Loraine - An interior-point solver for low-rank semidefinite programming

Soodeh Habibi, Michal Kočvara, Michael Stingl

Loraine—An interior-point solver for low-rank semidefinite programming

Soodeh Habibi[a], Michal Kočvara[a,b] and Michael Stingl[c]

[a]School of Mathematics, University of Birmingham, United Kingdom; [b]Institute of Information Theory and Automation, Prague, Czech Republic; [c]Applied Mathematics (Continuous Optimization), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

ARTICLE HISTORY
Compiled December 20, 2022

Dedicated to the memory of Oleg Burdakov

ABSTRACT
The aim of this paper is to introduce a new code for the solution of large-and-sparse linear Semidefinite Programs (SDPs) with low-rank solutions and/or low-rank data. We propose to use a preconditioned conjugate gradient method within an interior-point SDP algorithm and an efficient preconditioner fully utilizing the low-rank information. The efficiency is demonstrated by numerical experiments using the truss topology optimization problems, Lasserre relaxations of the MAXCUT problems and the sensor network localization problems.

KEYWORDS
Semidefinite optimization, interior-point methods, preconditioned conjugate gradients, truss topology optimization, MAXCUT problem, Lasserre relaxations, sensor network localization

AMS CLASSIFICATION
90C22, 90C51, 65F08, 74P05

1. Introduction

The first efficient solvers for semidefinite optimization emerged more than twenty years ago. All of the general purpose solvers have been using second-order algorithms, predominantly the interior-point (IP) method. In every iteration of such an algorithm, one has to solve a system of linear equations $Hx = g$ of size $n \times n$, where n is the dimension of the unknown vector x in the linear semidefinite programming (SDP) problem

$$\min_{x\in\mathbb{R}^n} c^\top x \quad \text{subject to} \quad \sum_{i=1}^{n} x_i A_i^{(k)} - B^{(k)} \succcurlyeq 0\,, \quad k = 1,\dots,p$$

with $A_i^{(k)}$, $B^{(k)} \in \mathbb{R}^{m\times m}$. The assembly and solution of this linear system is a well-known bottleneck of these solvers. This is the case even for problems with sparse data and sparse linear systems when the assembly of the system requires $\mathscr{O}\left(pnm^3\right)$ flops, and the solution by sparse Cholesky factorization $\mathscr{O}\left(n^\alpha\right)$ flops with some $\alpha \in [1,3]$.

CONTACT M. Kočvara. Email: m.kocvara@bham.ac.uk

On the other hand, many real-world applications lead to large-scale SDP, often unsolvable by current general-purpose software. There are, essentially, three ways how to approach this conundrum:

(1) Reformulating the problem to make it more suitable for general-purpose solvers. This can be done, for instance, by facial reduction [21, 29, 42] or by decomposition of large matrix inequalities (and thus reducing m) into several smaller ones [18, 19].
(2) Using a different algorithm, such as spectral bundle [14], ADMM [28], augmented Lagrangian [25, 41], optimization on manifolds [17], randomized algorithms [38] or techniques of nonlinear programming [8, 9].
(3) Using one of the second-order algorithms with Cholesky factorization replaced by an iterative method for the solution of the linear systems $Hx = g$; see, e.g., [20, 33, 39].

The last approach is particularly attractive whenever $n \gg m$, and addresses both bottlenecks of a second-order SDP solver.

- The assembling of the system matrix: an iterative solver only needs matrix-vector multiplications, and so the matrix does not have to be explicitly assembled and stored.
- The solution of the linear system itself: an iterative solver can handle very large systems, as compared to a sparse Cholesky solver, under the assumption of good conditioning of the matrix or existence of a good preconditioner.

When the data matrices $A_i^{(k)}$ are of very low rank (such as rank one), the complexity of the linear system assembling can be substantially reduced by using this fact. This is particularly true for rank-one dense matrices. This fact is, however, rarely utilized in standard SDP software, due to the complications related to a different data input.

Our goal in this paper is to introduce Loraine, a new general-purpose interior-point SDP solver targeted to problems with *low-rank data* and *low-rank solutions*. It employs special treatment of low-rank data matrices and, in particular, an option to use an iterative Krylov type method for the solution of the linear system. As the choice of an efficient preconditioner in a Krylov type method is problem dependent, in particular, in the context of optimization algorithms. We will focus on problems with expected very low-rank solutions.

SDP problems with low-rank solutions are common in relaxations of optimization problems in different areas such as combinatorial optimization [4], approximation theory [10, 22], control theory [22], structural optimization [31], and power systems [24].

It is *not* our goal to find a low-rank solution in case of non-unique solutions; indeed, it is well-known that an interior-point method will converge to a maximal complementary solution. However, if the method will converge to a low-rank solution, we will use this fact to improve its computational complexity and memory demands. The preconditioner is not only limited to problems with (theoretically exactly) low-rank solutions, it can be efficiently applied when the solution has a few outlying eigenvalues.

There is one major difference to other algorithms for SDP problems with low-rank solutions, such as SDPLR [8, 9], optimization on manifolds [17] and other approaches [3, 38]. All these methods require the knowledge of a guaranteed (possibly tight) upper bound on the rank of the solution. If the estimate of the rank was lower than the actual rank, the algorithm would not find a solution. In our approach, the rank information is merely used to speed up a standard, well-established algorithm. If our estimated rank is smaller than the actual one, we will only see it in possibly more iterations of the Krylov solver; the convergence behaviour of the optimization algorithm will be unchanged.

The actual optimization algorithm used in Loraine is a "standard" primal-dual interior-point method. The algorithm mimics the reliable primal-dual algorithm with Nesterov-Todd direction, as used, e.g., in the SDPT3 solver [32, 33].

Our preconditioner is motivated by the work of Zhang and Lavaei [39]. They present a preconditioner for the CG method within a standard IP method that makes it more efficient for large-and-sparse low-rank SDPs. We partly follow their approach though there are some major differences: we abandon a crucial assumption on the sparsity of the data matrices (Assumption 2 in [39]); our method allows for SDPs with more than one LMI and with (simple) linear constraints; the complexity of computing our preconditioner is substantially lower; we can also efficiently solve problems with a few outlying eigenvalues in the optimal solution.

To test Loraine and the underlying algorithms, we have generated a library of problems arising from truss topology optimization, sensor network localization and Lasserre relaxations of the MAXCUT problem. These problems are characterized by very low-rank solutions and scalability, in the sense that one can generate a range of problems of various dimension but of the same type. By comparing Loraine with other SDP solvers, we will demonstrate the efficiency of our approach.

The paper is organized as follows. Section 2 introduces the SDP problems we want to solve and some basic assumptions. In Sections 3 an IP method is briefly presented. Next, in Section 4, we introduce the preconditioners used with the iterative solvers. Section 5 presents the solver Loraine. Sections 6–8 describe our applications, truss topology optimization problems, the sensor network localization problem and Lasserre hierarchies of the MAXCUT problem. Finally, in Section 9, we present the results of our numerical experiments. We also give comparisons with other existing SDP software and demonstrate the high efficiency of our solver.

Notation We denote by $\mathbb{S}^m, \mathbb{S}^m_+$ and $\mathbb{S}^m_{++}$, respectively, the space of $m \times m$ symmetric matrices, positive semidefinite and positive definite matrices. The eigenvalues of a given matrix $X \in \mathbb{S}^m_{++}$ are ordered as $\lambda_1(X) \leq \cdots \leq \lambda_m(X)$, and its condition number is defined as $\kappa(X) = \lambda_1(X)/\lambda_m(X)$. The notation "vec" and "$\otimes$" refer to the (non-symmetrized) vectorization and Kronecker product, respectively, while "svec" and "smat" refer to symmetrized vectorization and its inverse operation. With this notation the following identity holds true: $\mathrm{vec}(AXB^\top) = (A \otimes B)\mathrm{vec}(X)$. The symbol $\bullet$ denotes the Frobenius inner product of two matrices, $A \bullet B = \mathrm{tr}(A^\top B)$. Finally, $e_i \in \mathbb{R}^n$ is a vector with one on the i-th position and zeros otherwise.

2. Low-rank semidefinite optimization

While Loraine is a general-purpose SDP solver, its main feature is the ability to efficiently solve SDP problems with low-rank solutions. The first part of the paper is thus focused on this feature.

We consider a (primal) semidefinite optimization problem with matrix variables $X_i \in \mathbb{S}^{m_i}$, $i = 1, \ldots, p$, and with linear constraints

$$\begin{aligned} \min_{X_1,\ldots,X_p,x_{\mathrm{lin}}} \ & \sum_{i=1}^{p} C_i \bullet X_i + d^\top x_{\mathrm{lin}} \\ \text{subject to } & \sum_{i=1}^{p} A_j^{(i)} \bullet X_i + (D^\top x_{\mathrm{lin}})_j = b_j, \quad j = 1, \ldots, n \\ & X \succcurlyeq 0,\ x_{\mathrm{lin}} \geq 0 \end{aligned} \tag{1}$$

together with its Lagrangian dual

$$\begin{aligned} \max_{y,S_1,\ldots,S_p,s_{\text{lin}}} \quad & b^\top y \\ \text{subject to} \quad & \sum_{j=1}^{n} y_j A_j^{(i)} + S_i = C_i, \quad i = 1,\ldots,p \\ & Dy + s_{\text{lin}} = d \\ & S_i \succcurlyeq 0, \quad i = 1,\ldots,p \\ & s_{\text{lin}} \geq 0\,. \end{aligned} \tag{2}$$

Here $A_j^{(i)} \in \mathbb{S}^{m_i}$, $j = 1,\ldots,n$, $i = 1,\ldots,p$, $C_i \in \mathbb{S}^{m_i}$, $i = 1,\ldots,p$, $b \in \mathbb{R}^n$, $D \in \mathbb{R}^{\nu\times n}$, $d \in \mathbb{R}^\nu$ are data of the problem.

Basic assumptions

We make the following assumptions.

Assumption 2.1. There exist strictly feasible $X_i \in \mathbb{S}^{m_i}_{++}$, $x_{\text{lin}} \in \mathbb{R}^\nu_{++}$, $y \in \mathbb{R}^n$, $S_i \in \mathbb{S}^{m_i}_{++}$, $s_{\text{lin}} \in \mathbb{R}^\nu_{++}$, $i = 1,\ldots,p$ satisfying the equality constraints in (1) and (2) (Slater's condition).

Assumption 2.2. Define the matrices $\mathbf{A}_i = [\text{vec}A_1^{(i)},\ldots,\text{vec}A_n^{(i)}]$, $i = 1,\ldots,p$. We assume that matrix-vector products with $\mathbf{A}_i$ and $\mathbf{A}_i^\top$ may each be computed in $\mathscr{O}(n)$ flops and memory.

Assumption 2.3. The inverse $(D^\top D)^{-1}$ and matrix-vector product with $(D^\top D)^{-1}$ may each be computed in $\mathscr{O}(n)$ flops and memory.

Assumption 2.4. The dimensions of X_i, $i = 1,\ldots,p$, are much smaller than the number of constraints, i.e., $m_i \ll n$.

Remark 1. Assumption 2.3 is satisfied, for instance, in case of box constraints on variable y in the dual formulation. Assumption 2.4 is not really needed as a "mathematical" assumption, the iterative methods presented in the paper would work without it. However, the complexity of these methods would then be comparable with standard software using Cholesky factorization. The presented methods will be only superior to the standard software under this assumption.

Low-rank assumption

As mentioned in the Introduction, our approach is focused on linear systems with matrices having few large outlying eigenvalues. This, of course, includes matrices with very low rank but also matrices with "approximate" low rank. These matrices frequently appear in optimization algorithms, such as interior-point methods, when the "exact" low rank is only attained at the (unreached) optimum. Moreover, the approach allows us to solve problems with matrices of high rank but with a few large outlying eigenvalues, such as the problems in Section 7.

Because our approach covers SDP problems with (genuinely) low-rank solutions and for the lack of better terminology for the larger class of matrices with outlying eigenvalues, we still call our main assumption "low-rank" and the problems we are targeting "SDP problems with low-rank solutions".

Our main assumption used in the design of an efficient preconditioner concerns the eigen-

value distribution of the solution $X^* = (X_1^*, \ldots, X_p^*)$ of (1). We assume that X^* has a very small number of large outlying eigenvalues. This includes problems where X_i^* are of a very low rank.

Assumption 2.5. Let X^* be the solution of (1). We assume that, for $i = 1, \ldots, p$, X_i^* has k outlying eigenvalues, i.e., that

$$(0 \leq)\ \lambda_1(X_i^*) \leq \cdots \leq \lambda(X_i^*)_{m_i-k} \ll \lambda(X_i^*)_{m_i-k+1} \leq \cdots \leq \lambda(X_i^*)_{m_i},$$

where k is very small, typically smaller than 10 and, often, equal to 1.

Remark 2. In order to emphasize the difference between our algorithm and other "low-rank algorithms", such as SDPLR [8] and the low-rank preconditioner introduced in [39]—that otherwise served as our motivation—we include the following remarks.

(1) We do not assume anything about the distribution of the first $(m_i - k)$ eigenvalues of X_i^*; they may not be equal to zero, they may not be clustered.
(2) We make no assumption about the "rank" of the solution x_{lin}^* of (1), i.e., about the number of active constraints in $Dy \leq d$ in (2). This can be arbitrarily large or small.
(3) We do not assume that the matrix $\mathbf{A}^\top\mathbf{A}$ is easily invertible. Indeed, it is *not* in our application in Section 6, contrary to examples in [39] where this matrix is diagonal.

3. Interior-point method

In this section, we will describe a primal-dual predictor-corrector interior-point method for SDP which is using the Nesterov–Todd (NT) direction, as implemented in Loraine. We will closely follow the articles by Todd, Toh and Tütüncü [32] and by Toh and Kojima [34] and only repeat formulas needed to present the structure of the matrix that is required to develop the preconditioners in Section 4. The basic framework of our interior-point solver thus, more or less, mimics that of the software SDPT3.

We use the notation of problems (1) and (2); however, for the sake of simplicity, we ignore the linear constraints in the following development and consider only one matrix variable, i.e., $p = 1$. Linear constraints and several matrix variables can, of course, be formally included into a single linear matrix inequality.

3.1. Basic framework

Let $\mathbf{A}$ be an $m^2 \times n$ matrix defined by

$$\mathbf{A} := [\text{vec}A_1, \ldots, \text{vec}A_n].$$

Our algorithm follows the standard steps of a primal-dual interior-point method. We write down optimality conditions for (1) with relaxed complementarity condition:

$$\begin{aligned} \mathbf{A}^\top \text{vec}(X) &= b, \\ \mathbf{A}y - S &= C, \\ XS &= \sigma\mu I, \end{aligned} \tag{3}$$

where $\mu = \frac{X \bullet S}{m}$ and σ is a centering parameter to be specified below. This system of nonlinear equations is solved repeatedly until convergence.

3.2. Newton direction

The system (3) is solved approximately by Newton's method, where in every iteration of the method we solve the following system of linear equations in variables $(\Delta X, \Delta y, \Delta S)$:

$$\mathbf{A}^\top \mathrm{vec}(\Delta X) = r_p, \tag{4a}$$

$$\mathbf{A}\Delta y + \mathrm{vec}(\Delta S) = \mathrm{vec}(R_d), \tag{4b}$$

$$H_W(X\Delta S + \Delta X S) = \sigma\mu I - H_W(XS)\,. \tag{4c}$$

Here

$$r_p := b - \mathbf{A}^\top X, \quad R_d := C - S - \sum_{i=1}^{n} y_i A_i,$$

and H_W is a linear transformation guaranteeing symmetry of the resulting matrix, in particular

$$H_W(M) = \frac{1}{2}\left(WMW^{-1} + W^{-\top}M^\top W^\top\right)$$

with some invertible matrix $W \in \mathbb{S}^m_{++}$ known as the *scaling matrix* [40]. By assuming (X, y, S) to be the current iterate, we define the so-called Nesterov-Todd (NT) scaling matrix

$$W = S^{-\frac{1}{2}}\left(S^{\frac{1}{2}}XS^{\frac{1}{2}}\right)^{\frac{1}{2}}S^{-\frac{1}{2}} = X^{\frac{1}{2}}\left(X^{-\frac{1}{2}}SX^{-\frac{1}{2}}\right)^{\frac{1}{2}}X^{\frac{1}{2}}$$

satisfying $X = WSW$ and $S = WX^{-1}W$. This is the scaling matrix used in Loraine.

Instead of solving the linear system of $2m^2 + n$ equations (4a)–(4c) directly, we can solve a Schur complement equation (SCE) involving only Δy. The general bottleneck of any interior-point method for SDP is assembling and solving this SCE

$$H\Delta y = r. \tag{5a}$$

Here H is the Schur complement matrix with elements

$$H_{ij} = A_i \bullet WA_jW, \quad i, j = 1, \ldots, n, \tag{5b}$$

and

$$r = r_p + \mathbf{A}^\top \mathrm{vec}\left(WR_dW + WSW\right). \tag{5c}$$

For details of computing W, r, ΔX and ΔS efficiently for the NT scaling, see [34].

By considering (5a) and (5b), we have the following linear system

$$(H\Delta y)_i = A_i \bullet \left[W\left(\sum_{j=1}^{n}\Delta y_i A_j\right)W\right] = r_i, \quad i = 1, \ldots, n. \tag{6}$$

Finally, vectorizing the matrix variables allows (6) to be written as

$$\left(\mathbf{A}^\top (W \otimes W)\mathbf{A}\right) \Delta y = r\,. \tag{7}$$

4. Preconditioners

As we mentioned before, the general bottleneck of the algorithm defined above is the assembling and solving of the Schur complement equation (7). One way to solve this equation is using an iterative method, in particular, a Krylov-type method such as the method of conjugate gradients (CG) or the minimum residual method (MINRES).

The Schur complement matrix H is large and the assembling of it may be expensive. Besides, and more importantly, it becomes increasingly ill-conditioned as the IP method makes progress toward the solution. So a successful CG-based solution of our linear systems must rely on an efficient preconditioner. In this section, we introduce two preconditioners implemented in Loraine and targeted pro-SDP problems with low-rank solutions.

4.1. Rank of H

The Schur complement matrix H can be written as a sum

$$H = \sum_{i=1}^{p} H^i_{\text{lmi}} + H_{\text{lin}}\,,$$

where H_{lin} is associated with linear constraints. Recall from (7) that each H^i_{lmi} is computed as

$$H^i_{\text{lmi}} = \mathbf{A}_i^\top (W_i \otimes W_i)\mathbf{A}_i\,, \tag{8}$$

where W_i is the NT scaling matrix.

In the next three Lemmata 4.1–4.3, we will omit the subscript, for simplicity, before returning to the full notation in Theorem 4.4. The next lemma shows that W has the same rank as X. So, as X is low-rank, W will be low-rank. Later in this section, we will use this low-rank property of W to present the preconditioners.

Let (X,S) be a pair of solutions to (1). By complementarity, we know that $XS = 0$, and $\text{rank}(X) + \text{rank}(S) = k + \sigma \leq m$. Assume $k + \sigma = m$, i.e., strict complementarity. From $XS = 0,\ X \succcurlyeq 0,\ S \succeq 0$, we know that X and S are simultaneously diagonalizable, i.e., there exists U such that $X = U\Lambda_X U^\top$, $S = U\Lambda_S U^\top$, where Λ_X and Λ_S are the diagonal matrices of eigenvalues and $U^\top U = I$.

Lemma 4.1. *Let X,S be as above. Let $W \in \mathbb{S}^m$ be any matrix such that $X = WSW$. Then $rank(W) = rank(X)$.*

Proof. Assume, without loss of generality, that eigenvalues of X and S are sorted such that $\Lambda_X = \text{Diag}\,(\lambda_1,\ldots,\lambda_k,0,\ldots,0)$ and $\Lambda_S = \text{Diag}\,(0,\ldots,0,\mu_1,\ldots,\mu_{m-k})$. Then $X = WSW$ is equivalent to $U\Lambda_X U^\top = WU\Lambda_S U^\top W$, and so, as $U^\top U = I$, $\Lambda_X = U^\top WU\Lambda_S U^\top WU$. Define $Z = U^\top WU$, so that $\Lambda_X = Z^\top \Lambda_S Z$. Because $(\Lambda_X)_{i,i} = 0$ for $i > k$, it must hold that $z_{:,i}^\top z_{:,i} = 0$, $i > k$, where $z_{:,i}$ is the i-th column of Z. As Z is positive semidefinite, it means that $Z_{i,j} = 0$ for $i,j > k$, i.e., the leading $k \times k$ submatrix of Z is a rank-k non-zero matrix and the rest of the matrix is zero. Hence, $\text{rank}(Z) = k$ and, as $W = UZU^\top$, so is the rank of W. □

The proof of the next lemma is straightforward.

Lemma 4.2. *Let $X \in \mathbb{S}^m$ such that* $\operatorname{rank} X = k$, $k \leq m$. *Then* $\operatorname{rank}(X \otimes X) = k^2$.

Lemma 4.3. *Let $Y \in \mathbb{S}^m$ such that* $\operatorname{rank} Y = k$, $k \leq m$, *and $A \in \mathbb{R}^{n\times m}$, $n < m$. Then* $\operatorname{rank}(AYA^\top) \leq k$.

Proof. Because $\operatorname{rank} Y = k$, we have $Y = \sum_{i=1}^k y^{(i)}(y^{(i)})^\top$ with some $y^{(i)} \in \mathbb{R}^m$, $i = 1,\ldots,k$. Hence $AYA^\top = A\left(\sum_{i=1}^k y^{(i)}(y^{(i)})^\top\right)A^\top = \sum_{i=1}^k z^{(i)}(z^{(i)})^\top$ with $z^{(i)} = Ay^{(i)}$. Therefore $\operatorname{rank}(AYA^\top) = \operatorname{rank}\sum_{i=1}^k z^{(i)}(z^{(i)})^\top \leq k$. A strict inequality occurs, trivially, when $n < k$. It can also occur when $z^{(i)}$, $i = 1,\ldots,k$, are linearly dependent. □

By combining the above results we get the following theorem.

Theorem 4.4. *Let, for some $i \in \{1,\ldots,p\}$, W_i be the scaling matrix from Section 3 and let* $\operatorname{rank} W_i = k$. *Let further H^i_{lmi} be defined as in (8). Then* $\operatorname{rank} H^i_{\text{lmi}} \leq k^2$.

Remark 3. All results in this section have been presented under the assumption of the exact rank of the involved matrices. This is, of course, only the limit case in our algorithms. The actual matrices have outlying eigenvalues and converge to the low-rank matrices. The message of Theorem 4.4 remains the same, though: when W_i has k outlying eigenvalues, then H^i_{lmi} will have at most k^2 outlying eigenvalues. This is the fact on which we build the preconditioner.

4.2. H_α preconditioner

The preconditioner introduced in this section has been motivated by the work of Zhang and Lavaei [39] and, in its derivation, we partly follow their paper; however, the new preconditioner differs in a substantial detail, as explained below. Also, our assumption about the matrix W is weaker.

In Section 3, we introduced the scaling matrix W. This matrix becomes progressively ill-conditioned as the IP method approaches the solution. This ill-conditioning of W is a result of Lemma 4.1. Assuming that the solution matrices X_i^*, $i = 1,\ldots,p$, have ranks k_i, the rank of matrices W_i will also tend to k_i.

The main idea of the preconditioner is to utilize Assumption 2.5 and decompose matrices W_i accordingly to their expected rank as follows:

$$W_i = \underbrace{\begin{bmatrix} V_i^s & V_i^l \end{bmatrix}\begin{bmatrix} \Lambda_i^s & 0 \\ 0 & \tau_i I \end{bmatrix}\begin{bmatrix} V_i^s & V_i^l \end{bmatrix}^\top}_{W_i^0} + \underbrace{V_i^l(\Lambda_i^l - \tau I)(V_i^l)^\top}_{U_iU_i^\top} \tag{9}$$

with τ_i satisfying $\lambda_1(W_i) \leq \tau_i < \lambda_{m_i-k_i}(W_i)$. Here U_i are $m_i \times k_i$ matrices of full column rank.

Recall now the form of the Schur complement matrix for problem (1):

$$H = \sum_{i=1}^p \mathbf{A}_i^\top (W_i \otimes W_i)\mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D, \tag{10}$$

in which $X_{\text{lin}} = \operatorname{diag}(x_{\text{lin}})$ and $S_{\text{lin}} = \operatorname{diag}(s_{\text{lin}})$.

Remark 4. Formally, we could treat the linear constraints, if present, as matrix constraints with diagonal matrices and perform the above decomposition for these constraints, too. How-

ever, it is rather unlikely that the corresponding part of the solution, vector x_{lin} would be "low rank", i.e., that only a very few of the linear constraints would be active. This fact could then ruin the whole idea. Here we propose to treat the linear constraints as "full rank" and add the matrix $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ to the preconditioner as is. In two of our applications, truss topology optimization and sensor network localization, the linear constraints only consist of upper and lower bounds (in the dual formulation), hence the matrix $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ is diagonal and satisfies Assumption 2.3. In the third application, relaxation of the MAXCUT problem, the matrix $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ is chordal and sparse ([13, Lemma 2.4]), so sparse Cholesky factorization leads to zero fill-in and is thus very efficient.

By substituting the splittings (9) for $i = 1, \ldots, p$ into the matrix (10), we get

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top \left(W_i^0 \otimes W_i^0 + U_i U_i^\top \otimes W_i^0 + W_i^0 \otimes U_i U_i^\top + U_i U_i^\top \otimes U_i U_i^\top \right) \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D.$$

Using the identity $\mathbf{A}^\top (\Phi \otimes \Xi) \mathbf{A} = \mathbf{A}^\top (\Xi \otimes \Phi) \mathbf{A}$ for any $\Phi, \Xi \in \mathbb{R}^{m_i \times m_i}$ with $\mathbf{A}$ defined as above ([39, Lemma 6] and [15, Chap. 4.2, Problem 25]), we obtain

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i + \sum_{i=1}^{p} \mathbf{A}_i^\top (U_i \otimes \Gamma_i)(U_i \otimes \Gamma_i)^\top \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D,$$

where Γ_i is any matrix satisfying $\Gamma_i \Gamma_i^\top = 2W_i^0 + U_i U_i^\top$. Next we define $V_i = \mathbf{A}_i^\top (U_i \otimes \Gamma_i)$ for $i = 1, \ldots, p$. Then,

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D + \tilde{V} \tilde{V}^\top, \tag{11}$$

where $\tilde{V} = [V_1, \ldots, V_p]$.

We now approximate $\mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i$ by $\tau_i^2 I$ in (11), to define the H_α preconditioner, which is

$$H_\alpha = \underbrace{\left(\sum_{i=1}^{p} \tau_i^2 I + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D \right)}_{\mathbf{A}_\alpha} + \tilde{V} \tilde{V}^\top. \tag{12}$$

By using the Sherman-Morrison-Woodbury (SMW) formula, its inverse will be

$$H_\alpha^{-1} = \mathbf{A}_\alpha^{-1} (I - \tilde{V} \Theta^{-1} \tilde{V}^\top \mathbf{A}_\alpha^{-1}), \tag{13}$$

where $\Theta = I + \tilde{V}^\top \mathbf{A}_\alpha^{-1} \tilde{V}$ is the Schur complement. Notice that, by Assumption 2.3, the inverse of $\mathbf{A}_\alpha$ is inexpensive.

Complexity of the PCG

- We compute the Schur complement

$$\Theta = I + [\mathbf{A}_1^\top (U_1 \otimes \Gamma_1) \ldots \mathbf{A}_p^\top (U_1 \otimes \Gamma_p)]^\top \mathbf{A}_\alpha^{-1} [\mathbf{A}_1^\top (U_1 \otimes \Gamma_1) \ldots \mathbf{A}_p^\top (U_p \otimes \Gamma_p)],$$

Algorithm 1 Solution of the linear system $H\Delta y = r$ by PCG

Given data $k > 0$ (solution rank), $W_i \in \mathbb{S}^m_{++}$, $i = 1, \ldots, p$, $r \in \mathbb{R}^n$ (right-hand side) and an initial iterate Δy^0.

procedure SETTING UP PRECONDITIONER H_α
  for $i = 1, \ldots, p$ **do** ▷ computing the decomposition (11)
    Compute eigenvalue decomposition $W_i = V_i \Lambda_i V_i^\top$ and set $\tau_i = \lambda_{\min}(W_i)$.
    Form the matrices W_i^0 and U_i by (9).
    Compute the Cholesky factorization $\Gamma_i \Gamma_i^\top = 2W_i^0 + U_i U_i^\top$.
  end for
  Compute the Schur complement $\Theta = I + \tilde{V}^\top \mathbf{A}_\alpha^{-1} \tilde{V}$ and its Cholesky factorization $\Theta = LL^\top$.
end procedure
procedure PCG
Use standard PCG algorithm to solve $H\Delta y = r$ until $\|H\Delta y - r\|/\|r\| \leq \varepsilon_{\text{CG}}$.
At each PCG iteration:
  Compute the matrix-vector product with H using (6).
  Compute the matrix-vector product with H_α^{-1} by means of the SMW in (13), using $\Theta^{-1} = L^{-\top} L^{-1}$.
end procedure

block-wise, with the (i, j)-block $\Theta_{ij} = B_i^\top B_j$, where $B_i = R^{-1} \mathbf{A}_i^\top (U_i \otimes \Gamma_i)$, $i = 1, \ldots, p$, and R is the Cholesky factor of $\mathbf{A}_\alpha$. Then, we multiply $(C_i =) R^{-1} \mathbf{A}_i^\top$, which is $\mathcal{O}(n)$ flops. The sparsity of $\mathbf{A}_i$ is maintained in C_i, and so $C_i d$ is still $\mathcal{O}(n)$ flops for an arbitrary vector d. Hence, as $(U_i \otimes \Gamma_i)$ has $m_i k_i$ columns, the product $C_i (U_i \otimes \Gamma_i)$ requires $\mathcal{O}(n m_i k_i)$ flops. Finally, we compute $B_i^\top B_j$, which requires $\mathcal{O}(m_i m_j k_i k_j)$ flops, if B_i and B_j are sparse as well. If this is not the case, the complexity would be $\mathcal{O}(m_i m_j k_i k_j n)$, however the additional factor n in the complexity formula can be avoided by reordering calculations appropriately. Define $\widehat{m} = \max_{i=1,\ldots,p} m_i$ and $\hat{k} = \max_{i=1,\ldots,p} k_i$. Forming and factorizing the Schur complement (i.e., forming the preconditioner) requires $\mathcal{O}\left(p^2 \widehat{m}^2 \hat{k}^2 + pn\widehat{m}\hat{k} + \widehat{m}^3 \hat{k}^3\right)$ flops; the last term comes from the Cholesky factorization of Θ.

- Each iteration of the PCG method is dominated by the matrix-matrix product in (6) requiring $\mathcal{O}\left(\sum_{i=1}^p m_i^3\right)$ flops and application of the preconditioner (13) requiring $\mathcal{O}\left(\sum_{i=1}^p m_i k_i\right)^2 + \mathcal{O}\left(n \sum_{i=1}^p m_i k_i\right)$ flops.
- The number of PCG iterations is a little unpredictable, in particular, when approaching the solution of the problem. However, our experience shows that this number is usually well below 10 and does not exceed 100 in the worst problems. Dropping the lower-order terms and assuming $n \approx \widehat{m}^2$, we will need $\mathcal{O}\left(\widehat{m}^3 \hat{k}^3\right)$ flops in the preconditioner and $\mathcal{O}\left(\widehat{m}^3 \hat{k}\right)$ flops in one PCG iteration. We come to the conclusion that the cost of the preconditioner is about the same as the cost of the solution of the linear system by the PCG method.

4.3. H_β and hybrid preconditioner

Recall the definition of the matrix H:

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D.$$

It turns out that, in our numerical examples, the last term is dominating in the first iterations of the IP algorithm, before the low-rank structure of W is clearly recognized. This is demonstrated in Figure 1 that shows distribution of eigenvalues of matrices $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ (in red) and $\sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top$ (in blue) in iterations 2, 17, 30 (final) in problem `tru7e` (see Section 9). This observation lead to the idea of a simplified preconditioner called H_β and

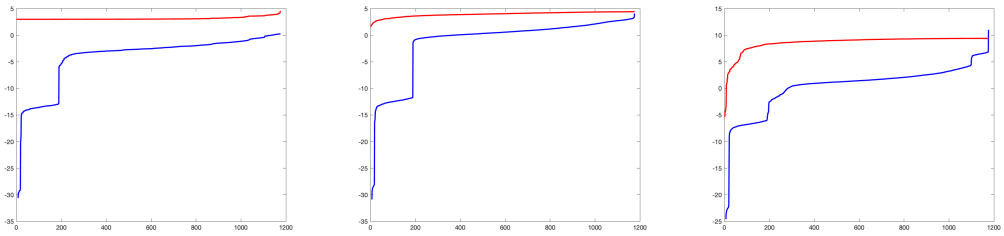

Figure 1. Problem tru7e; eigenvalues of $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ (red) and $\sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top$ (blue) in iteration 2, 17 and 30 (left to right).

defined as follows

$$H_\beta = \sum_{i=1}^{p} \tau_i^2 I + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D, \tag{14}$$

in which τ_i is defined as in the previous section. This matrix is easy to invert by Assumption 2.3; in fact, the matrix is diagonal in many problems. It is therefore an extremely "cheap" preconditioner that is efficient in the first iterations of the IP algorithm.

For relevant problems, we therefore recommend to use a *hybrid preconditioner*: we start the IP iterations with H_β and, once it becomes inefficient, switch to the more expensive but more efficient H_α. See more details in Section 9.2 and an example in Section 9.4.2.

5. The code Loraine

Algorithms presented in the previous sections have been implemented in a code Loraine (for LOw-RAnk INtErior point). The implementation was done in MATLAB and Julia programming environments and is available as open source[1].

5.1. Special features

Loraine is a general-purpose solver for any linear SDP with linear equality and inequality constraints. Compared to other general-purpose SDP software, it particularly targets two classes of problems.

[1] github.com/kocvara/Loraine.m

Problems with low-rank solutions

Loraine is using the preconditioned conjugate gradient method, as described in detail in Section 4. In particular, the user can choose between the direct and iterative solver, the type of preconditioner and the expected rank of the solution. The direct solver relies on the implementation of the (sparse or dense) Cholesky factorization provided by MATLAB or Julia.

Problems with low-rank data input

This feature is only useful when a direct solver is used. In this case, the bottleneck of an interior-point algorithm is the computation of the Schur complement matrix in (5b) and (7). When the data matrices A_i are of low rank (typically of rank one), the complexity of (5b) can be drastically reduced. The user has a choice

- to provide vectors $(a_i^j)_k,\ k = 1,\ldots,r$, defining matrices $A_j^{(i)} = \sum_{k=1}^{r}(a_i^j)_k(a_i^j)_k^\top$ (field `Avec` in the Data input section below);
- to indicate that the input matrices $A_j^{(i)}$ are all expected to be of rank one. Then their decomposition to a vector-vector product can be automatically computed by Loraine.

5.2. Data input

The notation in the input file is related to the dual formulation of the problem (2) without slack variables. The input for the MATLAB version of Loraine is a MATLAB structure with the following fields:

type ... 'sdp' [field used for future extensions]
name ... 'name of the problem'
nvar ... number of variables n
c ... objective vector b
nlmi ... number of linear matrix inequalities p
msizes ... dimensions of the LMIs $m_1,\ldots,m_p$
nlin ... number of linear constraints ν
lsi_op ... (optional) binary vector of length nlin, set to zero for equality constraints and one for inequality constraints; if this vector is not present and $\nu > 0$, inequality type constraints are assumed.
A ... matrices $A_j^{(i)}$ stored in MATLAB sparse format as cells `A{i,j}`
Avec ... vectors $(a_i^j)_k,\ k = 1,\ldots,r$, defining matrices $A_j^{(i)} = \sum_{k=1}^{r}(a_i^j)_k(a_i^j)_k^\top$
C ... matrix C of linear constraints
d ... vector d of linear constraints

We also provide several converters, e.g., from SDPA input files. For the Julia version of Loraine, the MATLAB structure can be exported and read by the Julia code.

5.3. Loraine options

The following parameters can be changed by the user:

```
kit = 1;           % 0..direct solver; 1..iterative solver
tol_cg = 1e-2;     % initial tolerance for iterative solver
tol_cg_up = 0.5;   % tolerance update
tol_cg_min = 1e-6; % minimal tolerance for CG solver
cg_type = 'minres';% 'minres' or 'cg' implemented
```

```
eDIMACS = 1e-6;    % epsilon for DIMACS error stopping criterion
prec = 1;      % 0..no; 1..H_alpha; 2..H_beta; 4..hybrid
erank = 1;     % estimated rank of the solution
mup = 1000;    % initial penalty parameter mu for l1-penalization
verb = 2;      % 2..full output, 1..short output, 0..no output
datarank = 0; % -1..A_i expected rank one; 0..full rank; 1..A_i by vectors
initpoint = 0;% 0..Loraine heuristics, 1..SDPT3-like heuristics
```

6. Application 1: Truss topology optimization

6.1. Truss notation

The notation used throughout Section 6 is specific to this application and unrelated to the rest of the paper.

By truss, we understand a mechanical structure, an assemblage of pin-jointed uniform straight bars made of elastic material, such as steel or aluminium. The bars can only carry axial tension and compression. We denote by m the number of bars and by N the number of joints. The positions of the joints are collected in a vector y of dimension $\widetilde{n} := dim \cdot N$ where dim is the spatial dimension of the truss. The material properties of bars are characterized by their Young's moduli E_i, the bar lengths are denoted by ℓ_i and bar volumes by t_i, $i = 1,\ldots,m$.

Let $f \in \mathbb{R}^{\widetilde{n}}$ be a load vector of nodal forces. The response of the truss to the load f is measured by nodal displacements collected in a displacement vector $u \in \mathbb{R}^{\widetilde{n}}$. Some of the displacement components may be restricted: a node can be fixed in a wall, then the corresponding displacements are prescribed to be zero. The number of free nodes multiplied by the spatial dimension will be denoted by n and we will assume that $f \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$.

We introduce the bar stiffness matrices K_i and assemble them in the global stiffness matrix of the truss

$$K(t) = \sum_{i=1}^{m} t_i K_i = \sum_{i=1}^{m} t_i \frac{E_i}{\ell_i^2} \delta_i \delta_i^\top, \quad i = 1,\ldots,m \tag{15}$$

with "position vectors" $\delta_i \in \mathbb{R}^n,\ i = 1,\ldots,m$, and Young's moduli $E_i \in \mathbb{R},\ i = 1,\ldots,m$, characterizing the stiffness of the material. The truss must satisfy the equilibrium equation

$$K(t)u = f\,. \tag{16}$$

Assumption 6.1. $K(\mathbf{1})$ with $\mathbf{1} = (1,1,\ldots,1) \in \mathbb{R}^m$ is positive definite and the load vector f is in the range space of $K(\mathbf{1})$.

6.2. Truss topology problem, rank of the solution

Let $0 \le \underline{t}_i \le \bar{t}_i,\ i = 1,\dots,m$, and γ be a positive constant. The basic "minimum volume" truss topology optimization (TTO) problem reads as follows:

$$\begin{aligned} &\min_{t\in\mathbb{R}^m} \sum_{i=1}^{m} t_i \\ &\text{subject to} \\ &\qquad \begin{pmatrix} \gamma & -f^\top \\ -f & K(t) \end{pmatrix} \succcurlyeq 0 \\ &\qquad \underline{t}_i \le t_i \le \bar{t}_i, \quad i = 1,\dots,m. \end{aligned} \tag{17}$$

The dual to (17) can be written as

$$\begin{aligned} &\max_{\substack{X\in\mathbb{S}^{n+1} \\ \overline{\rho}\in\mathbb{R}^m, \underline{\rho}\in\mathbb{R}^m}} \begin{pmatrix} -\gamma & f^\top \\ f & 0 \end{pmatrix} \bullet X - \sum_{i=1}^{m} \overline{\rho}_i \bar{t}_i + \sum_{i=1}^{m} \underline{\rho}_i \underline{t}_i \\ &\text{subject to} \\ &\qquad \begin{pmatrix} 0 & 0 \\ 0 & K_i \end{pmatrix} \bullet X - \overline{\rho}_i + \underline{\rho}_i = 1, \quad i = 1,\dots,m \\ &\qquad X \succcurlyeq 0 \\ &\qquad \overline{\rho}_i \ge 0,\ \underline{\rho}_i \ge 0, \quad i = 1,\dots,m. \end{aligned} \tag{18}$$

Lemma 6.1. *Problems (17), (18) satisfy Assumption 2.2 and Assumption 2.3.*

Proof. Every vector δ_i has at most 4 non-zero elements (6 in 3D space) and thus every matrix K_i has at most 16 (36) non-zero elements, independently of the size of the problem; hence Assumption 2.2 is satisfied. Assumption 2.3 is trivially satisfied for box constraints on t_i. □

Using SDP complementarity, it is straightforward to prove the following result.

Theorem 6.2. *There exists a solution $(X^*, \overline{\rho}^*, \underline{\rho}^*) \in \mathbb{S}^{n+1} \times \mathbb{R}^m \times \mathbb{R}^m$ of the dual SDP problem (18) such that the rank of X^* is one. For $\underline{t}_i > 0$ this solution is unique.*

6.3. Truss topology problem, vibration constraints

Formulation of the basic truss topology problem as an SDP is rather academic. It was shown, e.g., in [16] that it is equivalent to a convex nonlinear programming problem that can be solved very efficiently by interior point methods. The SDP formulation gains significance once we add more, important and practical, constraints to the problem. In particular, it was shown in [1, 2] that a constraint on natural (free) vibrations of the optimal structure leads to a linear matrix inequality and that this is, arguably, the best way how to treat this constraint.

The free vibrations are the squares of the eigenvalues of the generalized eigenvalue problem

$$K(t)w = \lambda\,(M(t) + M_0)\,w, \tag{19}$$

where $M(t) = \sum_{i=1}^{m} t_i M_i$ is the so-called mass matrix that collects information about the mass distribution in the truss. The matrices M_i are positive semidefinite and have the same sparsity

structure as K_i. The non-structural mass matrix M_0 is a constant, typically diagonal matrix with very few nonzero elements.

Low vibrations are dangerous and may lead to structural collapse, hence a typical vibration constraint is $\lambda_{\min} \geq \underline{\lambda}$ for a given $\underline{\lambda} > 0$ where $\lambda_{\min}$ is the smallest eigenvalue of problem (19). This constraint can be equivalently written as a linear matrix inequality $K(t) - \underline{\lambda}\,(M(t) + M_0) \succcurlyeq 0$. We will thus get the following SDP formulation of the truss topology design with a vibration constraint:

$$\begin{aligned} &\min_{t \in \mathbb{R}^m} \sum_{i=1}^{m} t_i \\ &\text{subject to} \\ &\qquad \begin{pmatrix} \gamma & -f^\top \\ -f & K(t) \end{pmatrix} \succcurlyeq 0 \\ &\qquad K(t) - \underline{\lambda}\,(M(t) + M_0) \succcurlyeq 0 \\ &\qquad \underline{t}_i \leq t_i \leq \bar{t}_i, \quad i = 1, \ldots, m. \end{aligned} \tag{20}$$

It has been shown, e.g., in [31] that, when $\underline{t}_i > 0$, the rank of the optimal dual variable to the second matrix inequality is equal to the multiplicity of the smallest eigenvalue of (19). This multiplicity depends on the geometry of the optimal structure but is, typically, very low, usually not bigger than 1 or 2. Problem (20) thus fits in our framework of SDPs with very low-rank solutions.

7. Application 2: Sensor network localization with noisy data

7.1. The problem

Assume there are m distinct (sensor) points in $x_i \in \mathbb{R}^d$, whose locations are to be determined, and ν other fixed (anchor) points, whose locations $a_1, \ldots, a_\nu$ are known. The Euclidean distance d_{ij} between the ith and jth sensor point is known if $(i,j) \in \mathscr{I}_x = \{(i,j) \mid \|x_i - x_j\| = \delta_{ij} \leq \rho\}$, where ρ is a fixed parameter called the radio range. Similarly, the Euclidean distance d_{kj} between the kth anchor and jth sensor point is known if $(k,j) \in \mathscr{I}_a = \{(k,j) \mid \|a_k - x_j\| = \delta_{ij} \leq \rho\}$. The *sensor network localization* (SNL) problem is to

$$\begin{aligned} &\text{Find } x_i \in \mathbb{R}^d,\ i = 1, \ldots, m, \text{ for which} \\ &\qquad \|x_i - x_j\|^2 = \delta_{ij}^2, \qquad (i,j) \in \mathscr{I}_x \\ &\qquad \|a_k - x_j\|^2 = \overline{\delta}_{kj}^2, \qquad (k,j) \in \mathscr{I}_a\,. \end{aligned} \tag{21}$$

Typical networks of this type consist of a large number of densely deployed sensor nodes which gather local data and communicate with other nearby nodes. The sensor data from the nodes are relevant only if we know to what location they refer. The problem arises in applications as different as the habitat monitoring system in the Great Duck Island, detecting volcano eruptions, industrial control in semiconductor manufacturing plants, structural health monitoring, battlefield surveillance, moving object tracking, asset location and the problem of determining protein molecule structure [5].

Figure 2–left shows an example of a network with 9 anchors and 512 sensors.

In many applications the data δ_{ij} are noisy. Assume perturbation of δ_{ij} and $\overline{\delta}_{ik}$ by random

noises ε_{ij} and ε_{ik}, respectively:

$$\delta_{ij} = \hat{\delta}_{ij}|1+\varepsilon_{ij}|, \quad (i,j) \in \mathscr{I}_x$$
$$\overline{\delta}_{ik} = \hat{\overline{\delta}}_{ik}|1+\varepsilon_{ik}|, \quad (i,j) \in \mathscr{I}_a$$

where $\hat{\delta}_{ij}$ are the true distances. Obviously, the problem (21) may now become infeasible.

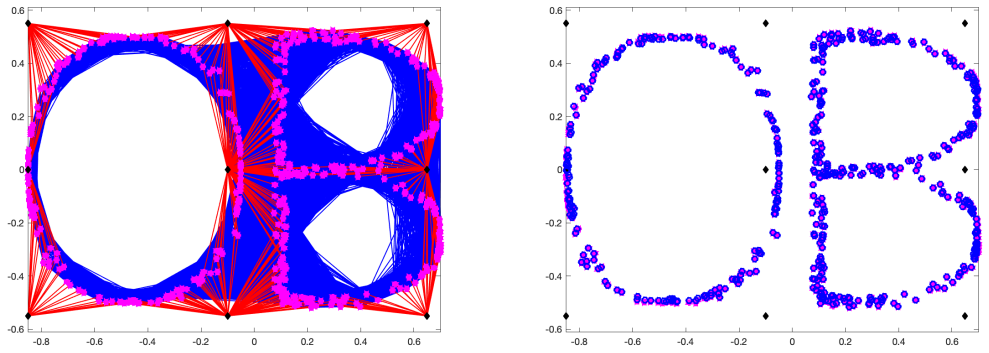

Figure 2. Sensor network localization problem: initial data (left) and optimal solution of the SDP relaxation (right). Black: anchors; magenta: exact positions of sensors; red (left): known distances between anchors and sensors; blue (left): known mutual distances of sensors; blue (right): computed positions of sensors.

7.2. SDP relaxation

In a series of papers, Biswas, Ye and their co-authors [5, 6, 35] proposed a relaxation of the problem based on the semidefinite programming formulation. The next lines follow the development from Biswas and Ye [6].

Let $X = [x_1\ x_2\ \ldots\ x_m]$ be a $d \times m$ unknown matrix. Then

$$\|x_i - x_j\|^2 = (e_i - e_j)^T X^T X (e_i - e_j)$$
$$\|a_k - x_j\|^2 = (a_k; -e_j)^T \begin{bmatrix} I_d \\ X^T \end{bmatrix} [I_d\ X](a_k; -e_j)$$

and the original problem (21) can be equivalently written as

$$\begin{aligned} &\text{Find } x_i \in \mathbb{R}^d,\ i = 1,\ldots,m, \text{ for which} \\ &\qquad (e_i - e_j)^T X^T X (e_i - e_j) = \delta_{ij}^2 \\ &\qquad (a_k; -e_j)^T \begin{pmatrix} I_d & X \\ X^T & Y \end{pmatrix} (a_k; -e_j) = \overline{\delta}_{kj}^2 \\ &\qquad Y = X^T X\,. \end{aligned} \tag{22}$$

We now relax the equality $Y = X^T X$ to $Y \succcurlyeq X^T X$, which is equivalent to the linear matrix

inequality

$$Z = \begin{pmatrix} I_d & X \\ X^T & Y \end{pmatrix} \succcurlyeq 0\,.$$

The semidefinite optimization problem

$$\begin{aligned} &\min_{Z,u,v} \sum_{(i,j)\in\mathscr{I}_x} (u_{ij}+v_{ij}) + \sum_{(k,j)\in\mathscr{I}_a} (u_{kj}+v_{kj}) \qquad (23)\\ &\text{subject to}\\ &\quad Z_{1:d,1:d} = I_d\\ &\quad (0; e_i - e_j)^T Z (0; e_i - e_j) - u_{ij} + v_{ij} = \delta_{ij}^2, \quad (i,j)\in\mathscr{I}_x\\ &\quad (a_k; -e_j)^T Z (a_k; -e_j) - u_{kj} + v_{kj} = \overline{\delta}_{kj}^2, \quad (k,j)\in\mathscr{I}_a\\ &\quad Z \succcurlyeq 0\\ &\quad u_{ij}, v_{ij} \geq 0, \quad (i,j)\in\mathscr{I}_x\,, \qquad u_{kj}, v_{kj} \geq 0, \quad (k,j)\in\mathscr{I}_a \end{aligned}$$

is called the *full SDP relaxation* (FSDP) of (21). It has been shown by So and Ye [30] that for certain "good" problems with no data noise this relaxation is exact.

For larger sensor network localization problems, the FSDP relaxation is difficult to solve numerically. The SDP problem has a large number of variables and the constraint matrix is thus *large* and *full*. In order to solve the problem, one may exploit the sparsity of $\mathscr{I}_x$ and $\mathscr{I}_a$ at the relaxation modelling level. Several approaches have been published on this subject. A simple, yet powerful approach was proposed by Wang et al. [35] and called *edge-based relaxation* (ESDP). However, ESDP does not maintain the localizability of FSDP: the solution can have high rank even for uniquely localizable problems. For problems with no data noise, Krislock and Wolkowicz [21] proposed a very efficient algorithm based on facial reductions. This approach, however, cannot be extended to the problems with noisy data.

As mentioned above, for many noiseless problems the relaxation is exact, i.e., the rank of the matrix Z is two. Then, for problems with small noise, Z is expected to have two large eigenvalues and several, perhaps many, small nonzero ones. The problem thus satisfies our low-rank Assumption 2.5 and can be efficiently solved by Loraine. This will be demonstrated in Section 9.

8. Application 3: Lasserre relaxations of the MAXCUT problem

8.1. The problem

Let Γ be an undirected n-node graph and let the arcs (i,j) be associated with nonnegative weights a_{ij}. The task is to find a cut of the largest weight, that is, to partition the set of nodes into two parts, S and S' such that the total weight of all arcs linking S and S' is as large as possible.

The MAXCUT problem is formulated as a quadratic optimization problem

$$\max_x \left\{ \frac{1}{4} \sum_{i,j=1}^{n} a_{ij}(1 - x_i x_j) \mid x_i^2 = 1,\ i = 1,\ldots,n \right\} \qquad (24)$$

or, equivalently, with $X = xx^T$, as a linear SDP with a rank-one constraint:

$$\max_X \left\{ \frac{1}{4} \sum_{i,j=1}^{n} a_{ij}(1 - X_{ij}) \mid X_{ii} = 1,\ i = 1, \ldots, n;\ \text{rank} X = 1;\ X \succeq 0 \right\}. \tag{25}$$

It is well-known that problem (24) is NP-hard. The celebrated result by Goemans and Williamson [12] shows that the (Shor's) relaxation of (25) by ignoring the rank constraints will deliver a solution with objective function not bigger than $\frac{1}{0.87865}$ times the global optimum. The relaxation is, however, almost never exact which means that the solution X^* of the relaxed (25) has, typically, an unknown rank bigger than one.

Tighter approximations with low-rank solutions can be obtained by higher order relaxations, introduced by Lasserre [22]. The original problem (24) is a polynomial optimization problem and we can just use the Lasserre machinery to obtain tight approximations of its global solution. It was shown by Fawzi et al. [11] that for this type of problems, the sequence of these approximations is *finite* and the upper bound on the order of the relaxation to obtain exact solution of (24) is $\lceil n/2 \rceil$. Laurent [23] showed that this is also a lower bound for unweighted complete graphs.

The SDP relaxations can be written in the following form:

$$\begin{aligned} &\min_{y \in \mathbb{R}^s} y^\top q \\ &\text{subject to } \sum_{i=1}^{s} M_i y_i + I \succcurlyeq 0\,. \end{aligned} \tag{26}$$

Here, for the relaxation of order one, Q is the Laplacian matrix of the graph, $q = \text{svec}(Q)$, $y = \text{svec}(X)$, $s = n(n+1)/2$, and $M_i \in \mathbb{R}^{n+1 \times n+1}$ are the (pseudo-)Hankel matrices; in this case (26) is just re-writing of the relaxed problem (25) (without the rank constraint). For higher-order relaxations, the dimensions s and m get bigger by adding higher-order terms in M_i and y; for details, see [22].

For an exact relaxation (i.e., relaxation delivering the global solution of (24)), the rank of the optimal moment matrix $M^* := \sum_{i=1}^{s} M_i y_i^* + I$ is equal to the number of global solutions; in particular, if the global solution is unique, the rank of M^* is one. So, while problem (26) has a low-rank solution, it is in the "wrong" form for our IP solver. While the solution M^* of (26) is of low rank, our solver expects the solution of an SDP problem in the *primal form* (1) to be of low rank. In [13] we show that the dual to (26) can be re-written in the dual form

$$\begin{aligned} &\max_{z \in \mathbb{R}^{\widetilde{n}}} (\text{svec}(I))^\top z \\ &\text{subject to } \text{smat}(z) \succcurlyeq 0 \\ &\qquad\qquad \mathbf{M} z = \tilde{q}\,, \end{aligned} \tag{27}$$

where $\widetilde{n} = m(m+1)/2$ and $\mathbf{M} = (\text{svec}(M_1), \ldots, \text{svec}(M_s))^T$, $\mathbf{M} \in \mathbb{R}^{s \times \widetilde{n}}$. Now, due to complementarity, we know that the dual solution to this problem (the Lagrangian multiplier to the constraint $\text{smat}(z) \succcurlyeq 0$ and the solution to (26)) is of low rank, assuming high enough relaxation order and unique solution of (24). Also in [13] we show how to treat the equality constraints in (27) in the interior point algorithm; in particular, we use the ℓ_1-penalty method for this purpose.

Problem (27) now satisfies the assumptions needed for the low-rank preconditioner.

9. Numerical experiments

9.1. Problem database

To test Loraine and the underlying algorithms, we generated a library of problems arising from truss topology optimization, sensor network localization and Lasserre relaxations of the MAXCUT problem. These problems are characterized by very low-rank solutions and scalability, in the sense that one can generate a range of problems of various dimension but of the same type.

9.1.1. TTO problems

We generated a set of truss topology optimization problems of various sizes, both with and without the vibration constraint. All problems have the same geometry, boundary conditions and loads and only differ in the number of potential nodes and bars. The geometry and loading for TTO problems without vibration (formulation (17)) are as shown in Figure 3-left, while the data for problems with the vibration constraint (formulation (20)) differ only in the horizontal orientation of the load vector. We consider two groups of problems with the following naming convention:

`tru<n>` standard TTO problem (17) with $\underline{t} = 0$, discretized by $n \times n$ nodes;
`vib<n>` same as above but with the vibration constraint (20).

In each group, all nodes in the initial ground structure are connected by potential bars. Table 1 presents the dimensions of the generated problems.

Table 1. Problems `tru<n>` and problems `vib<n>`, number of variables n, size of the LMI constraint m and number of linear constraints.

problem	n	m	lin. constr.
tru3	36	13	72
tru5	300	41	600
tru7	1176	85	2352
tru9	3240	145	6480
tru11	7260	221	14520
tru13	14196	313	28392
tru15	25200	421	50400
tru17	41616	545	83232
tru19	64980	685	129960
tru21	97020	841	194040
tru23	139656	1013	279312
tru25	195000	1201	390000

problem	n	m	lin. constr.
vib3	36	(13, 12)	72
vib5	300	(41, 40)	600
vib7	1176	(85, 84)	2352
vib9	3240	(145, 144)	6480
vib11	7260	(221, 220)	14520
vib13	14196	(313, 312)	28392
vib15	25200	(421, 420)	50400
vib17	41616	(545, 544)	83232
vib19	64980	(685, 684)	129960
vib21	97020	(841, 840)	194040
vib23	139656	(1013, 1012)	279312
vib25	195000	(1201, 1200)	390000

For illustration, Figure 3 shows optimal results for problems `tru25` and `vib19`.

9.1.2. SNL problems

We consider SNL problems of growing dimension with nine anchors and n sensors randomly distributed around the letters O and B; see Figure 2. The radio range ρ (see the beginning of Section 7) was set to $\frac{1}{5}$ of the maximal vertical distance of the anchors. The known distances between the sensors were perturbed by 1% Gaussian noise. Table 2 (left) shows the dimensions of the generated problems.

9.1.3. MAXCUT problems

Recall that the solution of the MAXCUT problem is of rank one only if the problem has a unique global solution. Obviously, a complete unweighted graph may have many “symmetric"

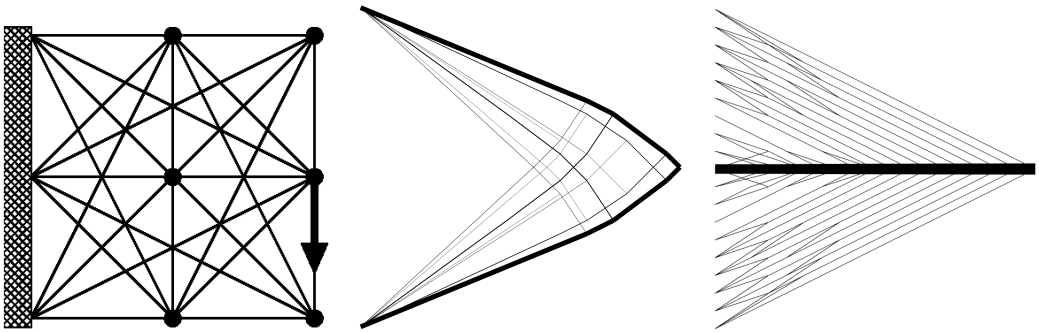

Figure 3. Initial design for tru* problems (left); optimal results for problems tru25 (middle) and vib19 (right).

solutions. To avoid the non-uniqueness, we generated undirected, weighted, generally complete graphs with weights randomly distributed between 0 and 12 with 20–50 nodes, using the MATLAB command `graph`. It turned out that for all these problems the relaxation order two in the Lasserre hierarchy is already high enough to deliver the optimal solution of the MAXCUT problem. The dimensions of the corresponding SDP problems (27) (for relaxation order two) are shown in Table 2 (right).

Table 2. Problems `SNL-<n>` and `MAXCUT-<n>`, number of variables n, size of the LMI constraint m and number of linear constraints.

problem	n	m	lin. constr.
SNL-16	1677	130	3348
SNL-32	6818	258	13630
SNL-48	14886	386	29766
SNL-64	26767	514	53528
SNL-80	40840	642	81674
SNL-96	58956	770	117906
SNL-112	82137	898	164268
SNL-128	109464	1026	218922

problem	n	m	lin. constr.
MAXCUT-20	22366	211	12390
MAXCUT-25	53301	326	30550
MAXCUT-30	108811	466	63860
MAXCUT-35	199396	631	119070
MAXCUT-40	337431	821	204180
MAXCUT-45	537166	1036	328440
MAXCUT-50	814726	1276	502350

9.2. Loraine setting

The algorithms presented in this article were implemented in MATLAB and Julia codes. The codes use DIMACS stopping criteria [26] with $\varepsilon_{\text{DIMACS}} = 10^{-5}$.

The stopping parameter for the CG method ε_{CG} in Algorithm 1 has been set and updated as follows: we start with $\varepsilon_{\text{CG}} = 0.01$ and multiply it after every major iteration by 0.5, until it reaches the value 10^{-6}; then we continue with this value till convergence of the optimization algorithm.

Unless stated otherwise, we use the following hybrid preconditioner called H_{hyb}: we start with H_β (14) until the criterion below is satisfied and then switch to H_α (12):

$$N_{\text{cg\_iter}} > kp\sqrt{n}/10 \;\;\&\;\; N_{\text{iter}} > \sqrt{n}/60\,.$$

Here N_{iter} is the current IP iteration and $N_{\text{cg\_iter}}$ the number of CG steps needed to solve the corrector equation in this iteration. This criterion is, obviously, purely heuristic. We further choose the value of τ in H_α and H_β as

$$\tau = \lambda_1(W_i) + 0.5\,\text{mean}(\lambda_1(W_i), \ldots, \lambda_{m-k_i}(W_i))\,.$$

To solve the problems, we used the default setting of Loraine parameters as shown in Section 5, with the following exceptions, as explained in the respective paragraphs below:

```
SNL: tol_cg = 1e-1, eDIMACS = 1e-4, erank = 12, initpoint = 1
MAXCUT: eDIMACS = 1e-5
```

All problems were solved on an iMac desktop computer with 3.6 GHz 8-Core Intel Core i9 and 64 GB 2667 MHz DDR4 using MATLAB R2022b.

9.3. Other software

Later in this section, we present results obtained by Loraine, together with (sometimes brief) comparison with other SDP software. For this comparison, we use the solver MOSEK [27] which is also based on an interior-point algorithm. While this comparison may seem unfair, as MOSEK uses a direct solver to solve the linear systems, we believe that it gives a good perspective of the clear advantage of iterative solvers with good preconditioners, when available and when applicable. We further present a comparison with our MATLAB implementation of the ADMM method, based on [36].

For the truss problems, we also offer a comparison with SDPNAL+ [37], an implementation of the semi-smooth Newton-CG augmented Lagrangian method; and SDPLR [8] using a nonlinear programming algorithm based on low-rank factorization of the variables.

9.4. Numerical results

9.4.1. Truss topology problem

We now present results for all `tru` and `vib` problems using Loraine. These are shown in Table 3. The table presents the number of iterations of the respective algorithm and the total number of CG iterations using the hybrid preconditioner H_{hyb}. This is followed by CPU time spent in the optimization solver and time per iteration. Notice that Loraine solves two linear systems per iteration (predictor and corrector). For all problems we set the expected rank of the dual solutions $k = 1$.

Table 3. Loraine in `tru<n>` and `vib<n>` problems

problem	iter	CG iter	time	time/iter
tru3	16	122	0.01	0.00
tru5	21	190	0.08	0.00
tru7	27	236	0.24	0.01
tru9	31	333	0.65	0.02
tru11	36	370	1.6	0.04
tru13	45	500	4.5	0.10
tru15	52	882	11	0.22
tru17	53	980	24	0.45
tru19	64	1310	51	0.80
tru21	65	1325	84	1.29
tru23	72	2450	189	2.63
tru25	87	2148	317	3.64

problem	iter	CG iter	time	time/iter
vib3	20	209	0.04	0.00
vib5	31	411	0.21	0.01
vib7	39	501	0.67	0.02
vib9	47	663	2.2	0.05
vib11	59	995	6.1	0.10
vib13	69	1153	16	0.23
vib15	80	1480	37	0.46
vib17	94	1781	92	0.98
vib19	108	2116	189	1.75
vib21	120	2844	401	3.34
vib23	130	2720	718	5.52
vib25	143	3752	1321	9.24

The next Table 4 presents results of Loraine with *direct* solver and rank-one input data, MOSEK, SDPNAL+ and SDPLR; we again give the number of iterations of the solver, CPU time and (for Loraine and MOSEK) CPU time per one iteration. The data matrices for the truss problems are of rank one; see (15). While Loraine is much more efficient for these

Table 4. Loraine direct and other solvers in `tru<n>` problems

	Loraine direct			MOSEK			SDPNAL+		SDPLR	
problem	iter	time	time/iter	iter	time	time/iter	iter	time	iter	time
tru3	16	0.01	0.001	14	0.22	0.02	440	0.5	15	0.01
tru5	21	0.09	0.004	15	0.23	0.02	9674	29	18	1.7
tru7	27	0.46	0.02	17	0.78	0.05	6324	55	19	21
tru9	31	4.3	0.14	20	5.6	0.28	47579	468	20	179
tru11	36	31	0.86	26	44	1.69	maxit		22	2469
tru13	45	239	5.31	29	235	8.10			maxit	
tru15	52	1216	23.38	32	1079	33.72				
tru17	51	4583	89.86	37	4671	126.24				

Table 5. Loraine, MOSEK and ADMM in `SNL-<n>` problems

	Loraine			MOSEK		ADMM	
problem	iter	CG iter	time	iter	time	iter	time
SNL-16	21	395	1.4	13	1.0	6239	12
SNL-32	25	532	7.4	15	21	19343	362
SNL-48	27	676	21	17	150	14328	1050
SNL-64	27	705	40	16	611	81938	17554
SNL-80	28	1064	84	21	2052		>50000
SNL-96	29	969	120	27	6579	time	
SNL-112	30	1293	224	memory			
SNL-128	30	1316	299				

problems with an iterative solver, the goal of this comparison is to show that when handling the rank-one data efficiently, we get a code that is (at least per one iteration) faster than other software even with a direct solver.

We only present results for problems `tru3`–`tru17`; for the larger problems MOSEK exceeded the available 64GB RAM. SDPNAL+ and SDPLR were used with default stopping tolerance. None of these codes can solve problems bigger than `tru11` before reaching the maximum number of iterations, either internal or global. While Loraine solutions satisfy DIMACS criteria with 10^{-5}, other codes often finish with lower precision. The ADMM is not included in this comparison as, even for the smallest problems, the method stagnated at a point far from the solution.

In Figure 4-left we give a comparison of the CPU time per iteration for Loraine (iterative) and MOSEK. While MOSEK actually exceeds the theoretical complexity with coefficient 2.2, the complexity of Loraine is almost linear for these problems.

9.4.2. Sensor network localization problems

Recall from Section 7 that problems with noisy data lead to optimal solution Z^* of (23) with two outlying and a number of small, nonzero eigenvalues. Selecting $k = 2$ therefore did not lead to an efficient preconditioner and we had to increase the estimate of the rank of Z^*. For the problems we solved, $k = 12$ turned out to be a good compromise between the quality of the preconditioner and the CPU time needed to apply it. Due to ill-conditioning of the problems, we relaxed the stopping criterion to `eDIMACS=1e-4`; the same criterion was used for the ADMM code.

Table 5 presents the results for Loraine, MOSEK and ADMM. While ADMM turned out to be more successful than for the truss problems, it is still much slower than the two IP codes. In this table, the number of CG iterations sums up the iterations with preconditioners H_β and H_α. For instance, in problem `SNL-64`, the average number of CG iterations with H_α was 10 and the maximal number (in the last IP iteration) was 23. In Figure 4-right we give a comparison of the CPU time per iteration for Loraine and MOSEK.

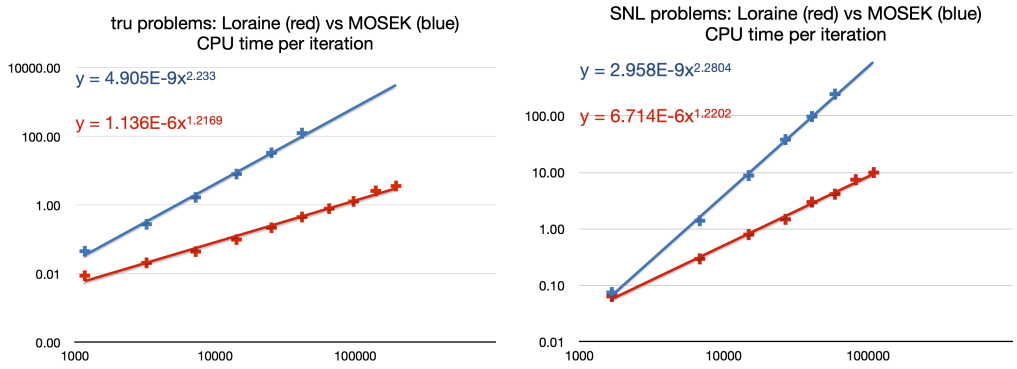

Figure 4. Loraine (red) and MOSEK (blue) in `tru*` problems (left) and `SNL*` problems (right). CPU time per iteration vs number of variables in log-log scale.

```
*** Loraine v0.1 ***
Number of variables: 40839      Preconditioner     :     4
Matrix size(s)     :   642      Expected rank      :    12
Linear constraints : 81674
it        obj         error    cg_iter time/iter
1   4.37357010e+02  3.32e+05       4     0.34
2   4.79694156e+02  7.79e+04       4     0.40
3   4.75844194e+02  8.44e+03      38     0.81
...
9   1.64854663e+00  7.24e+02      45     0.71
10  -4.55049809e+00  1.73e+02      69     0.91
11  -1.14083700e+01  2.25e+01     112     1.23
Switching to preconditioner 1
12  -1.95288379e+01  8.02e+00      14     2.74
13  -2.43383931e+01  4.02e+00      13     2.65
14  -2.97196245e+01  2.32e+00      14     2.70
...
25  -4.17128514e+01  1.71e-03      40     4.61
26  -4.17274123e+01  6.84e-04      58     5.95
27  -4.17345508e+01  2.33e-04      81     7.67
28  -4.17372049e+01  7.21e-05     111     9.71
*** Total CG iterations:     1064
*** Total CPU time:      80.70 seconds
```

Figure 5. Problem `SNL-80` solved using the hybrid preconditioner.

For these problems, we also demonstrate the usefulness of the hybrid preconditioner (see Sections 4 and 9.2). Figure 5 shows the iteration log for problem `SNL-80`; the first IP iterations are solved using H_β; once it requires too many iterations, we switch to H_α.

9.4.3. MAXCUT problems

The results obtained for the MAXCUT problems are summarized in Table 6. These problems are relatively “simple": the ADMM method is rather efficient now (this is well-known from other studies) and MOSEK converges in a very small number of iterations. However, the sheer size prevents MOSEK with a direct solver to solve larger problems, while ADMM is still about three times slower than Loraine. Moreover, the ADMM behaviour is rather unpredictable, regarding the number of iterations.

Table 6. Loraine, MOSEK and ADMM in `MAXCUT-<n>` problems

	Loraine			MOSEK		ADMM	
problem	iter	CG iter	time	iter	time	iter	time
MAXCUT-20	18	559	3.8	6	9	3486	15
MAXCUT-25	20	728	11	7	78	4314	45
MAXCUT-30	21	1032	28	9	607	4837	105
MAXCUT-35	23	2183	96	9	2911	3030	126
MAXCUT-40	27	2275	186	memory		1280	92
MAXCUT-45	25	2521	335			7976	1034
MAXCUT-50	24	2540	528			8783	1832

10. Conclusions

We have demonstrated the efficiency of Loraine for three classes of well-known SDP problems. While we tested our software for other problems from standard libraries, we do not report it here for one of the following reasons:

(1) The available collections, such as SDPLIB [7], are not representative enough in the sense that they do not include enough problems of the same type. We did not want to pick up single problems.
(2) Loraine mainly benefits from the use of an iterative solver for the linear system using a specific preconditioner targeted to problems with low-rank solutions. For many such problems, however, no preconditioner is needed for the convergence of the iterative method; these "easy" problems were not included in our testing. This is the case of matrix completion problems [3, 39] or some problems arising in the relaxation of combinatorial optimization. It is *not* the case of problems from Section 9; for those, the preconditioner is essential for convergence.
(3) The problems do not satisfy our dimensional assumption $n \gg m$. For these problems, the preconditioner is still very efficient (assuming low-rank solutions) but the use of iterative methods does not bring any significant advantage to direct solvers. This is the case of many problems found in SDPLIB.

Funding

This work has been supported by European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Actions, grant agreement 813211 (POEMA). The second author has been partly supported by the Czech Science Foundation through project No. 22-15524S.

References

[1] W. Achtziger and M. Kočvara, *On the maximization of the fundamental eigenvalue in topology optimization*, Structural and Multidisciplinary Optimization 34 (2007), pp. 181–195.
[2] W. Achtziger and M. Kočvara, *Structural topology optimization with eigenvalues*, SIAM Journal on Optimization 18 (2008), pp. 1129–1164.
[3] S. Bellavia, J. Gondzio, and M. Porcelli, *A relaxed interior point method for low-rank semidefinite programming problems with applications to matrix completion*, Journal of Scientific Computing 89 (2021), pp. 1–36.
[4] S.J. Benson, Y. Ye, and X. Zhang, *Solving large-scale sparse semidefinite programs for combinatorial optimization*, SIAM Journal on Optimization 10 (2000), pp. 443–461.
[5] P. Biswas, T.C. Liang, K.C. Toh, Y. Ye, and T.C. Wang, *Semidefinite programming approaches for*

sensor network localization with noisy distance measurements, IEEE transactions on automation science and engineering 3 (2006), pp. 360–371.

[6] P. Biswas and Y. Ye, *Semidefinite programming for ad hoc wireless sensor network localization*, in *Proceedings of the 3rd international symposium on Information processing in sensor networks*. 2004, pp. 46–54.

[7] B. Borchers, *SDPLIB 1.2, a library of semidefinite programming test problems*, Optimization Methods and Software 11 (1999), pp. 683–690.

[8] S. Burer and R. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Mathematical Programming 95 (2003), pp. 329–357.

[9] S. Burer and R. Monteiro, *Local minima and convergence in low-rank semidefinite programming*, Mathematical Programming 103 (2005), pp. 427–444.

[10] E.J. Candès and T. Tao, *The power of convex relaxation: Near-optimal matrix completion*, IEEE Transactions on Information Theory 56 (2010), pp. 2053–2080.

[11] H. Fawzi, J. Saunderson, and P.A. Parrilo, *Sparse sums of squares on finite abelian groups and improved semidefinite lifts*, Mathematical Programming 160 (2016), pp. 149–191.

[12] M.X. Goemans and D.P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of the ACM (JACM) 42 (1995), pp. 1115–1145.

[13] S. Habibi, M. Kočvara, and M. Stingl, *An interior-point method for Lasserre relaxations of unconstrained binary quadratic optimization problems*, preprint (2022).

[14] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, SIAM Journal on Optimization 10 (2000), pp. 673–696.

[15] R.A. Horn and C.R. Johnson, *Topics in matrix analysis*, Cambridge University Press, 1991.

[16] F. Jarre, M. Kočvara, and J. Zowe, *Optimal truss design by interior-point methods*, SIAM Journal on Optimization 8 (1998), pp. 1084–1107.

[17] M. Journée, F. Bach, P.A. Absil, and R. Sepulchre, *Low-rank optimization on the cone of positive semidefinite matrices*, SIAM Journal on Optimization 20 (2010), pp. 2327–2351.

[18] 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 (2011), pp. 33–68.

[19] M. Kočvara, *Decomposition of arrow type positive semidefinite matrices with application to topology optimization*, Mathematical Programming (2020), pp. 1–30.

[20] M. Kočvara and M. Stingl, *On the solution of large-scale SDP problems by the modified barrier method using iterative solvers*, Mathematical Programming 109 (2007), pp. 413–444.

[21] N. Krislock and H. Wolkowicz, *Explicit sensor network localization using semidefinite representations and facial reductions*, SIAM Journal on Optimization 20 (2010), pp. 2679–2708.

[22] J.B. Lasserre, *Moments, positive polynomials and their applications*, World Scientific, 2009.

[23] M. Laurent, *Lower bound for the number of iterations in semidefinite hierarchies for the cut polytope*, Mathematics of operations research 28 (2003), pp. 871–883.

[24] R. Madani, A. Kalbat, and J. Lavaei, *ADMM for sparse semidefinite programming with applications to optimal power flow problem*, in *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 5932–5939.

[25] J. Malick, J. Povh, F. Rendl, and A. Wiegele, *Regularization methods for semidefinite programming*, SIAM Journal on Optimization 20 (2009), pp. 336–356.

[26] H.D. Mittelmann, *An independent benchmarking of SDP and SOCP solvers*, Mathematical Programming 95 (2003), pp. 407–430.

[27] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* (2019). Available at http://docs.mosek.com/9.0/toolbox/index.html.

[28] D.E. Oliveira, H. Wolkowicz, and Y. Xu, *ADMM for the SDP relaxation of the QAP*, Mathematical Programming Computation 10 (2018), pp. 631–658.

[29] F. Permenter and P. Parrilo, *Partial facial reduction: simplified, equivalent SDPs via approximations of the PSD cone*, Mathematical Programming 171 (2018), pp. 1–54.

[30] A.M.C. So and Y. Ye, *Theory of semidefinite programming for sensor network localization*, Mathematical Programming 109 (2007), pp. 367–384.

[31] M. Stingl, M. Kočvara, and G. Leugering, *A sequential convex semidefinite programming algorithm with an application to multiple-load free material optimization*, SIAM Journal on Optimization 20 (2009), pp. 130–155.

[32] M.J. Todd, K.C. Toh, and R.H. Tütüncü, *On the Nesterov–Todd direction in semidefinite programming*, SIAM Journal on Optimization 8 (1998), pp. 769–796.

[33] K.C. Toh, *Solving large scale semidefinite programs via an iterative solver on the augmented systems*, SIAM Journal on Optimization 14 (2004), pp. 670–698.

[34] K.C. Toh and M. Kojima, *Solving some large scale semidefinite programs via the conjugate residual method*, SIAM Journal on Optimization 12 (2002), pp. 669–691.

[35] Z. Wang, S. Zheng, Y. Ye, and S. Boyd, *Further relaxations of the semidefinite programming approach to sensor network localization*, SIAM Journal on Optimization 19 (2008), pp. 655–673.

[36] Z. Wen, D. Goldfarb, and W. Yin, *Alternating direction augmented lagrangian methods for semidefinite programming*, Mathematical Programming Computation 2 (2010), pp. 203–230.

[37] L. Yang, D. Sun, and K.C. Toh, *SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints*, Mathematical Programming Computation 7 (2015), pp. 331–366.

[38] A. Yurtsever, J.A. Tropp, O. Fercoq, M. Udell, and V. Cevher, *Scalable semidefinite programming*, SIAM Journal on Mathematics of Data Science 3 (2021), pp. 171–200.

[39] R.Y. Zhang and J. Lavaei, *Modified interior-point method for large-and-sparse low-rank semidefinite programs*, 2017 IEEE 56th Annual Conference on Decision and Control (CDC) (2017). Available at http://dx.doi.org/10.1109/CDC.2017.8264510.

[40] Y. Zhang, *On extending some primal–dual interior-point algorithms from linear programming to semidefinite programming*, SIAM Journal on Optimization 8 (1998), pp. 365–386.

[41] X.Y. Zhao, D. Sun, and K.C. Toh, *A Newton-CG augmented lagrangian method for semidefinite programming*, SIAM Journal on Optimization 20 (2010), pp. 1737–1765.

[42] Y. Zhu, G. Pataki, and Q. Tran-Dinh, *Sieve-SDP: a simple facial reduction algorithm to preprocess semidefinite programs*, Mathematical Programming Computation 11 (2019), pp. 503–586.