: $\mathbf{N}_{i,i}$ and $\mathbf{N}_{i,j}$, where j is the parent of i on the tree $\mathcal{T}$. It immediately follows that the block sparsity patterns of $\mathbf{N}^T\mathbf{N}$ and $\mathbf{N}\mathbf{N}^T$ coincide with the adjacency matrices of T and its line graph $\mathcal{L}(T)$, respectively.

Lemma 4 *Define the block matrix $\mathbf{N} = [\mathbf{N}_{i,j}]$ with ℓ block-rows and ℓ block-columns to satisfy (36) with respect to the tree decomposition $\mathcal{T} = (\mathcal{J}, T)$. Then,*

- *The block sparsity pattern of $\mathbf{N}^T\mathbf{N}$ coincides with the adjacency matrix of T:*

$$i \neq j, \quad \sum_{k=1}^{\ell} \mathbf{N}_{k,i}\mathbf{N}_{k,j} \neq 0 \quad \Longleftrightarrow \quad (i,j) \in E(T).$$

- *The block sparsity pattern of $\mathbf{N}\mathbf{N}^T$ coincides with the adjacency matrix of the line graph $\mathcal{L}(T)$ of T:*

$$i \neq j, \quad \sum_{k=1}^{\ell} \mathbf{N}_{i,k}\mathbf{N}_{j,k} \neq 0 \quad \Longleftrightarrow \quad (i,p(i)) \cap (j,p(j)) \neq \varnothing$$

The line graph of a tree is not necessarily sparse. For example, if T were the star graph on n vertices, then its associated line graph $\mathcal{L}(T)$ would be the complete graph on $n-1$ vertices. In this case, matrix $\mathbf{N}\mathbf{N}^T$ is fully-dense, and the cost of an interior-point iteration is at least cubic $\Omega(n^3)$ time and quadratic $\Omega(n^2)$ memory. Indeed, we can give an explicit SDP whose tree decomposition is precisely a star graph.

Example 1 (Star graph) Given $b \in \mathbb{R}^n$, consider embedding the Euclidean norm problem $\|b\| = \max\{b^T y : \|y\| \leq 1\}$ into the size-$(n+1)$ SDP:

$$\begin{aligned} &\text{minimize} && \operatorname{tr}(X) \\ &\text{subject to} && X[i,(n+1)] = b_i \quad \forall i \in \{1,\dots,n\} \\ & && X \succeq 0 \end{aligned}$$

The associated sparsity graph is the *star graph* on $n+1$ nodes, and its tree decomposition is the star graph on n nodes:

$$\begin{gathered} J_1 = \{1, n+1\}, \quad J_2 = \{2, n+1\}, \quad \dots, \quad J_n = \{n, n+1\}, \\ T = (\{1,\dots,n\},\ \{(1,n),\ (2,n),\ \dots\ ,(n-1,n)\}). \end{gathered}$$

Applying clique tree conversion to Example 1 yields

$$\begin{aligned} &\text{minimize} && \operatorname{tr}(X_n) + \sum_{j=1}^{n-1} X_j[1,1] && && (40) \\ &\text{subject to} && X_i[2,1] = b_i && \forall i \in \{1,\dots,n\} \\ & && X_i[2,2] = X_n[2,2] && \forall i \in \{1,\dots,n-1\} \\ & && X_i \succeq 0 && \forall i \in \{1,\dots,n\}, \end{aligned}$$

which vectorizes into an instance of (33) with the following (where I is the identity matrix and $\mathbf{1}$ is the vector of ones)

$$\mathbf{A} = I_n \otimes \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \qquad \mathbf{N} = \begin{bmatrix} I_{(n-1)} & -\mathbf{1}_{(n-1)} \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}.$$

The problem data $\mathbf{A}$ is block-diagonal with $\mathbf{A}\mathbf{A}^T = I_n$, so its corresponding primal intersection graph G_p in (39) is the trivial empty graph. Nevertheless, the overlap constraints $\mathbf{N}$ produce a fully-dense $\mathbf{N}\mathbf{N}^T = I_{(n-1)} + \mathbf{1}\mathbf{1}^T$ block in $\mathbf{H}$. The normal matrix $\mathbf{H}$ has a diagonal $(1,1)$ block but a fully-dense $(2,2)$ block of size $(n-1)\times(n-1)$, and so requires at least $\Omega(n^3)$ time and $\Omega(n^2)$ memory to form and factor. When (40) is solved using an interior-point method, the cost of a single iteration is at least cubic time and quadratic memory.

7 Dualized Clique Tree Conversion

In the previous section, we studied the overlap constraint matrix $\mathbf{N}$, and found that the matrix $\mathbf{N}\mathbf{N}^T$ can be fully-dense, despite sparsity in the problem data. When this occurs, the normal matrix solved at each interation of an interior-point method is fully-dense, so the cost of solving (CTC) is at least cubic time and quadratic memory.

On the other hand, the matrix $\mathbf{N}^T\mathbf{N}$ is *guaranteed* to be sparse, with $\ell \leq n$ dense blocks each no bigger than $\frac{1}{2}\omega(\omega+1) \times \frac{1}{2}\omega(\omega+1)$, and arranged in a pattern corresponding to a tree (or forest). We can exploit this favorable sparsity structure using the dualization technique of Löfberg [27], which we had earlier reviewed in Section 3.6. Applying dualization to (CTC) yields the following pair:

$$\begin{aligned}
&\text{(P) minimize} && -\begin{bmatrix} b \\ 0 \end{bmatrix}^T x_1 && \text{(D) maximize} && -c^T y \qquad (41)\\
&\text{subject to} && \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T x_1 + x_2 = c, && \text{subject to} && s_0 = 0,\\
& && \|x_1\| \leq x_0, \quad x_2 \in \mathcal{K}_*. && && \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} y + s_1 = \begin{bmatrix} b \\ 0 \end{bmatrix},\\
& && && && -y + s_2 = 0,\\
& && && && \|s_1\| \leq s_0, \quad s_2 \in \mathcal{K}.
\end{aligned}$$

When (41) is solved using an interior-point method, the per-iteration costs are dominated by the solution of a set of normal equations governed by the following matrix

$$\mathbf{H} = \begin{bmatrix} 0 \\ \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \nabla^2 F^{\text{SOCP}}(w_1) \begin{bmatrix} 0 \\ \mathbf{A} \\ \mathbf{N} \end{bmatrix} + \nabla^2 F^{\mathcal{K}}(w_2) \qquad (42)$$

$$= \begin{bmatrix} 0 \\ \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \left(\begin{bmatrix} d_0 & 0 & 0 \\ 0 & D_1 & 0 \\ 0 & 0 & D_2 \end{bmatrix} + \begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix}^T \right) \begin{bmatrix} 0 \\ \mathbf{A} \\ \mathbf{N} \end{bmatrix} + \nabla^2 F^{\mathcal{K}}(w_2) \qquad (43)$$

$$= \left(\begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix} + \nabla^2 F^{\mathcal{K}}(w_2) \right) + \left(\begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \right) \left(\begin{bmatrix} \mathbf{A} \\ \mathbf{N} \end{bmatrix}^T \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \right)^T \qquad (44)$$

$$= \mathbf{H}_s + \mathbf{q}\mathbf{q}^T \qquad (45)$$

where $\nabla^2 F^{\text{SOCP}}$ is the Hessian of the self-concordant barrier for the second-order cone used to embed the free and fixed variables, and $\nabla^2 F^{\mathcal{K}}$ is the same block-diagonal Hessian in (38). Clearly, $\mathbf{H}$ is a low-rank perturbation of a

block-sparse matrix $\mathbf{H}_s$, whose block sparsity pattern coincides with the supergraph of the tree T associated with our tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ that we call the *dual intersection graph*:

$$G_d = (\{1, \ldots, \ell\}, E_d \cup E(T)) \tag{46}$$

$(i,j) \in E_d$ if there exists $k \in \{1, \ldots, m\}$ such that $A_{k,i} \neq 0$ and $A_{k,j} \neq 0$.

If G_d has bounded treewidth, then the cost of solving the normal equations $\mathbf{H}u = r$ is $O(\omega^4 n)$ time and memory. In this case, the per-iteration cost of the interior-point method is $O(\omega^4 n)$ time and memory.

Lemma 5 *Let Q be a tree decomposition for the dual intersection graph G_d defined in (46), and write $\tau \equiv 1 + \mathrm{wid}(Q)$. Then,*

1. *The data matrices $\mathbf{A}$ and $\mathbf{N}$ contain at most $O(\omega^4 \tau n)$ nonzero elements.*
2. *The cost of solving $\mathbf{H}u = r$ for u given $\mathbf{A}$, $\mathbf{N}$, w_1, w_2, r, and Q is bound*

$$O(\omega^6 \tau^2 n) \text{ time and } O(\omega^4 \tau n) \text{ memory.}$$

Proof By definition, G_d is the block sparsity graph for the sparse $\mathbf{H}_s$ part of $\mathbf{H}$ defined in (45), over square blocks no larger than $\omega(\omega+1)/2$. To prove the statement, we begin with three observations. First, both $\mathbf{H}$ and $\mathbf{H}_s$ contain up to $\omega^2 n$ columns and rows. Second, the treewidth of the sparsity graph of $\mathbf{H}_s$ is upper-bounded by $\tau\omega^2$, since its block sparsity graph over blocks of ω^2 has treewidth bound by τ. Third, its sparsity pattern coincides with that of $\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T$, where the tall-and-skinny matrix $\tilde{\mathbf{A}} \equiv [\mathbf{A}^T, \mathbf{N}^T]$ has more rows than it has columns.

First, we show that it costs $O(\omega^6\tau^2 n)$ time and $O(\omega^4 \tau n)$ memory to form the sparse matrix $\mathbf{H}_s$ in (45) given $\tilde{\mathbf{A}}$. This follows largely by repeating the proof of Lemma 3. First, we note that each column a_j of $\tilde{\mathbf{A}}$ must satisfy $\mathrm{nnz}\,(a_j) \leq \tau\omega^2$, as it contributes a clique of size $\mathrm{nnz}\,(a_j)$ to the sparsity graph of $\tilde{\mathbf{A}}\tilde{\mathbf{A}}^T$, but the latter does not contain cliques bigger than $\tau\omega^2$. Summing up over $\leq \omega^2 n$ columns in $\tilde{\mathbf{A}}$ yields $\mathrm{nnz}\,(\tilde{\mathbf{A}}) \leq \tau\omega^4 n$ memory. Adding up at most $\omega^2 n$ sets of dense $\tau\omega^2 \times \tau\omega^2$ blocks to form $\mathbf{H}_s$ costs at most $\omega^6\tau^2 n$ operations.

Next, we show that it costs $O(\omega^6\tau^2 n)$ time and $O(\omega^4\tau n)$ memory to solve $\mathbf{H}u = r$ given $\mathbf{H}$ and r, by solving $\mathbf{H}_s\tilde{u} = r$ and $\mathbf{H}_s v = \mathbf{q}$, and making the low-rank update $u = \mathbf{H}^{-1}r = \tilde{u} - v(I + v^T\mathbf{q})^{-1}v^T r$. The claimed complexity figure follows because we can factor $\mathbf{H}_s$ into its Cholesky factor in $O(\omega^6\tau^2 n)$ time and $O(\omega^4\tau n)$ memory by block-permuting the matrix with the elimination ordering of Q. Once factored, it costs $O(\omega^4\tau n)$ time and memory to solve for $\tilde{u}$, form $\mathbf{q}$, solve for v, and to compute the update term $v(I + v^T\mathbf{q})^{-1}v^T r$. □

The complete dualized clique tree conversion procedure is summarized in Algorithm 3. The associated complexity figures are summarized in the following statement.

Theorem 3 *Algorithm 3 computes an ϵ-accurate solution of (SDP) in*

$$O(\omega^{6.5}\tau^2 n^{1.5} \log \epsilon^{-1}) \text{ time and } O(\omega^4 \tau n) \text{ space,}$$

where $\omega \equiv 1 + \mathrm{wid}(\mathcal{T})$ and $\tau \equiv 1 + \mathrm{wid}(Q)$.

Algorithm 3 Dualized clique tree conversion

Input. Data vector $b \in \mathbb{R}^m$ and data matrices $C, A_1, \ldots, A_m \in \mathbb{S}^n_E$ with sparsity graph $G = (\{1, \ldots, n\}, E)$. Tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ for the sparsity graph G. Tree decomposition Q for the dual intersection graph $G_d \supseteq T$ defined in (46).
Output. An ϵ-accurate solution of (SDP) in factored form $\hat{X} = R^{-T} D R^{-1}$.
Algorithm.

1. (Conversion) Reformulate (SDP) into (CTC) by following the steps outlined in Section 4.
2. (Dualization) Vectorize (CTC) into (33) and dualize into (41).
3. (Solution) Solve (41) as a conic linear program in standard canonical form (10) using an interior-point method with $O(\sqrt{\nu})$ iteration complexity that solves each normal equation using sparse Cholesky factorization block-permuted by the perfect elimination ordering of Q (see Section 3.2).
4. (Recovery) Recover the dense solution $\hat{X}$ of the semidefinite program (SDP) in factored form $\hat{X} = R^{-T} D R^{-1}$ using Algorithm 1 in Section 4.3.

Proof Steps 1 and 2 are algebraic manipulations, and require $\operatorname{nnz}(\mathbf{A}) + \operatorname{nnz}(\mathbf{N}) = O(\omega^4 \tau n)$ time and memory to perform due to Proposition 8. Step 3 performs $O(\sqrt{\omega n} \log \epsilon^{-1})$ interior-point iterations, costing $O(\omega^6 \tau^2 n)$ time and $O(\omega^4 \tau n)$ memory per-iteration due to Lemma 5. Step 5 requires $O(\omega^3 n)$ time and $O(\omega^2 n)$ memory due to Proposition 7. □

Under the assumption that both the sparsity graph G and the dual intersection graph G_d have bounded treewidth, we are guaranteed to solve (SDP) in

$$O(n^{1.5} \log \epsilon^{-1}) \text{ time and } O(n) \text{ memory,} \tag{47}$$

by pairing Theorem 3 with the linear-time tree decomposition algorithm of Bodlaender [19]. In practice, it is far more efficient to use a sparse linear algebra heuristic like minimum degree or nested dissection to compute tree decompositions with small but suboptimal widths. The resulting algorithm will frequently attain the optimal complexity figure (47) on nondegenerate problems.

One special case where the dual intersection graph G_d is guaranteed to have bounded treewidth is if $\mathbf{A}^T \mathbf{A}$ is block-diagonal. Indeed, if a block-diagonal choice of $\mathbf{A}^T \mathbf{A}$ exists, then our optimal constraint splitting algorithm (Algorithm 2) is guaranteed to find it. In this case, the block sparsity pattern of $\mathbf{H}_s$ coincides with that of $\mathbf{N}^T \mathbf{N}$, and G_d coincides with the tree T associated with our tree decomposition $\mathcal{T} = (\mathcal{J}, T)$. As such, $\mathbf{H}_s$ has a block chordal sparsity pattern, and can be factored without block fill after any topological ordering, such as minimum degree. Each interior-point iteration is guaranteed to cost $O(n)$ time and memory. Substituting these insights into Theorem 3 yields our first main result.

Proof (Theorem 1) It is easy to verify that the optimal solution $\mathcal{S}^\star$ to the Set Cover problem (30) has just a single element $|\mathcal{S}^\star| = 1$ if and only if the given matrix M is *decoupled* (in the sense of Definition 3). Hence, if all $A_1, \ldots, A_m$ are decoupled, then by virtue of the optimality of Algorithm 2, the resulting $\mathbf{A}^T \mathbf{A}$ will always be block-diagonal. The dual intersection graph G_d is always

the tree T associated with the tree decomposition $\mathcal{T} = (\mathcal{J}, T)$. The optimal tree decomposition Q for a tree T has $\text{wid}(Q) = 1$, and is trivially obtained in linear time by reordering the vertices of T by a minimum degree ordering [64] and computing the associated elimination tree (see Section 3.3). Substituting $\tau = 2$ into Theorem 3 yields the desired complexity estimate. □

8 Dualized Clique Tree Conversion with Auxillary Variables

The previous section described a dualized version of the clique tree conversion algorithm, with complexity determined by the treewidth of the dual intersection graph G_d defined in (46). In particular, if we assume that G_d has bounded treewidth, then the complexity of computing an ϵ-accurate solution to (CTC) is $O(\omega^{6.5} n^{1.5} \log \epsilon^{-1})$ time and $O(\omega^4 n)$ memory, where $\omega = 1 + \text{wid}(\mathcal{T})$ and $\mathcal{T}$ is the tree decomposition used to convert (SDP) into (CTC).

There are two issues with this result in practice. First, each choice of G_d is tied to a specific instance of (CTC), but the conversion from (SDP) to (CTC) is nonunique. Just because a choice of G_d with bounded treewidth exists does not mean that it will be found. Second, the treewidth of G_d cannot be easily determined from properties of (SDP). Its value is unrelated to ω: any dense instance of (CTC) with $A_{i,j} \neq 0$ for all (i, j) must have $\text{tw}(G_d) = n - 1$, irrespective of ω.

In this section, we address both issues for a large class of sparse SDPs called network flow SDPs. For these problems, an optimal embedding of (SDP) into (CTC) can be determined analytically, by appealing to properties of the tree decomposition. The resulting dual intersection graph G_d is not guaranteed to have bounded treewidth, but can be systematically forced have a treewidth of 1 by introducing auxillary variables. Bounding the number of auxillary variables then allows us to guarantee near-linear time and linear memory complexity for these problems.

8.1 An illustrative example

Let us begin by motivating and elaborating on the idea of introducing auxillary variables with an illustrative example.

Example 2 (Path graph) Given two $(n+1) \times (n+1)$ symmetric tridiagonal matrices

$$A = \begin{bmatrix} A_{11} & A_{21} & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ 0 & A_{23} & A_{33} & \ddots \\ 0 & 0 & \ddots & \ddots \end{bmatrix}, \qquad C = \begin{bmatrix} C_{11} & C_{21} & 0 & 0 \\ C_{21} & C_{22} & C_{23} & 0 \\ 0 & C_{23} & C_{33} & \ddots \\ 0 & 0 & \ddots & \ddots \end{bmatrix},$$

with A positive definite, consider the Rayleigh quotient problem

$$\text{minimize } C \bullet X \qquad \text{subject to} \qquad A \bullet X = 1, \quad X \succeq 0. \tag{48}$$

The associated sparsity graph is the *path graph* on $n+1$ nodes, and its tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ is the path graph on n nodes:

$$\begin{aligned} &J_1 = \{1,2\}, \quad J_2 = \{2,3\}, \quad \ldots, \quad J_n = \{n, n+1\}, \\ &T = (\{1,\ldots,n\}, \quad \{(1,2),\ (2,3),\ \ldots\ ,(n-1,n)\}). \end{aligned}$$

Applying clique-tree conversion to Example 2 yields

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} C_j \bullet X_j \\ \text{subject to} \quad & \sum_{j=1}^{n} A_j \bullet X_j = 1 \\ & X_j[2,2] = X_{j+1}[1,1] \qquad \forall j \in \{1,\ldots,n\} \\ & X_j \succeq 0 \qquad\qquad\qquad\ \ \forall j \in \{1,\ldots,n\} \end{aligned} \tag{49}$$

Here, we have split A into

$$A_1 = \begin{bmatrix} A_{1,1} & A_{2,1} \\ A_{2,1} & A_{2,2} \end{bmatrix}, \quad A_j = \begin{bmatrix} 0 & A_{(j+1),j} \\ A_{(j+1),j} & A_{(j+1),(j+1)} \end{bmatrix} \quad j \in \{2,\ldots,n\}$$

and similarly split C into $C_1, C_2, \ldots, C_n$. The dual intersection graph G_d for (49) is the complete graph on n nodes, because A_i and A_j appear in the same constraint $\sum_j A_j \bullet X_j = 1$ for every (i,j). Accordingly, Theorem 3 predicts a solution complexity of $O(n^{3.5}\log\epsilon^{-1})$ time and $O(n^2)$ memory. Indeed, this figure is attained (within a factor of $\Theta(\sqrt{n})$) because the associated normal equations are fully-dense.

Instead, the problematic constraint $\sum_j A_j \bullet X_j = 1$ can be enforced as n constraints by introducing $n-1$ auxillary variables $u_1, \ldots, u_{n-1}$, as in

$$\begin{aligned} A_1 \bullet X_1 \qquad -u_1 \qquad\qquad\qquad\quad &= b, \\ u_1 + A_2 \bullet X_2 \ -u_2 \qquad\quad &= 0, \\ \ddots \qquad\quad \ddots \qquad\quad &\vdots \\ u_{n-1} + A_n \bullet X_n &= 0. \end{aligned} \tag{50}$$

Grouping each X_j together with u_{j-1} and vectoring yields the following:

$$\begin{aligned} \text{(P)} \quad \text{minimize} \quad & \tilde{c}^T \xi, \\ \text{subject to} \quad & \begin{bmatrix} \tilde{\mathbf{A}} \\ \tilde{\mathbf{N}} \end{bmatrix} \xi = \begin{bmatrix} \tilde{b} \\ 0 \end{bmatrix}, \\ & \xi \in \mathcal{K}. \end{aligned} \qquad \begin{aligned} \text{(D)} \quad \text{maximize} \quad & \begin{bmatrix} \tilde{b} \\ 0 \end{bmatrix}^T y, \\ \text{subject to} \quad & \begin{bmatrix} \tilde{\mathbf{A}} \\ \tilde{\mathbf{N}} \end{bmatrix}^T y + s = \tilde{c}, \\ & s \in \mathcal{K}_*. \end{aligned} \tag{51}$$

The new problem cone $\mathcal{K}$ is the Cartesian product of n lower-dimensional semidefinite cones and $\mathbb{R}^{n-1}$ in the following order:

$$\mathcal{K} = \mathbb{S}_+^{|J_1|} \times (\mathbb{S}_+^{|J_2|} \times \mathbb{R}) \times \cdots \times (\mathbb{S}_+^{|J_n|} \times \mathbb{R}). \tag{52}$$

Each primal decision variable is now padded with an auxillary variable,

$$\xi = [\xi_j]_{j=1}^n, \qquad \xi_1 = \operatorname{vec} X_1, \qquad \xi_j = \begin{bmatrix} \operatorname{vec} X_j \\ u_{j-1} \end{bmatrix} \quad j \in \{2, \dots, n\},$$

and the the corresponding data $\tilde{c}$ and $\tilde{\mathbf{N}}$ are padded with zeros in a similar way. The new choice of $\tilde{\mathbf{A}} = [\tilde{\mathbf{A}}_{i,j}]_{i,j=1}^{n,n}$ reads:

$$\tilde{\mathbf{A}}_{1,1} = (\operatorname{vec} A_1)^T, \qquad \begin{bmatrix} \tilde{\mathbf{A}}_{j,(j-1)} \\ \tilde{\mathbf{A}}_{j,j} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ (\operatorname{vec} A_j)^T & +1 \end{bmatrix} \quad j \in \{2, \dots, n\},$$

and we see that the block-sparsity pattern of $\tilde{\mathbf{A}}$ now matches the incidence matrix of T. The normal matrix $\mathbf{H}$ for the dualized version of (51) has block sparsity graph $\tilde{G}_d = T$, because we have $\tilde{\mathbf{A}}_{k,i} \neq 0$ and $\tilde{\mathbf{A}}_{k,j} \neq 0$ only if $(i,j) \in E(T)$. Repeating the proof of Theorem 3, we find that the dualized version of (51) can be solved to ϵ-accuracy in $O(n^{1.5} \log \epsilon^{-1})$ time and $O(n)$ memory.

8.2 Introducing auxillary variables

Let us formalize and generalize the auxillary variable procedure from above. For an arbitrary constraint $A \bullet X = b$ in (SDP), we assume without loss of generality[5] that the corresponding constraint in (CTC) is split over a *connected subtree* of T induced by a subset of vertices $W \subseteq V(T)$, as in

$$\sum_{j \in W} A_j \bullet X_j = b, \qquad T_W \equiv (W,\ E(T)) \text{ is connected.} \tag{53}$$

Then, the coupled constraint (53) can be decoupled into $|W|$ constraints, by introducing $|W| - 1$ auxillary variables, one for each edge of the connected subtree T_W:

$$\forall i \in C: \qquad A_i \bullet X_i + \sum_{j \in \mathrm{ch}(i)} u_j = \begin{cases} b & i \text{ is root of } T_W, \\ u_i & \text{otherwise.} \end{cases} \tag{54}$$

For each $i \in C$, we identify the tree node J_i in T_W with the variable group $\{X_i, u_{\mathrm{ch}(i)}\}$ where $u_{\mathrm{ch}(i)} = \{u_j : j \in \mathrm{ch}(i)\}$. It is easy to see that (53) and (54) are equivalent.

Lemma 6 *We have (53) if and only if there exists $\{u_j\}$ satisfying (54).*

[5] We may assume that T is connected without loss of generality, because we can arbitrarily add an edge between two disconnected components without affecting the tree decomposition. Since T is connected, we can always find a connected subset U satisfying $W \subseteq U \subseteq V(T)$ and replace W by U.

Proof Vectorizing the relation (54) yields $\mathbf{A}x + \mathbf{B}u = \mathbf{c}$, where $\mathbf{A} = [\mathbf{A}_{i,j}]$ and $\mathbf{A}_{i,j} = (\operatorname{vec} A_j)^T$ for $j = C(i)$ and $\mathbf{A}_{i,j} = 0$ otherwise, and $\mathbf{c}_i = b$ for $i = 1$ and $\mathbf{c}_i = 0$ otherwise. The matrix $\mathbf{B}$ is the directed incidence matrix of T_C (relabeled using the ordering in C),

$$\mathbf{B}_{i,j} = \begin{cases} -1 & C(i) = C(j) \text{ and } C(j) \text{ is not root,} \\ +1 & C(i) = p(C(j)), \\ 0 & \text{otherwise,} \end{cases}$$

so its kernel has dimension one (i.e. number of connected components in T_C), and is spanned by the vector-of-ones $\mathbf{1}$. The result follows because there exists u such that $\mathbf{B}u = \mathbf{c} - \mathbf{A}x$ if and only if $\mathbf{1}^T(\mathbf{c} - \mathbf{A}x) = b - \sum_{j \in C} A_j \bullet X_j = 0$. □

Repeating the splitting procedure for every constraint in (CTC) yields Algorithm 4. By performing the decoupling procedure in Step 2, the block sparsity graph of the interior-point Hessian matrix $\mathbf{H}$ is guaranteed to match the tree T associated with our tree decomposition $\mathcal{T} = (\mathcal{J}, T)$, but at the cost of enlargening the individual blocks. Balancing these two considerations yields the following complexity estimate.

Theorem 4 *Algorithm 4 computes an ϵ-accurate solution to (SDP) in*

$$O((\omega^2 + \gamma_{\max})^3 \omega^{0.5} n^{1.5} \log \epsilon^{-1}) \text{ time and } O((\omega^2 + \gamma_{\max})^2 n) \text{ memory,}$$

where $\omega = 1 + \operatorname{wid}(\mathcal{T})$ and $\gamma_{\max} = \max_j \gamma_j$ is the maximum number of auxillary variables added to a single variable block.

Proof We repeat the proof of Theorem 3, but slightly modify the cost of solving (CTC). After the decoupling step, the sparse portion $\mathbf{H}_s$ of the normal matrix $\mathbf{H}$ is guaranteed to have a block sparsity pattern that coincides with the tree T associated with our tree decomposition $\mathcal{T} = (\mathcal{J}, T)$. However, the blocks are now sized $\frac{1}{2}\omega(\omega+1) + \gamma_{\max}$. Hence, each interior-point iteration now costs $O((\omega^2 + \gamma_{\max})^3 n)$ time and $O((\omega^2 + \gamma_{\max})^2 n)$ memory. After $O(\sqrt{\omega n} \log \epsilon^{-1})$ interior-point iterations, we arrive at an ϵ-accurate solution to (CTC). □

8.3 Application to network flow SDPs

Given a graph $G = (V, E)$ on n vertices $V = \{1, \ldots, n\}$, let $\mathcal{T} = (\mathcal{J}, T)$ with nodes $\mathcal{J} = \{J_1, \ldots, J_\ell\}$ be a tree decomposition for the graph G. Define $\mathcal{S}_k$ as the set of tree nodes that contain the element k:

$$\mathcal{S}_k \equiv \{j \in \{1, \ldots, \ell\} : k \in J_j\}.$$

It is a classic result of tree decompositions that $\mathcal{S}_k$ induces a *connected subtree* $T_k = (\mathcal{S}_k, E(T))$ on T. Indeed, any two $u, v \in \mathcal{S}_k$ are connected on T_k, because $k \in J_u$ and $k \in J_v$ (by definition of $\mathcal{S}_k$) implies $k \in J_w$ for any w that lies on the path from u to v (by the running intersection property), and hence $w \in \mathcal{S}_k$.

Algorithm 4 Dualized clique tree conversion with auxillary variables

Input. Data vector $b \in \mathbb{R}^m$ and data matrices $C, A_1, \ldots, A_m \in \mathbb{S}^n_E$ with sparsity graph $G = (\{1, \ldots, n\}, E)$. Tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ with $\mathcal{J} = \{J_1, \ldots, J_\ell\}$ for the sparsity graph G.
Output. An ϵ-accurate solution of (SDP) in factored form $\hat{X} = R^{-T} D R^{-1}$. The list $\gamma \in \mathbb{R}^\ell$ for the number of auxillary variables added to each block.
Algorithm.

1. (Conversion) Reformulate (SDP) into (CTC) by following the steps outlined in Section 4.
2. (Decoupling) Define ℓ variable groups $\xi_1, \ldots, \xi_\ell$ as in $\xi_j = \operatorname{vec} X_j$ and set $\gamma = 0$. Iterate over $i \in \{1, \ldots, m\}$ and decouple the i-th constraint $\sum_j A_{i,j} \bullet X_j = b_i$ in (CTC) as follows:
   (a) Compute the smallest $W \subseteq \{1, \ldots, \ell\}$ so that
   $$\sum_{j=1}^{\ell} A_{i,j} \bullet X_j = \sum_{j \in W} A_{i,j} \bullet X_j \qquad T_W \equiv (W,\ E(T)) \text{ is connected.}$$
   (b) If $|W| \leq 2$, then the constraint is already decoupled; go to next value of i. Otherwise, proceed below.
   (c) Convert the coupled constraint $\sum_{j \in W} A_{i,j} \bullet X_j = b_i$ into $|W|$ decoupled constraints
   $$\forall j \in W: \qquad A_{i,j} \bullet X_j + \sum_{k \in \mathrm{ch}(j)} u_{i,k} = \begin{cases} b_i & j \text{ is root of } T_W, \\ u_{i,j} & \text{otherwise.} \end{cases}$$
   (d) For each $j \in W$, add $u_{i,\mathrm{ch}(j)}$ to the j-th variable group and increment γ_j by the number of auxillary variables added:
   $$\xi_j \leftarrow \xi_j \cup \{u_{i,\mathrm{ch}(j)}\}, \qquad \gamma_j \leftarrow \gamma_j + |\mathrm{ch}(j)|.$$
3. (Dualization) Vectorize the resulting reformulation and dualize.
4. (Solution) Solve the dualized problem as a conic linear program in standard canonical form (10) using an interior-point method with $O(\sqrt{\nu})$ iteration complexity that solves each normal equation using sparse Cholesky factorization block-permuted by a minimum degree elimination ordering.
5. (Recovery) Recover the dense solution $\hat{X}$ of the semidefinite program (SDP) in factored form $\hat{X} = R^{-T} D R^{-1}$ using Algorithm 1 in Section 4.3.

Now, recall from Definition 4 that the i-th constraint matrix A_i in (SDP) is said to be a network flow constraint at vertex k if it can be written

$$A_i = \alpha_{i,k} e_k e_k^T + \sum_{(j,k) \in E} \alpha_{i,j} (e_k e_j^T + e_j e_k^T)$$

where e_i is the i-th column of the size-n identity matrix. It turns out that such a constraint can always be split over the k-th set $\mathcal{S}_k$.

Lemma 7 *Let A_i be a network flow constraint at vertex k. Then, there exists a choice of $\{A_{i,j}\}$ such that:*

$$A_i \bullet X = \sum_{j \in \mathcal{S}_k} A_{i,j} \bullet X[J_j, J_j] \qquad \forall X \in \mathbb{S}^n.$$

Proof Let us prove this by giving an explicit algorithm for its construction. We start off by initializing $A_{i,j} \leftarrow 0$ for all j. By the vertex cover property

of the tree decomposition, the vertex k is contained within at least one node $J_w \in \mathcal{J}$, so we set

$$A_{i,j} \leftarrow A_{i,j} + \alpha_{i,k} e_u e_u^T \qquad \text{where } k = J_w(u).$$

Note that since $k \in J_w$, we have $w \in \mathcal{S}_k$. Similarly, by the edge cover property of the tree decomposition, every edge $(j,k) \in E$ is contained within at least one node $J_{w'} \in \mathcal{J}$, so we set

$$A_{i,w'} \leftarrow A_{i,w'} + \alpha_{i,j}(e_u e_v^T + e_v e_u^T) \qquad \text{where } k = J_{w'}(u) \text{ and } j = J_{w'}(v)$$

for each neighbor j of k on G. Again, since $k \in J_{w'}$, we must have $w' \in \mathcal{S}_k$. This way, we have covered all nonzero elements in A_i while only setting $A_{i,j} \neq 0$ if $j \in \mathcal{S}_k$. □

Given a network flow SDP on G, if we force the conversion from (SDP) to (CTC) using the splitting described in Lemma 7, then it is possible to bound the number of auxillary variables needed to perform the decoupling in Step 2 of Algorithm 4. This results in a proof of Theorem 2.

Proof (Theorem 2) The key is to show that under the conditions stated in the theorem, the maximum number of auxillary variables added to each variable block is bound $\gamma_j \leq m_k \cdot \omega \cdot d_{\max}$. We do this via the following line of reasoning:

- A single network flow constraint at vertex k contributes $|\mathrm{ch}(j)| \leq d_{\max}$ auxillary variables to every j-th clique J_j satisfying $j \in \mathcal{S}_k$.
- Having one network flow constraint at every $k \in \{1, \ldots, \ell\}$ contributes at most $\omega \cdot d_{\max}$ auxillary variables to every j-th clique J_j. This is because the set of $\mathcal{S}_k$ for which $j \in \mathcal{S}_k$ is exactly $J_j = \{\{1, \ldots, \ell\} : j \in \mathcal{S}_k\}$, and $|J_j| \leq \omega$ by definition.
- Having m_k constraints coupled at each $k \in \{1, \ldots, \ell\}$ contributes at most $m_k \cdot \omega \cdot d_{\max}$ auxillary variables to every j-th clique J_j.

Finally, applying $\gamma_j \leq m_k \cdot \omega \cdot d_{\max}$ to Theorem 4 yields the desired complexity figures. □

9 Numerical Experiments

Using the techniques described in this paper, we solve sparse SDPs posed on the 40 power system test cases in the MATPOWER suite [65]. The largest two cases have $n = 9241$ and $n = 13659$, and are designed to accurately represent the size and complexity of the European high voltage electricity transmission network [66]. The associated SDPs are so large that simply forming the $n \times n$ matrix decision variable X would deplete all memory. Fortunately, power systems are graphs with bounded treewidths [16], and so the ideas of clique tree conversion are applicable.

```
E = C | A1 | A2 | A3 | A4;
p = amd(E); % fill-reducing ordering
[~,~,parT,~,R] = symbfact(E(p,p), 'sym', 'lower');
R(p,:) = R; % Reverse the ordering
for i = 1:n, J{i} = find(R(:,i)); end
```

Fig. 1: MATLAB code for computing the tree decomposition of a given sparsity graph. The code terminates with tree decomposition $\mathcal{T} = (\mathcal{J}, T)$ in which the index sets $\mathcal{J} = \{J_1, \ldots, J_n\}$ are stored as the cell array `J`, and the tree T is stored in terms of its parent pointer `parT`.

In all of our trials below, the accuracy of a primal-dual iterate (X, y, S) is measured using the DIMACS feasibility and duality gap metrics [67] and stated as the number of accurate decimal digits:

$$\begin{aligned}
\text{pinf} &= -\log_{10}\left[\ \|\mathcal{A}(X) - b\|_2/(1 + \|b\|_2)\ \right], \\
\text{dinf} &= -\log_{10}\left[\ \lambda_{\max}(\mathcal{A}^T(y) - C)/(1 + \|C\|_2)\ \right], \\
\text{gap} &= -\log_{10}\left[\ (C \bullet X - b^T y)/(1 + |C \bullet X| + |b^T y|)\ \right],
\end{aligned}$$

where $\mathcal{A}(X) = [A_i \bullet X]_{i=1}^m$ and $\mathcal{A}^T(y) = \sum_{i=1}^m y_i A_i$. We will frequently measure the overall number of accurate digits as $L = \min\{\text{gap}, \text{pinf}, \text{dinf}\}$. The experiments are performed on a Xeon 3.3 GHz quad-core CPU with 16 GB of RAM. The interior-point solvers used in our trials are SeDuMi v1.32 [56] and MOSEK v8.0.0.53 [21]. Both solvers have a guarantee iteration bound of $k = O(\sqrt{\theta} \log \epsilon^{-1})$ for an order-θ convex cone [20].

9.1 Tree decompositions via supernodal elimination trees

All of our trials use MATLAB's internal approximate minimum degree heuristic (due to Amestoy, Davis and Duff [68]) to compute tree decompositions with low width. Given a problem sparsity pattern, we compute a fill-reducing ordering using the `amd` command, and then compute an elimination tree using the `symbfact` command. A simplified version of our code is shown as the snippet in Figure 1. (Our actual code uses Algorithm 4.1 in [25] to reduce the computed elimination tree to the *supernodal* elimination tree, for a slight reduction in the number of index sets $\ell = |\mathcal{J}|$.)

Table 1 gives the details and timings for the 40 power system graphs from the MATPOWER suite [65]. As shown, the approximate minimum degree heuristic is able to compute tree decompositions with $\text{wid}(\mathcal{T}) \leq 34$ in less than 2 seconds. It is worth noting that all of the SDPs considered in this section have sparsity graphs that either exactly coincide with the network graph, or nearly coincide with it. Hence, the timings shown in Table 1 match actual in-situ timings, up to a small constant. In practice, the bottleneck of the preprocessing step is not the tree decomposition, but the constraint splitting step in Algorithm 2.

Table 1: Tree decompositions for the 40 test power systems under study: $|V(G)|$ - number of vertices; $|E(G)|$ - number of edges; $\omega = 1 + \text{wid}(\mathcal{T})$ - computed clique number; "Time" - total computation time in seconds.

#	Name	$\|V(G)\|$	$\|E(G)\|$	ω	Time	#	Name	$\|V(G)\|$	$\|E(G)\|$	ω	Time
1	case4gs	4	4	3	0.171	21	case1354pegase	1354	1991	13	0.155
2	case5	5	6	3	0.030	22	case1888rte	1888	2531	13	0.213
3	case6ww	6	11	4	0.014	23	case1951rte	1951	2596	14	0.219
4	case9	9	9	3	0.027	24	case2383wp	2383	2896	25	0.278
5	case9Q	9	9	3	0.011	25	case2736sp	2736	3504	25	0.310
6	case9target	9	9	3	0.002	26	case2737sop	2737	3506	24	0.317
7	case14	14	20	3	0.006	27	case2746wop	2746	3514	26	0.314
8	case24_ieee_rts	24	38	5	0.017	28	case2746wp	2746	3514	24	0.312
9	case30	30	41	4	0.005	29	case2848rte	2848	3776	18	0.334
10	case30Q	30	41	4	0.004	30	case2868rte	2868	3808	17	0.323
11	case30pwl	30	41	4	0.004	31	case2869pegase	2869	4582	15	0.317
12	case_ieee30	30	41	4	0.004	32	case3012wp	3012	3572	28	0.344
13	case33bw	33	37	2	0.005	33	case3120sp	3120	3693	27	0.353
14	case39	39	46	4	0.005	34	case3375wp	3374	4161	30	0.378
15	case57	57	80	6	0.010	35	case6468rte	6468	9000	30	0.725
16	case89pegase	89	210	12	0.011	36	case6470rte	6470	9005	30	0.716
17	case145	145	453	5	0.018	37	case6495rte	6495	9019	31	0.713
18	case118	118	186	11	0.020	38	case6515rte	6515	9037	31	0.716
19	case_illinois200	200	245	9	0.024	39	case9241pegase	9241	16049	35	1.009
20	case300	300	411	7	0.035	40	case13659pegase	13659	20467	35	1.520

9.2 MAX 3-CUT and Lovasz Theta

We begin by considering the MAX 3-CUT and Lovasz Theta problems, which are decoupled by default, and hence have solution complexities of $O(n^{1.5})$ time and $O(n)$ memory. For each of the 40 test cases, we use the MATPOWER function `makeYbus` to generate the bus admittance matrix $Y_{bus} = [Y_{i,j}]_{i,j=1}^{n}$, and symmetricize to yield $Y_{abs} = \frac{1}{2}[|Y_{i,j}| + |Y_{j,i}|]_{i,j=1}^{n}$. We view this matrix as the weighted adjacency matrix for the system graph. For MAX 3-CUT, we define the weighted Laplacian matrix $C = \text{diag}(Y_{abs}\mathbf{1}) - Y_{abs}$, and set up problem (MkC). For Lovasz Theta, we extract the location of the graph edges from Y_{abs} and set up (LT′).

First, we use Algorithm 3 alongside SeDuMi to solve the 80 SDPs. Of the 80 SDPs considered, 79 solved to $L \geq 5$ digits in $k \leq 23$ iterations and $T \leq 306$ seconds; the largest instance solved to $L = 4.48$. Table 2 shows the accuracy and timing details for the 20 largest problems solved. Figure 2a plots T/k, the mean time taken per-iteration. Unsurprisingly, the per-iteration time is linear with respect to n. A log-log regression yields $T/k = 10^{-3}n$, with $R^2 = 0.9636$. Figure 2b plots k/L, the number of iterations to a factor-of-ten error reduction. We see that SeDuMi's guaranteed iteration complexity $k = O(\sqrt{n}\log\epsilon^{-1}) = O(\sqrt{n}L)$ is a significant over-estimate; a log-log regression yields $k/L = 0.929n^{0.123} \approx n^{1/8}$, with $R^2 = 0.5432$. Combined, the data suggests an actual time complexity of $T \approx 10^{-3}n^{1.1}L$.

Next, we use Algorithm 3 alongside MOSEK to solve the 80 SDPs. It turns out that MOSEK is both more accurate than SeDuMi, as well as a factor of 5-10 faster. It manages to solve all 80 SDPs to $L \geq 6$ digits in $k \leq 21$ iterations and $T \leq 24$ seconds. Table 3 shows the accuracy and timing details for the 20 largest problems solved. Figure 3a plots T/k, the mean time taken per-

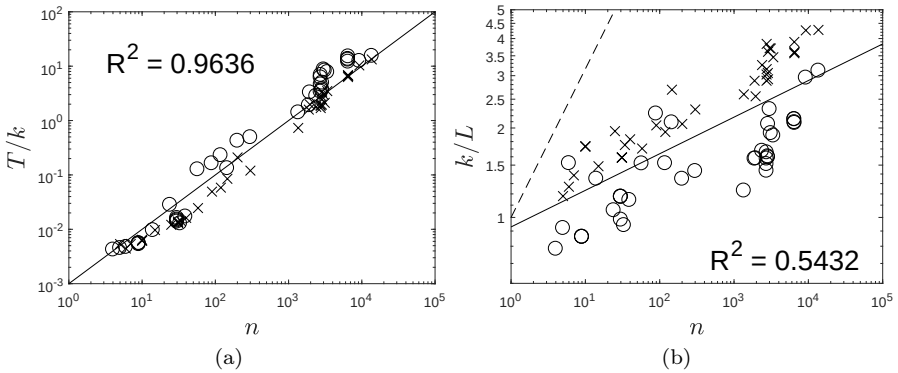


Fig. 2: SeDuMi Timings for MAX 3-CUT (∘) and Lovasz Theta (×) problems: (a) Time per iteration, with regression $T/k = 10^{-3}n$; (b) Iterations per decimal digit of accuracy, with (solid) regression $k/L = 0.929n^{0.123}$ and (dashed) bound $k/L = \sqrt{n}$.

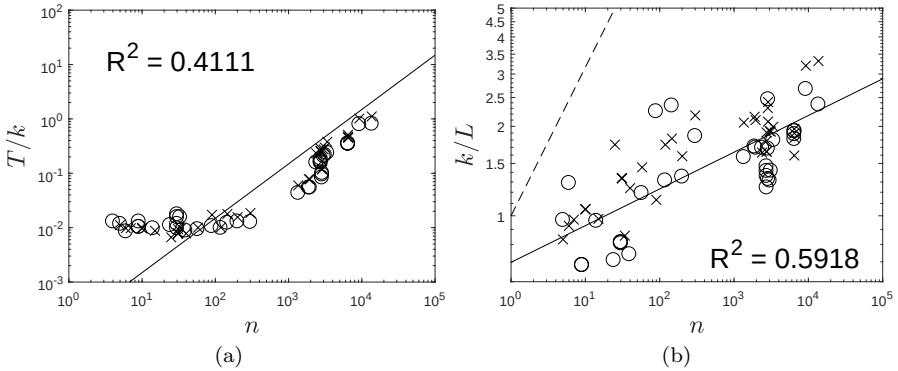


Fig. 3: MOSEK Timings for MAX 3-CUT (∘) and Lovasz Theta (×) problems: (a) Time per iteration, with regression $T/k = 1.488 \times 10^{-4}n$; (b) Iterations per decimal digit of accuracy, with (solid) regression $k/L = 0.697n^{0.123}$ and (dashed) bound $k/L = \sqrt{n}$.

iteration. For very small values of n, the trend flattens out at around 0.01 seconds due to the communication overhead between MATLAB and the MOSEK optimizer. Figure 3b plots k/L, the number of iterations to a factor-of-ten error reduction. Again, we see that MOSEK's guaranteed iteration complexity $k = O(\sqrt{n}\log\epsilon^{-1}) = O(\sqrt{n}L)$ is a significant over-estimate. A log-log regression yields an empirical time complexity of $T \approx 10^{-4}n^{1.12}L$, which is very close to being linear-time.

Table 2: Accuracy (in decimal digits) and timing (in seconds) for 20 largest MAX 3-CUT problems: n - size of matrix variable; m - number of constraints; "Pre-proc" - post-processing time; "gap" - duality gap; "pinf" - primal infeasibility; "dinf" - dual infeasibility; k - number of interior-point iterations; T - total interior-point time; "Post-proc" - post-processing time.

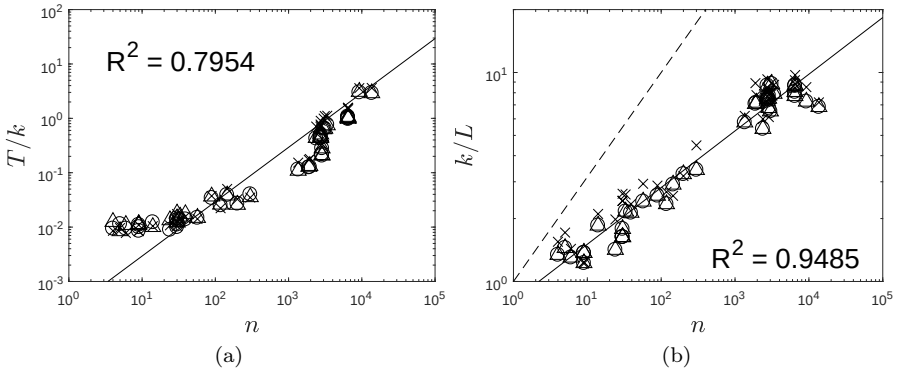
			Pre-	MOSEK					SeDuMi					Post-
#	n	m	proc	gap	pinf	dinf	k	T	gap	pinf	dinf	k	T	proc.
21	1354	3064	1.1	9.6	8.9	9.1	14	0.6	11.6	7.5	9.7	12	17.0	0.1
22	1888	4196	1.5	8.9	8.2	8.4	14	0.8	8.2	7.1	9.4	13	24.9	0.2
23	1951	4326	1.6	8.9	8.3	8.4	14	0.8	8.9	7.3	10.1	14	46.2	0.2
24	2383	5269	2.1	9.0	8.3	8.4	14	2.2	7.8	7.3	8.5	13	37.1	0.4
25	2736	5999	2.4	8.8	8.2	8.3	12	2.0	12.0	7.5	10.6	16	99.6	0.4
26	2737	6000	2.4	9.0	8.5	8.5	12	1.9	11.4	6.8	9.7	14	47.7	0.4
27	2746	6045	2.4	11.3	10.4	10.8	13	2.4	11.3	6.4	9.5	15	69.3	0.4
28	2746	6019	2.4	10.9	10.3	10.3	14	2.2	11.9	7.1	10.3	17	117.3	0.4
29	2848	6290	2.5	8.9	8.3	8.4	14	1.2	8.1	6.9	9.4	13	46.7	0.4
30	2868	6339	2.6	10.4	9.8	9.9	13	1.2	8.2	6.9	9.5	13	49.5	0.4
31	2869	6837	2.7	8.7	8.1	8.2	20	2.0	9.6	5.2	8.2	17	84.7	0.5
32	3012	6578	2.7	9.8	9.1	9.3	12	2.5	7.8	7.3	10.1	18	157.1	0.5
33	3120	6804	2.8	9.3	8.5	8.7	12	2.6	11.7	7.7	10.4	20	166.4	0.5
34	3374	7442	3.2	9.1	8.3	8.5	15	3.6	10.0	5.8	8.5	16	124.7	0.6
35	6468	14533	7.6	9.2	8.5	8.7	16	5.6	9.4	4.9	7.5	16	210.9	1.6
36	6470	14536	7.6	9.7	8.8	9.2	16	5.6	9.4	5.0	7.5	16	218.2	1.6
37	6495	14579	7.7	9.0	8.3	8.5	16	5.6	9.4	4.7	7.6	16	193.8	1.6
38	6515	14619	7.7	9.0	8.3	8.5	16	5.6	9.8	5.3	8.2	17	257.8	1.6
39	9241	23448	14.0	9.2	8.2	8.7	22	17.6	5.1	4.1	6.6	15	187.5	3.5
40	13659	32284	23.5	9.2	8.4	8.7	20	16.3	4.5	3.8	6.0	14	216.9	6.0

Table 3: Accuracy and Timing for 20 largest Lovasz Theta problems.

			Pre-	MOSEK					SeDuMi					Post-
#	n	m	proc	gap	pinf	dinf	k	T	gap	pinf	dinf	k	T	proc.
21	1355	1711	0.8	11.6	8.3	8.8	17	1.0	6.4	5.4	6.2	16	11.7	0.2
22	1889	2309	1.2	11.4	7.9	8.4	17	1.3	5.9	5.1	6.7	17	27.0	0.3
23	1952	2376	1.2	11.2	7.6	8.1	16	1.2	6.3	5.5	6.9	16	31.6	0.3
24	2384	2887	1.6	12.2	8.6	9.1	14	3.3	5.8	5.1	6.6	19	33.9	0.4
25	2737	3264	1.8	11.4	7.9	8.4	13	3.4	6.6	5.6	6.8	19	36.9	0.5
26	2738	3264	1.8	10.9	7.3	7.8	14	3.5	7.4	5.8	6.3	19	35.3	0.5
27	2747	3300	1.8	13.2	9.1	9.8	15	4.3	5.2	4.7	6.7	20	57.6	0.6
28	2747	3274	1.9	11.4	7.8	8.4	14	3.5	7.5	5.3	5.7	18	30.5	0.5
29	2849	3443	1.9	11.1	7.4	7.8	17	2.0	8.5	5.1	5.6	17	33.3	0.5
30	2869	3472	1.9	11.4	7.7	8.2	16	1.9	5.8	4.9	6.5	17	41.4	0.5
31	2870	3969	2.0	11.1	7.5	7.9	18	3.0	6.1	5.2	6.1	22	74.0	0.6
32	3013	3567	2.1	11.4	7.8	8.3	15	4.1	9.1	5.9	5.9	22	65.5	0.6
33	3121	3685	2.2	14.6	8.9	10.5	17	5.1	8.8	5.6	5.7	21	44.5	0.7
34	3375	4069	2.4	12.6	8.5	9.7	17	6.5	9.2	5.8	6.1	21	73.1	0.8
35	6469	8066	5.5	13.7	8.8	9.4	14	7.2	5.1	4.8	6.9	20	137.0	2.1
36	6471	8067	5.5	12.2	8.2	8.7	16	7.5	5.7	4.9	5.6	20	125.9	2.1
37	6496	8085	5.6	12.9	9.0	9.4	17	7.9	5.7	4.9	5.6	20	131.4	2.0
38	6516	8105	5.6	13.2	8.8	9.3	16	7.1	5.7	4.9	5.6	20	133.9	2.1
39	9242	14208	10.0	10.4	6.2	6.7	20	20.3	6.2	4.7	5.4	23	237.2	4.6
40	13660	18626	16.4	10.8	6.3	6.7	21	23.3	5.7	4.5	5.4	23	305.9	8.0

9.3 Optimal power flow

We now solve instances of the OPF posed on the same 40 power systems as mentioned above. Here, we use the MATPOWER function `makeYbus` to generate the bus admittance matrix Y_{bus}, and then manually generate each



(a) (b)

Fig. 4: OPF problems solved using clique tree conversion ($\times$), dualized clique tree conversion ($\circ$) and dualized clique tree conversion with auxillary variables ($\triangle$): (a) Time per iteration, with regression $T/k = 2.931\times10^{-4}n$; (b) Iterations per decimal digit of accuracy, with (solid) regression $k/L = 0.807n^{0.271}$ and (dashed) bound $k/L = \sqrt{n}$.

Table 4: Accuracy (in decimal digits) and timing (in seconds) for 20 largest OPF problems: n - size of matrix variable; m - number of constraints; "Pre-proc" - post-processing time; $L = \min\{\text{gap}, \text{pinf}, \text{dinf}\}$ - accurate decimal digits; k - number of interior-point iterations; T - total interior-point time; "Post-proc" - post-processing time.

			Pre-	CTC			Dual CTC			Dual CTC w/ aux			Post-
#	n	m	proc	L	k	T	L	k	T	L	k	T	proc.
21	1354	4060	3.0	7.3	45	6.9	7.2	42	4.2	7.1	41	4.5	0.2
22	1888	5662	4.1	7.2	64	11.0	6.9	48	6.3	6.8	48	6.2	0.3
23	1951	5851	4.2	7.8	61	10.9	7.1	46	6.0	7.0	50	6.7	0.3
24	2383	7147	5.9	7.2	43	30.4	6.9	38	16.6	6.9	37	16.2	0.4
25	2736	8206	7.0	7.2	60	46.2	6.8	53	23.8	6.5	48	22.3	0.5
26	2737	8209	6.8	7.1	66	45.7	6.7	57	24.5	6.9	53	23.1	0.5
27	2746	8236	7.0	6.9	50	45.1	6.6	50	24.7	6.3	47	23.9	0.5
28	2746	8236	6.9	7.1	60	44.2	6.7	56	25.2	6.9	60	26.7	0.6
29	2848	8542	6.8	7.1	56	18.9	6.4	49	10.1	6.4	48	10.2	0.5
30	2868	8602	6.8	7.4	56	18.8	6.6	51	10.6	6.7	52	10.8	0.5
31	2869	8605	7.4	7.7	47	19.6	7.1	46	12.7	7.4	50	14.1	0.6
32	3012	9034	7.9	7.0	55	54.5	6.1	54	31.6	6.9	45	28.5	0.6
33	3120	9358	8.1	7.2	64	70.7	6.3	58	38.5	6.7	59	38.2	0.7
34	3374	10120	8.9	7.1	62	69.3	6.6	56	39.8	6.6	52	39.0	0.7
35	6468	19402	17.9	7.6	64	99.9	7.0	54	53.7	6.9	53	56.7	2.0
36	6470	19408	18.0	7.4	68	106.1	6.8	57	56.3	6.9	56	57.2	2.0
37	6495	19483	17.7	7.5	66	102.8	7.3	54	53.2	7.0	60	62.3	2.0
38	6515	19543	17.7	7.2	70	103.4	6.8	54	54.7	6.8	59	58.1	2.0
39	9241	27721	31.3	7.5	64	230.1	7.0	57	165.0	7.6	55	169.7	4.3
40	13659	40975	47.9	6.8	49	177.4	7.9	48	154.6	7.9	54	157.4	7.7

constraint matrix A_i from Y_{bus} using the recipes described in [41]. Specifically, we formulate each OPF problem given the power flow case as follows:

- Minimize the cost of generation. This is the sum of real-power injection at each generator times $1 per MW.
- Constrain all bus voltages to be from 95% to 105% of their nominal values.
- Constrain all load bus real-power and reactive-power values to be from 95% to 105% of their nominal values.
- Constrain all generator bus real-power and reactive-power values within their power curve. The actual minimum and maximum real and reactive power limits are obtained from the case description.

We use three different algorithms based to solve the resulting SDP:

1. The original clique tree conversion of Fukuda and Nakata et al. [12,13], described in Section 4.
2. The dualized version of clique tree conversion, described in Section 7 as Algorithm 3.
3. The dualized version of clique tree conversion with auxillary variables, described in Section 8 as Algorithm 4.

Recall that only the third algorithm is guaranteed to have near-linear time complexity via Theorem 2. Nevertheless,

We solved all 40 problems using the three algorithms and MOSEK as the internal interior-point solver. Table 4 shows the accuracy and timing details for the 20 largest problems solved. All three algorithms achieved near-linear time performance, solving each problem instances to 7 digits of accuracy within 6 minutes. Upon closer examination, we see that the two dualized algorithms are both about a factor-of-two faster than the basic CTC method. Figure 4 plots T/k, the mean time taken per-iteration, and k/L, the number of iterations for a factor-of-ten error reduction, and their respective log-log regressions. The data suggests an empirical time complexity of $T \approx 2.3 \times 10^{-4} n^{1.3} L$ over the three algorithms.

10 Conclusion

Under the assumption of a sparsity graph with bounded treewidth, clique tree conversion splits a large semidefinite variable $X \succeq 0$ into many smaller semidefinite variables $X_j \succeq 0$. These smaller variables are coupled by "overlapping constraints", whose block-sparsity pattern coincides with the incidence matrix of a tree. These overlapping constraints can adversely affect complexity, causing highly sparse SDPs to be solved in cubic time and quadratic memory.

In this paper, we apply *dualization* to clique tree decomposition. Under a *decoupled* sparsity assumption, we show that the resulting normal equations have a block-sparsity pattern that coincides with the adjacency matrix of a tree graph, so the per-iteration complexity of an interior-point method is guaranteed to be *linear time* and *linear memory*. Problems that do not

satisfy the decoupled assumption can be systematically decoupled by introducing auxillary variables. In the case of *network flow* SDPs, the number of auxillary variables can be bounded, so an interior-point method again has a per-iteration complexity of linear time and memory.

Using these insights, we prove that the MAX k-CUT relaxation, the MAX BISECTION relaxation, the Lovasz Theta problem, and the AC optimal power flow SDP relaxation can all be solved in guaranteed *near-linear time* and *linear* memory. We present numerical evidence to support our theoretical claims. For the MAX 3-CUT and Lovasz Theta relaxations, our results have empirical time complexity for $T \approx 10^{-4} n^{1.12} L$ seconds for L accurate digits; for the AC optimal power flow relaxation, our empirical complexity is $T \approx 2.3 \times 10^{-4} n^{1.3} L$ to L accurate digits.

Acknowledgments

The authors are grateful to Daniel Bienstock, Salar Fattahi, Cédric Josz, and Yi Ouyang for insightful discussions and helpful comments on earlier versions of this manuscript.

References

1. L. Lovász, "On the Shannon capacity of a graph," *IEEE Transactions on Information Theory* **25**(1), pp. 1–7, 1979.
2. M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM* **42**(6), pp. 1115–1145, 1995.
3. H. D. Sherali and W. P. Adams, "A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems," *SIAM Journal on Discrete Mathematics* **3**(3), pp. 411–430, 1990.
4. L. Lovász and A. Schrijver, "Cones of matrices and set-functions and 0–1 optimization," *SIAM Journal on Optimization* **1**(2), pp. 166–190, 1991.
5. J. B. Lasserre, "An explicit exact SDP relaxation for nonlinear 0-1 programs," in *International Conference on Integer Programming and Combinatorial Optimization*, pp. 293–303, Springer, 2001.
6. M. Laurent, "A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0–1 programming," *Mathematics of Operations Research* **28**(3), pp. 470–496, 2003.
7. J. B. Lasserre, "Global optimization with polynomials and the problem of moments," *SIAM Journal on Optimization* **11**(3), pp. 796–817, 2001.
8. P. A. Parrilo, *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization.* PhD thesis, California Institute of Technology, 2000.
9. Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*, SIAM, 1994.
10. F. Alizadeh, "Interior point methods in semidefinite programming with applications to combinatorial optimization," *SIAM Journal on Optimization* **5**(1), pp. 13–51, 1995.
11. S. J. Benson, Y. Ye, and X. Zhang, "Solving large-scale sparse semidefinite programs for combinatorial optimization," *SIAM Journal on Optimization* **10**(2), pp. 443–461, 2000.
12. 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**(3), pp. 647–674, 2001.

13. K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima, and K. Murota, "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," *Mathematical Programming* **95**(2), pp. 303–327, 2003.
14. M. S. Andersen, J. Dahl, and L. Vandenberghe, "Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones," *Mathematical Programming Computation* **2**(3), pp. 167–201, 2010.
15. D. K. Molzahn, J. T. Holzer, B. C. Lesieutre, and C. L. DeMarco, "Implementation of a large-scale optimal power flow solver based on semidefinite programming," *IEEE Transactions on Power Systems* **28**(4), pp. 3987–3998, 2013.
16. R. Madani, M. Ashraphijuo, and J. Lavaei, "Promises of conic relaxation for contingency-constrained optimal power flow problem," *IEEE Transactions on Power Systems* **31**(2), pp. 1297–1307, 2016.
17. R. Madani, S. Sojoudi, G. Fazelnia, and J. Lavaei, "Finding low-rank solutions of sparse linear matrix inequalities using convex optimization," *SIAM Journal on Optimization* **27**(2), pp. 725–758, 2017.
18. R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*, Springer, 1999.
19. H. L. Bodlaender, "A linear-time algorithm for finding tree-decompositions of small treewidth," *SIAM Journal on computing* **25**(6), pp. 1305–1317, 1996.
20. J. F. Sturm, "Implementation of interior point methods for mixed semidefinite and second order cone optimization problems," *Optimization Methods and Software* **17**(6), pp. 1105–1154, 2002.
21. E. D. Andersen and K. D. Andersen, "The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm," in *High Performance Optimization*, pp. 197–232, Springer, 2000.
22. Y. Ye, M. J. Todd, and S. Mizuno, "An $O(\sqrt{nL})$-iteration homogeneous and self-dual linear programming algorithm," *Mathematics of Operations Research* **19**(1), pp. 53–67, 1994.
23. S. Kim, M. Kojima, M. Mevissen, and M. Yamashita, "Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion," *Mathematical Programming* **129**(1), pp. 33–68, 2011.
24. R. Madani, S. Sojoudi, and J. Lavaei, "Convex relaxation for optimal power flow problem: Mesh networks," *IEEE Transactions on Power Systems* **30**(1), pp. 199–211, 2015.
25. L. Vandenberghe, M. S. Andersen, *et al.*, "Chordal graphs and semidefinite optimization," *Foundations and Trends in Optimization* **1**(4), pp. 241–433, 2015.
26. M. S. Andersen, A. Hansson, and L. Vandenberghe, "Reduced-complexity semidefinite relaxations of optimal power flow problems," *IEEE Transactions on Power Systems* **29**(4), pp. 1855–1863, 2014.
27. J. Löfberg, "Dualize it: software for automatic primal and dual conversions of conic programs," *Optimization Methods and Software* **24**(3), pp. 313–325, 2009.
28. D. Fulkerson and O. Gross, "Incidence matrices and interval graphs," *Pacific Journal of Mathematics* **15**(3), pp. 835–855, 1965.
29. K. Fujisawa, S. Kim, M. Kojima, Y. Okamoto, and M. Yamashita, "User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems," tech. rep., Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2009. Research Report B-453.
30. R. A. Jabr, "Exploiting sparsity in SDP relaxations of the OPF problem," *IEEE Transactions on Power Systems* **27**(2), pp. 1138–1139, 2012.
31. R. Madani, A. Kalbat, and J. Lavaei, "ADMM for sparse semidefinite programming with applications to optimal power flow problem," in *IEEE 54th Annual Conference on Decision and Control (CDC)*, pp. 5932–5939, IEEE, 2015.
32. A. Kalbat and J. Lavaei, "A fast distributed algorithm for decomposable semidefinite programs," in *IEEE 54th Annual Conference on Decision and Control (CDC)*, pp. 1742–1749, IEEE, 2015.
33. D. Bienstock and G. Munoz, "Lp formulations for polynomial optimization problems," *SIAM Journal on Optimization* **28**(2), pp. 1121–1150, 2018.

34. Y. Sun, M. S. Andersen, and L. Vandenberghe, "Decomposition in conic optimization with partially separable structure," *SIAM Journal on Optimization* **24**(2), pp. 873–897, 2014.
35. S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine learning* **3**(1), pp. 1–122, 2011.
36. N. Parikh, S. Boyd, *et al.*, "Proximal algorithms," *Foundations and Trends in Optimization* **1**(3), pp. 127–239, 2014.
37. M. S. Andersen, J. Dahl, and L. Vandenberghe, "Logarithmic barriers for sparse matrix cones," *Optimization Methods and Software* **28**(3), pp. 396–423, 2013.
38. M. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT: A Python package for convex optimization," *abel.ee.ucla.edu/cvxopt* , 2013.
39. A. Frieze and M. Jerrum, "Improved approximation algorithms for MAX k-CUT and MAX BISECTION," *Algorithmica* **18**(1), pp. 67–81, 1997.
40. X. Bai, H. Wei, K. Fujisawa, and Y. Wang, "Semidefinite programming for optimal power flow problems," *International Journal of Electrical Power & Energy Systems* **30**(6-7), pp. 383–392, 2008.
41. J. Lavaei and S. H. Low, "Zero duality gap in optimal power flow problem," *IEEE Transactions on Power Systems* **27**(1), p. 92, 2012.
42. D. J. Rose, R. E. Tarjan, and G. S. Lueker, "Algorithmic aspects of vertex elimination on graphs," *SIAM Journal on computing* **5**(2), pp. 266–283, 1976.
43. F. Gavril, "The intersection graphs of subtrees in trees are exactly the chordal graphs," *Journal of Combinatorial Theory, Series B* **16**(1), pp. 47–56, 1974.
44. J. W. Liu, "The role of elimination trees in sparse factorization," *SIAM Journal on Matrix Analysis and Applications* **11**(1), pp. 134–172, 1990.
45. H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, "Approximating treewidth, pathwidth, frontsize, and shortest elimination tree," *Journal of Algorithms* **18**(2), pp. 238–255, 1995.
46. S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic Discrete Methods* **8**(2), pp. 277–284, 1987.
47. J. R. Gilbert, "Some nested dissection order is nearly optimal," *Information Processing Letters* **26**(6), pp. 325–328, 1988.
48. A. Agrawal, P. Klein, and R. Ravi, "Cutting down on fill using nested dissection: Provably good elimination orderings," in *Graph Theory and Sparse Matrix Computation*, pp. 31–55, Springer, 1993.
49. D. J. Rose, "Triangulated graphs and the elimination process," *Journal of Mathematical Analysis and Applications* **32**(3), pp. 597–609, 1970.
50. D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in *Graph Theory and Computing*, pp. 183–217, Elsevier, 1972.
51. J. Faraut and A. Korányi, *Analysis on symmetric cones*, Clarendon Press Oxford, 1994.
52. Y. E. Nesterov and M. J. Todd, "Self-scaled barriers and interior-point methods for convex programming," *Mathematics of Operations Research* **22**(1), pp. 1–42, 1997.
53. L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review* **38**(1), pp. 49–95, 1996.
54. 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 Journal on Optimization* **8**(3), pp. 746–768, 1998.
55. D. Goldfarb and K. Scheinberg, "Product-form Cholesky factorization in interior point methods for second-order cone programming," *Mathematical Programming* **103**(1), pp. 153–179, 2005.
56. J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software* **11**(1-4), pp. 625–653, 1999.
57. R. J. Vanderbei, "Affine-scaling for linear programs with free variables," *Mathematical Programming* **43**(1-3), pp. 31–44, 1989.
58. I. J. Lustig, "Feasibility issues in a primal-dual interior-point method for linear programming," *Mathematical Programming* **49**(1-3), pp. 145–162, 1990.
59. R. J. Vanderbei, "LOQO: An interior point code for quadratic programming," *Optimization Methods and Software* **11**(1-4), pp. 451–484, 1999.

60. R. Grone, C. R. Johnson, E. M. Sá, and H. Wolkowicz, "Positive definite completions of partial Hermitian matrices," *Linear Algebra and Its Applications* **58**, pp. 109–124, 1984.
61. J. Guo and R. Niedermeier, "Exact algorithms and applications for Tree-like Weighted Set Cover," *Journal of Discrete Algorithms* **4**(4), pp. 608–622, 2006.
62. T. A. Davis, *Direct methods for sparse linear systems*, vol. 2, Siam, 2006.
63. 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), pp. 69–88, 2008.
64. A. George and J. W. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Review* **31**(1), pp. 1–19, 1989.
65. R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems* **26**(1), pp. 12–19, 2011.
66. C. Josz, S. Fliscounakis, J. Maeght, and P. Panciatici, "AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE," *arXiv preprint arXiv:1603.01533* , 2016.
67. H. D. Mittelmann, "An independent benchmarking of SDP and SOCP solvers," *Mathematical Programming* **95**(2), pp. 407–430, 2003.
68. P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software (TOMS)* **30**(3), pp. 381–388, 2004.