Sieve-SDP: a simple facial reduction algorithm to preprocess semidefinite programs

Yuzixuan Zhu[†] Gábor Pataki[†*] Quoc Tran-Dinh[†]

October 24, 2017

Abstract

We introduce Sieve-SDP, a simple algorithm to preprocess semidefinite programs (SDPs). Sieve-SDP belongs to the class of facial reduction algorithms. It inspects the constraints of the problem, deletes redundant rows and columns, and reduces the size of the variable matrix. It often detects infeasibility. It does not rely on any optimization solver: the only subroutine it needs is Cholesky factorization, hence it can be implemented in a few lines of code in machine precision. We present extensive computational results on several problem collections from the literature.

We also highlight an issue arising in SDPs with positive duality gap: on such problems SDP solvers may compute a "fake" solution with an arbitrarily small constraint violation, and arbitrarily small duality gap.

Key words: Semidefinite programming; preprocessing; strict feasibility; strong duality; facial reduction

MSC 2010 subject classification: Primary: 90-08, 90C22; secondary: 90C25, 90C06

1 Introduction and the preprocessing algorithm

Consider a semidefinite programming problem (SDP) in the form

$$\begin{array}{rrcl} \inf & C \bullet X & & \\ \text{s.t.} & A_i \bullet X & = & b_i \ (i = 1, \ldots, m) \\ & X & \succeq & 0 \end{array} \tag{P}$$

where the A_i and C are $n \times n$ symmetric matrices, the b_i scalars, $X \succeq 0$ means that X is in $\mathcal{S}^n_+$, the set of symmetric, positive semidefinite (psd) matrices, and the $\bullet$ inner product of symmetric matrices is the trace of their regular product.

SDPs are some of the most versatile, useful, and widespread optimization problems of the last three decades. They find applications in control theory, integer programming and combinatorial optimization, to name just a few areas. Several good solvers are available to solve SDPs: see for example [35, 31, 17, 16, 40, 19, 6, 7, 1], the last one, Mosek, is commercially available.

*Corresponding author

[†]Y. Zhu, G. Pataki and Q. Tran-Dinh are with the Department of Statistics and Operations Research, University of North Carolina at Chapel Hill.

Address: Hanes Hall, Chapel Hill, NC27599-3260. *Email: zyzx@live.unc.edu, gabor@unc.edu, quoctd@email.unc.edu*

SDPs – as all optimization problems – often have redundant variables and/or constraints. The redundancy we focus on is lack of *strict feasibility*, i.e., when there is no feasible positive definite X in (P). When (P) is not strictly feasible, the optimal value of (P) and of its dual may differ, and the latter may not be attained [1]. On such instances SDP solvers often struggle, or fail.

It is, of course, useful to detect lack of strict feasibility in a preprocessing stage. This paper provides a very simple preprocessing algorithm for SDPs, called Sieve-SDP, which belongs to the class of facial reduction algorithms [4, 38, 25, 34, 26, 20, 12, 13, 28] [2]. Sieve-SDP can detect lack of strict feasibility, reduce the size of the problem, and can be implemented in a few lines of code in machine precision.

To motivate our algorithm, let us consider an example:

Example 1. *The SDP instance (with an arbitrary objective function)*

$$\begin{aligned} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \bullet X &= 0 \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \bullet X &= -1 \\ X &\succeq 0, \end{aligned} \tag{1.1}$$

is infeasible. Indeed, suppose $X = (x_{ij})_{i,j=1}^3$ is feasible in (1.1). Then $x_{11} = 0$, hence the first row and column of X are zero by positive semidefiniteness, so the second constraint implies $x_{22} = -1$, which is a contradiction.

Note that if we replace -1 in the second constraint of (1.1) by a positive number, then (1.1) can be restated over the set of psd matrices with first row and column equal to zero. Thus, even if we do not detect infeasibility, such preprocessing is still useful.

Our algorithm Sieve-SDP repeats the Basic Step shown on Figure 1. Here we write $D \succ 0$ to denote that a symmetric matrix D is positive definite.

Basic Step

(1) Find if there is $i \in \{1, \ldots, m\}$ such that the ith constraint, after permuting rows and columns, and possibly multiplying both sides by -1, is of the form

$$\begin{pmatrix} D_i & 0 \\ 0 & 0 \end{pmatrix} \bullet X = b_i, \tag{1.2}$$

where $D_i \succ 0, b_i \leq 0$.

(2) If $b_i < 0$ STOP; (P) is infeasible.

(3) If $b_i = 0$, delete this constraint. Also delete all rows and columns in the other constraints that correspond to rows and columns of D_i.

Figure 1: The Basic Step of Sieve-SDP

[1] More precisely, when (P) *is* strictly feasible, *strong duality* holds between (P) and its dual, i.e., their values agree and the latter is attained. If (P) is strictly feasible, we also say that it satisfies *Slater's condition.*

[2] However, the correctness of Sieve-SDP can be explained without relying on the theory of facial reduction.

We emphasize that to find a constraint of the form (1.2), we may only permute rows and columns in a constraint and possibly multiply both sides of a constraint by -1. We do not take linear combinations of the constraints.

Example 1 continued *When we first execute the Basic Step on this example, we find the first constraint, delete it, and also delete the first row and column from the second constraint matrix. Next, we find the constraint*

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \bullet X = -1,$$

and declare that (1.1) is infeasible.

We call the algorithm Sieve-SDP, since shading the deleted rows and columns in the variable matrix X (and the A_i) we obtain a sieve like structure: see Figure 2.

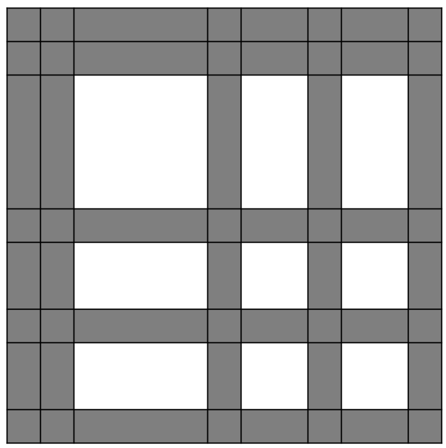

Figure 2: The sieve structure

Clearly, Sieve-SDP is easy to implement: it only needs an incomplete Cholesky factorization subroutine to check whether $D_i \succ 0$ holds in the Basic Step. We can, of course, replace Cholesky factorization by any other routine that can check positive definiteness.

A simple calculation shows that Sieve-SDP can be implemented using $O(\min\{m,n\}n^3m)$ arithmetic operations.

Sieve-SDP is heuristic in nature: it does not always detect infeasibility, or lack of strict feasibility. For example, it will not work on the instance of Example 1, if we apply a rotation $T^T()T$ to all A_i, where T is a random invertible matrix. However, Sieve-SDP still works well in practice.

Related work Our algorithm belongs to the family of *facial reduction algorithms*, which we now describe. When (P) is not strictly feasible, one can replace the constraint $X \in \mathcal{S}^n_+$ by

$$X \in F,$$

where F is a proper face of $\mathcal{S}^n_+$.[3] Since any such face can be written as (see e.g. [24])

$$F = V\mathcal{S}^r_+V^T, \tag{1.3}$$

[3] That is, $F \neq \mathcal{S}^n_+$, F is convex, and $X, Y \in \mathcal{S}^n_+$, $\frac{1}{2}(X+Y) \in F$ implies that X and Y are in F.

where $r < n$, V is an $n \times r$ matrix, the reduced problem can be restated over a smaller semidefinite cone. Facial reduction algorithms – for more general conic programs – originated in the papers [4, 3]. Later simplified, more easily implementable variants were given in [25, 38, 26], and in [34] for the SDP case. A recent, very concise version with a short proof of convergence is in [21].

Facial reduction algorithms, when applied to (P), find the face F by solving a sequence of SDP subproblems, which may be as hard to solve as (P). Thus one is led to seek simpler alternatives.

The idea of reducing SDPs by simply inspecting constraints appears in several papers. For example, [15] notes that if

$$A \bullet X = 0$$

is a constraint in (P) with $A \succeq 0$, then we can restrict X to belong to a face of the form (1.3) (where V spans the nullspace of A). A similar idea is used in [20] to reduce Euclidean Distance Matrix completion problems. For a rigorous derivation of the algorithm in [20] see [12]; the latter paper uses an intermediate step of analyzing the semidefinite completion problem (whether a partially defined matrix can be completed to be positive semidefinite). For followup work, see [13] on the noisy version of the same problem; and [32] for a more theoretical study.

A quite different approach to simplify facial reduction is described in [28]. The algorithms in [28] solve linear programs instead of SDPs to reduce the feasible set. Thus they do not find *all* reductions, but still simplify the SDPs in many cases. They are available as public domain codes, and we will compare them with Sieve-SDP in Section 2.

We finally mention two very accurate approaches to solve SDPs. The first is SDPA-GMP [16], which is based on the GMP package, and it computes solutions of (P) and of its dual using several hundred digits of accuracy. The second is the algorithm in [18], which computes a feasible solution of (P) (if there is one) in exact arithmetic. Although these methods cannot handle large SDPs, they can solve small ones very accurately.

Sieve-SDP differs in several aspects from the facial reduction algorithms put forth in the above papers:

- It needs only Cholesky factorization as a subroutine [4], and it does not rely on an optimization solver, like the algorithms in [28].
- It detects very simple redundancies, which are easy to explain even to a user not trained in optimization, and can help him/her to better formulate other problems.
- As soon as Sieve-SDP finds a reducing constraint, it deletes it, and it also deletes the corresponding rows and columns from the constraint matrices. Hence errors do not accumulate. Thus Sieve-SDP is as accurate as incomplete Cholesky factorization, which works in machine precision [33, Theorem 23.2].
- Sieve-SDP can also detect infeasibility.
- It is easy to run in a *safe mode* (explained in the next section) to even better safeguard against numerical errors.
- Finally, we present extensive computational results on general SDPs, which, to the best of our knowledge, are not yet available for such a simple algorithm.

The rest of the paper is organized as follows. In Section 2 we describe how we implemented Sieve-SDP, the computational setup, and the criteria for comparison with competing codes.

[4] Of course, it can use any routine to check positive definiteness of a matrix

In this section we also give a small SDP with a positive duality gap (in Example 2), and show how to construct a pair of primal-dual solutions with arbitrarily small constraint violation and arbitrarily small duality gap. This example shows that a solution with a *smaller* DIMACS error (see [23]) may be actually *less accurate*. We also show that such a less accurate solution is actually computed by Mosek, one of the leading SDP solvers.

The SDP of Example 2 is similar in spirit to the SDPs given in [39], which have known optimal solutions, but SDP solvers, other than the very accurate SDPA-GMP, fail on them. However, our SDP has a positive duality gap and it is smaller, thus our analysis is different and shorter than the analysis in [39]. Our SDP, as well as the instances in [39], show that some of the SDPs in the literature that are considered "solved" may actually have been solved very inaccurately.

In Section 3 we comment on the preprocessing results on some of the problems, mainly on those whose mathematically correct solution is known. We examine whether preprocessing helps to find the correct solution.

In Section 4 we summarize the preprocessing results, and conclude the paper.

We have three appendices. In Appendix A we give very detailed computational results on all the problems. In Appendix B we give the core Matlab code of Sieve-SDP, containing only about 60 lines. In Appendix C we provide the definition of the DIMACS errors for completeness.

2 Implementation, setup for computational testing, and the issue of positive duality gaps

Implementation We implemented our algorithm in Matlab, using the standard incomplete Cholesky factorization to check positive definiteness. Our implementation takes into account the block structure of the variable matrices in several SDPs in our problem collections.

Safe mode To safeguard against numerical errors we use a *safe mode.* We set

$$\epsilon := 2^{-52} \approx 2.2204 \cdot 10^{-16} = \text{the machine precision in Matlab.}$$

In the Basic Step, if we find a constraint of type (1.2), then, instead of checking $b_i < 0$ we check whether

$$b_i < -\sqrt{\epsilon}\max\{\, \|b\|_\infty, 1\} \text{ holds.}$$

If this test fails, then instead of checking $b_i = 0$ we check whether

$$b_i > -\epsilon\max\{\, \|b\|_\infty, 1\} \text{ holds.}$$

Internal format and input format Internally we store the A_i and C matrices as a cell array of sparse Matlab matrices. This internal format is fairly close to the input format of the leading SDP solvers Mosek, SDPT3, and SDPA, but less similar to the format used by Sedumi, which stores the A_i and C stretched out as vectors.

We store our datasets in the widely used Mosekopt format.

The comparison We compare our preprocessing algorithm with the algorithms proposed by Permenter and Parrilo in [28]. Their algorithms solve linear programming subproblems to reduce the size of an SDP. They can work either on the problem (P), which we call the *primal*; or on its dual:

$$\begin{array}{ll} \sup & \sum_{i=1}^m b_i y_i \\ s.t. & \sum_{i=1}^m y_i A_i \preceq C. \end{array} \tag{D}$$

They can use either diagonal, or diagonally dominant reductions (for details, see [28]).

Thus, there are four algorithms from [28] that we tested: pd1, pd2, dd1 and dd2. Here pd1 stands for primal diagonal; pd2 for primal diagonally dominant; dd1 for dual diagonal; and dd2 for dual diagonally dominant.

The datasets We tested Sieve-SDP and competing methods on 21 datasets:

- Twelve datasets from [28], with 68 problems in total. The original sources are [37, 10, 2, 14, 27, 11, 30, 29, 5, 39, 8, 36].

These datasets contain problems which are notoriously difficult for SDP solvers, and some are known to be not strictly feasible. Hence we added two more datasets to make our testing more comprehensive:

- A problem set kindly provided to us by Didier Henrion and Kim Chuan Toh, which we call the Henrion-Toh dataset. This dataset contains 98 problems.
- Eight datasets in the "moreSDPs" collection available from http://plato.asu.edu/ftp/sdp/. These contain 31 problems overall.

From the datasets of [28] we excluded only three problems. The first two are copos_5 and cprank_3, since they were too large to be solved by Mosek on our computer. The last one is the Example5 problem (originally from [10]), whose objective value is listed as unknown.

It is interesting that even in the Henrion-Toh and the moreSDPs datasets many SDPs can be reduced by some method. In the Henrion-Toh dataset, pd1, pd2, and Sieve-SDP all reduced 18 problems; dd1 and dd2 reduced none. In the "moreSDPs" collection, pd1, pd2, and Sieve-SDP reduced 8 problems; dd1 and dd2 reduced none.

The computational setup Both for Sieve-SDP and for the algorithms of [28] we use

- the Mosekopt input format;
- Mosek 8.0 (from now on, simply "Mosek") as an SDP solver: we solve the SDPs with Mosek before and after preprocessing.

We also use Mosek as an LP solver (to solve the LP subproblems) in the algorithms of [28].

We consider Mosek to be the best choice, since it is a reliable commercial SDP solver, and it is being actively developed.

We implemented Sieve-SDP in Matlab R2015a, and ran all codes on a MacBook Air with processor Intel Core i5 running at 2.7GHz, and 8GB of RAM.

By default the algorithms in [28] use Sedumi format as input format, Mosek as LP solver, and Sedumi as SDP solver. With our settings (using Mosek both as LP and as SDP solver) the algorithms of [28] work faster: though we must convert the data from Mosekopt format to Sedumi format, the required time is negligible. To be fair, in the detailed comparison tables of Appendix 3 we list separately the time spent on preprocessing, and the time spent on conversion.

Criteria for comparison We compare the preprocessing methods based on several criteria, and we capture information in *help codes*. In general, a positive help code means that a preprocessing method helps, and a negative code means that it hurts.

Our top comparison criteria are:

- Does the preprocessing reduce a problem? If yes, by how much?
- Does the preprocessing help detect infeasibility? To capture this information,
  - we set the help code as 1, if
    - Sieve-SDP detects infeasibility, or
    - Mosek does not detect infeasibility *before* preprocessing, but it does detect infeasibility *after* preprocessing,
  - we set the help code as -1, if
    - Mosek detects infeasibility *before* preprocessing, but does not detect infeasibilty *after* preprocessing.
- Does the preprocessing help recover a known optimal solution? Precisely, suppose the optimal solution value of a problem is known mathematically, but Mosek reports an incorrect solution before preprocessing. Does Mosek find the optimal solution after preprocessing?

Note that when Sieve-SDP received a 1 help code, this means that it detected infeasibility by itself. In contrast, when a code from [28] received a 1 help code, this means that Mosek detected the infeasibility.

The following criteria are also useful, but they must be taken with a grain of salt, as we illustrate in Example 2.

- Does the preprocessing improve numerical accuracy measured by the six DIMACS errors [23] [5]? To make our tables concise, we report the *maximum* of the absolute values of the DIMACS errors before and after preprocessing. We write

  $$\text{DIMACS}_{\text{before}} \text{ and } \text{DIMACS}_{\text{after}}$$

  for the largest DIMACS error before and after preprocessing, respectively, and
  - we set the help code as 2, if
    - help codes ± 1 do not apply, and

    $$\text{DIMACS}_{\text{before}} > 10^{-6} \text{ and } \frac{\text{DIMACS}_{\text{after}}}{\text{DIMACS}_{\text{before}}} < \frac{1}{10}.$$
  - we set the help code as -2 if
    - help codes ± 1 do not apply, and

    $$\text{DIMACS}_{\text{before}} > 10^{-6} \text{ and } \frac{\text{DIMACS}_{\text{after}}}{\text{DIMACS}_{\text{before}}} > 10,$$
- Does the preprocessing shift the objective function value? We write $\text{obj}_{\text{before}}$ and $\text{obj}_{\text{after}}$ for the average of the primal and dual objective values before and after preprocessing and
  - we set the help code as 3, if

[5] The description of the DIMACS errors is given in Appendix C.

* if help codes ± 1 and -2 do not apply, and

$$\frac{|\text{obj}_{\text{before}} - \text{obj}_{\text{after}}|}{1 + |\text{obj}_{\text{before}}|} > 10^{-6},$$

While the DIMACS errors are very natural (they measure constraint violation and duality gap) and are widely used, the following example shows that they do not always measure accurately how good a solution is. In fact, a *larger* DIMACS error may correspond to a *better* solution!

Example 2. *Consider the SDP*

$$\begin{array}{rrcl} \inf & x_{11} + x_{22} & & \\ s.t. & x_{11} & = & 0 \\ & x_{22} + 2x_{13} & = & 1 \\ & X = (x_{ij}) & \succeq & 0 \end{array} \tag{2.4}$$

with its dual

$$\begin{array}{rl} \sup & y_2 \\ s.t. & y_1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + y_2 \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \preceq \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \tag{2.5}$$

We claim that the duality gap between them is 1*. Indeed, let* X *be a feasible solution in (2.4). Since* $x_{11} = 0$, *the first row and column of* X *must be zero, hence*

$$X = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

is an optimal solution. In turn, in (2.5) we have $y_2 = 0$ *for all feasible* y.

Next, let $\epsilon > 0$ *and define* $M_\epsilon > 0$ *so that*

$$X_\epsilon := \begin{pmatrix} \epsilon & 0 & (1-\epsilon)/2 \\ 0 & \epsilon & 0 \\ (1-\epsilon)/2 & 0 & M_\epsilon \end{pmatrix}$$

is positive semidefinite. Then X_ϵ *is an approximate solution of (2.4) which violates only the first constraint (by* ϵ*) and has objective value* 2ϵ.

Do such "fake" solutions arise in practice? At first look it seems that they do not. If we feed the pair (2.4)-(2.5) to Mosek, it returns a solution with DIMACS errors

$$(0.5000, 0, 0.7071, 0, -5.5673 \cdot 10^{-9}, 5.9077 \cdot 10^{-17}).$$

Since the first and third errors are large, a user is unlikely to conclude that the problem has been "solved".

However, suppose we apply a rotation $T^T()T$ *to all matrices in (2.5) with*

$$T = \begin{pmatrix} 3 & 5 & -2 \\ 4 & 1 & 1 \\ -4 & -4 & 5 \end{pmatrix}.$$

Then the resulting primal-dual pair still has a duality gap of 1*. However, Mosek returns a solution with DIMACS errors*

$$(1.6093 \cdot 10^{-6}, 0, 5.2111 \cdot 10^{-9}, 3.287 \cdot 10^{-12}, -8.1484 \cdot 10^{-5}, 3.0511 \cdot 10^{-5}),$$

which may seem "essentially all zero" to a user.

It would be interesting to see whether "fake" solutions arise in *any* SDP with positive duality gap. We note that [9] show that in such SDPs an arbitrarily small perturbation of the *data* can change the duality gap by at least half.

We note that [10] also presented computational results on SDPs with positive duality gaps, and noted that Sedumi often gave an incorrect solution on such problems. However, [10] did not report the DIMACS errors, and did not give an ϵ-theoretical analysis.

3 Detailed comments on some problems

In this section we give detailed comments on the "Compact" dataset from [37]; on the "unBounded" dataset from [39]; on the "Example" and "RandGen" datasets from [10]; and on the "finance" dataset from [5].

In the first three datasets the exact optimal solution values of the problems are known, and we examine whether these values can be recovered by the preprocessing methods. We consider the solution value (primal or dual) "recovered" if 1) either help code 1 applies (see Section 2); or 2) after preprocessing Mosek computes an optimal solution with value within 10^{-6} of the exact value, and largest DIMACS error at most 10^{-6}.

We should really expect only the primal objective values to be the same before and after preprocessing by Sieve-SDP. Indeed, suppose that Sieve-SDP does not detect infeasibility. Then it only removes variables which are always zero anyway, i.e., it makes it explicit that all feasible solutions of (P) are in

$$F = \left\{ \begin{pmatrix} X_{11} & 0 \\ 0 & 0 \end{pmatrix} \right\} \big| X_{11} \in \mathcal{S}^r_+ \Bigg\} \text{ for some } 0 < r < n.$$

However, this means removing constraints from the dual (D), i.e., after preprocessing we only require the upper left $r \times r$ submatrix of $C - \sum_{i=1}^m y_i A_i$ to be positive semidefinite. In other words we require

$$C - \sum_{i=1}^{m} y_i A_i \in F^*,$$

where F^* is the dual cone of F. Thus the optimal value of (D) may change after preprocessing by Sieve-SDP.

Let us use the operator val() to denote the optimal value of an optimization problem, and write (P_{pre}) and (D_{pre}) for the primal and dual problems after preprocessing. Then by the previous argument

$$\text{val}(D) \le \text{val}(D_{\text{pre}}) \le \text{val}(P_{\text{pre}}) = \text{val}(P) \text{ holds,} \tag{3.6}$$

and all inequalities may be strict. For example, in the primal-dual pair (2.4) and (2.5) the corresponding optimal values are $0 < 1 = 1 = 1,$ respectively.

Algorithms pd1 and pd2 in [28] also reduce the primal problem. Thus we should expect inequality (3.6) to hold as well, if the subscript "pre" now refers to the problems obtained by preprocessing by pd1 and pd2. Algorithms dd1 and dd2 in [28] reduce the dual problem (D), thus we can expect the inequality

$$\mathrm{val}(D) = \mathrm{val}(D_{\mathrm{pre}}) \leq \mathrm{val}(P_{\mathrm{pre}}) \leq \mathrm{val}(P) \tag{3.7}$$

to hold.

The problems in the "finance" dataset from [5] are interesting, since these are the largest.

In all tables we use the following convention: among the reported objective values the first is the primal and the second is the dual.

3.1 "Compact" problems – 10 problems from [37]

These instances are *weakly infeasible*, i.e., the affine subspace

$$H = \{ X \mid A_i \bullet X = b_i \, (i = 1, \ldots, m) \}$$

does not intersect $\mathcal{S}^n_+$, but the distance of H to $\mathcal{S}^n_+$ is zero. Weakly infeasible SDPs are particularly challenging to SDP solvers. However, we refer to a recent algorithm in [18] which can detect (in)feasibility of small SDPs in exact arithmetic; and to [22] for an algorithm that is tailored to detect weak infeasibility.

On these problems pd1 and pd2 gave the same reduction results, while dd1 and dd2 did not reduce any of the problems. Pd1 and pd2 combined with Mosek correctly detected infeasibility in 9 out of the 10 problems, while Sieve-SDP correctly found infeasibility of all 10. (Since it found the primal infeasible, we did not compute a dual solution).

The results are in Table 1.

problem	correct obj	obj before (P,D)	after pd1/pd2	after dd1/dd2	after Sieve-SDP
CompactDim2R1	Infeas, $+\infty$	3.83e-06, 4.12e-06	0, 1	3.83e-06, 4.12e-06	Infeas, -
CompactDim2R2	Infeas, $+\infty$	4.87e-08, 5.24e-08	Infeas, $+\infty$	4.87e-08, 5.24e-08	Infeas, -
CompactDim2R3	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R4	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R5	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R6	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R7	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R8	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R9	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
CompactDim2R10	Infeas, $+\infty$	1.5, 1.5	Infeas, $+\infty$	1.5, 1.5	Infeas, -
correctness %	100%, 100%	0%, 0%	90%, 90%	0%, 0%	100%, -

Table 1: Results on the "CompactDim" dataset

Note that [21] provided a set of infeasible, and weakly infeasible SDPs, some of them classified "clean" and some of them classified "messy." In the "clean" instances the structure that proves infeasibility is apparent, so Sieve-SDP easily detects their infeasibility. In the "messy" instances the structure that proves infeasibility is hidden.

Indeed, in our testing Sieve-SDP, pd1 and pd2 detected infeasibility of all clean instances (Sieve-SDP does this without using an SDP solver). None of the codes reduced the messy instances. We did not include these instances in our test set, since we felt that it would not be fair to the other codes.

3.2 “Example” problems – 8 problems from [10]

The mathematically correct objective values are reported in [10] in table 12.1. (Note that in [10] our primal is considered a dual, and vice versa, so that table must be read accordingly.)

Table 2 shows the objective values before and after preprocessing. A “*” symbol in the row of Example4 means that Mosek computed an optimal value within 10^{-6} of the correct value, but the corresponding solution had largest DIMACS error above 10^{-6}.

We excluded Example5 from this table, since in table 12.1 in [10] the optimal value is not reported. For all other problems, except for Example9size20 and Example9size100, we manually verified the correctness of the optimal values in exact arithmetic.

problem	correct obj	obj before (P,D)	after pd1/pd2	after dd1/dd2	after Sieve-SDP
Example1	0, 0	0, 0	0, 0	0, 0	0, 0
Example2	1, 0	3.33e-01, 3.33e-01	1, 1	4.73e-15, 1.82e-14	1, 1
Example3	0, 0	3.33e-01, 3.33e-01	1.17e-07, 1.69e-07	4.73e-15, 1.82e-14	1.17e-07, 1.69e-07
Example4	Infeas, 0	0, 1.74e-07*	0, 1	0, 0	Infeas, -
Example6	1, 1	1, 1	1, 1	1, 1	1, 1
Example7	0, 0	0, 0	0, 0	0, 0	0, 0
Example9size20	Infeas, 0	0, 3.39e-01	0, 1	0, 0	Infeas, -
Example9size100	Infeas, 0	0, 3.43e-01	0, 1	0, 0	Infeas, -
correctness %	100%, 100%	38%, 38%	63%, 50%	50%, 100%	100%, 50%

Table 2: Results on the “Example” dataset

Note that the comparison in Table 2 is somewhat unfair to Sieve-SDP: if it found a problem infeasible, it did not compute a dual solution.

3.3 RandGen – 10 problems from [10]

These are instances taken from Table 12.1 in [10]. (Recall that that [10] considers our primal as the dual, and vice versa.) Among all problems in this dataset, only RandGen6, 7, 8 are reduced by pd2, dd2 or Sieve-SDP. Although all of these methods approximately recover the dual objective value on RandGen7 and RandGen8 (the DIMACS error is larger than 10^{-6} on RandGen6), none of them recover the primal objective value reported in [10].

Table 3 shows the details on these three problems.

problem	correct obj	obj before (P,D)	objs after pd2	objs after dd2	objs after Sieve-SDP
RandGen6	2.5869e+05, 0	3.95e-06, 3.24e-06	3.95e-06, 3.24e-06	1.68e-07, 1.26e-11	3.73e-06, 3.04e-06
RandGen7	168.5226, 0	9.42e-07, 4.22e-07	9.85e-07, 4.53e-07	2.65e-11, 4.69e-16	9.85e-07, 4.53e-07
RandGen8	4.1908, 0	5.41e-09, 2.44e-09	5.41e-09, 2.44e-09	2.15e-15, 2.78e-19	1.52e-09, 6.33e-10

Table 3: Results on the “RandGen” dataset

3.4 unBounded – 10 problems from [39]

In this dataset the mathematically correct values are 0 for both the primal and dual problems. Mosek returns wrong objective values for 6 out of 10 problems before preprocessing. Although Mosek finds an almost correct optimal solution value on problems 2,3 and 4, these solutions are inaccurate, as the DIMACS errors are of the order 10^{-1}.

In summary, 9 out of 10 problems in this dataset need preprocessing to obtain a reasonable solution.

Methods Sieve-SDP, pd1 and pd2 correct all objective values, as Table 4 shows.

It is interesting that [39] used SDPA-GMP [16], a very high precision SDP solver with 900 significant digits, to calculate the correct solutions of these instances.

problem	correct obj	obj before (P,D)	after pd1/pd2	after dd1/dd2	after Sieve-SDP
unboundDim1R1	0, 0	1.33e-09, -7.05e-10	1.33e-09, -7.05e-10	1.33e-09, -7.05e-10	0, 0
unboundDim1R2	0, 0	-9.26e-12*, -8.65e-12*	0, 0	-9.26e-12*, -8.65e-12*	0, 0
unboundDim1R3	0, 0	-4.85e-09*, -4.71e-09*	0, 0	-4.85e-09*, -4.71e-09*	0, 0
unboundDim1R4	0, 0	-2.58e-08*, -2.55e-08*	0, 0	-2.58e-08*, -2.55e-08*	0, 0
unboundDim1R5	0, 0	-1, -1	0, 0	-1, -1	0, 0
unboundDim1R6	0, 0	-1, -1	0, 0	-1, -1	0, 0
unboundDim1R7	0, 0	-1, -1	0, 0	-1, -1	0, 0
unboundDim1R8	0, 0	-1, -1	0, 0	-1, -1	0, 0
unboundDim1R9	0, 0	-1, -1	0, 0	-1, -1	0, 0
unboundDim1R10	0, 0	-1, -1	0, 0	-1, -1	0, 0
correct%	100%, 100%	10%, 10%	100%, 100%	10%, 10%	100%, 100%

Table 4: Results on the “unBoundDim” dataset

3.5 finance – 4 problems from [5]

There are four problems in this dataset: “leverage_limit”, “long_only”, “sector_neutral” and “unconstrained”. We report on these problems in detail, since these are the largest. They are also outliers: all methods, except for pd1, reduce all problems, but they do not reduce the solution time.

Table 5 shows the reduction achieved by the five methods: here n_sdp is the total size of the semidefinite constrained blocks; n_nonneg is the total number of noonnegative variables; n_free is the total number of free variables; and m is the total number of constraints.

While dd1 and dd2 significantly reduced the size of the SDP blocks, they added many free variables. Sieve-SDP reduced the size of SDP blocks without adding free variables, and it eliminated the most constraints.

	before	after pd1	after pd2	after dd1	after dd2	after Sieve-SDP
n_sdp	60400	60400	60280	36400	36400	56766
n_nonneg	51100	51100	51100	51100	51100	50673
n_free	0	0	0	2286000	2286000	0
m	251777	251777	249797	251777	251777	215210
help		-, -, -, -	3, 3, 3, 3	3, 3, 3, -2	3, 3, 3, -2	3, 3, 3, 3

Table 5: Results on the finance problems

Table 6 shows the time details for each method. Note that all methods which achieved reduction actually increased the total time including preprocessing and solving.

method	before	pd1	pd2	dd1	dd2	Sieve-SDP
preprocessing time	0.00	7.76	436.55	12.54	2073.87	817.90
conversion time	0.00	0.84	32.37	32.03	32.32	0.00
solving time under Mosek	844.21	844.21	682.75	856.41	872.71	691.47
total time	844.21	852.81	1151.67	900.98	2978.90	1509.37

Table 6: Time results on the "finance" problems

It is easy to reduce the time spent by Sieve-SDP by setting a limit on the maximum number of iterations it is allowed to perform. We do not report on experiments with such a setting, since there are only four problems on which Sieve-SDP increases the solution time, so we do not want to "overtune" our code.

3.6 Henrion-Toh dataset (98 problems)

This dataset was kindly provided to us by Didier Henrion and Kim Chuan Toh. The problems come mostly from polynomial optimization. Among these problems 18 are reduced by pd1, pd2, or Sieve-SDP, and 9 are helped or hurt (see Section 2 for the help codes). None are reduced by dd1 or dd2.

Table 7 shows the details on these 9 problems.

problem	pd1/pd2	Sieve-SDP
sedumi-fp23	3	3
sedumi-fp25	2, 3	2, 3
sedumi-fp32	3	3
sedumi-fp33		-2
sedumi-fp35	2, 3	2, 3
sedumi-fp49		3
sedumi-fp210		3
sedumi-fp410		3
sedumi-l4	1	1

Table 7: Help codes on 9 problems in the Henrion-Toh dataset

In Figure 3 we illustrate how Sieve-SDP works on the instance "sedumi-fp32.mat": we show the sparsity structure of the constraints of the original problem (on the left), and after we applied Sieve-SDP (on the right). Each row corresponds to an A_i matrix stretched out as a vector. A blue dot corresponds to a positive entry, a red dot corresponds to a negative entry, and white areas correspond to zero entries.

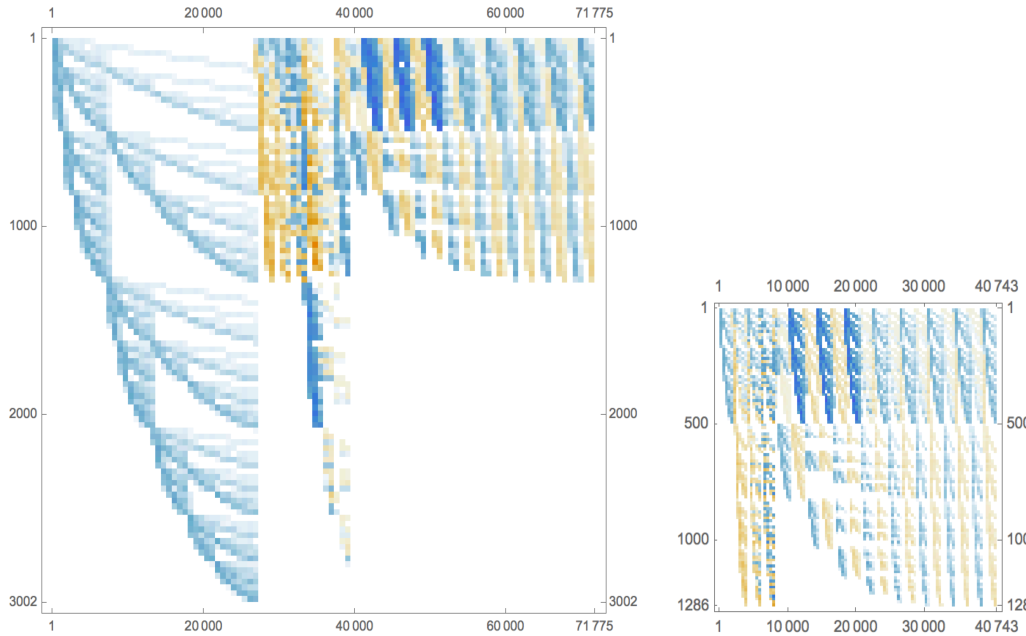

Figure 3: Instance "fp32-sedumi": size and sparsity before and after preprocessing

On problem "sedumi-fp33" Sieve-SDP hurts the DIMACS error (help code is -2). Since this is the only such instance, we looked at it in more detail. The worst DIMACS error (of a solution computed by Mosek) before Sieve-SDP is $3.36 \cdot 10^{-7}$, which is acceptable. After Sieve-SDP the worst error is about 0.0928, which is unacceptable.

We also solved this instance using the high accuracy SDP solver SDPA-GMP [16]. The DIMACS errors were

$$2.3497 \cdot 10^2, 0.0000, 1.8552 \cdot 10^1, 0.0000, -9.9999 \cdot 10^{-1}, 8.5173 \cdot 10^{-2}$$

before Sieve-SDP, and

$$3.4075 \cdot 10^2, 0.0000, 1.9636 \cdot 10^1, 0.0000, -9.9999 \cdot 10^{-1}, 6.1901 \cdot 10^{-1}$$

after Sieve-SDP.

Given the high accuracy of SDPA-GMP, it is safe to say that this problem cannot be accurately solved by current fast SDP solvers, and the *worse* DIMACS error after Sieve-SDP alerts the user to this fact: this problem may actually have a positive duality gap (cf. Example 2).

It is easy to see that by slightly perturbing this instance we get a primal-dual pair with zero duality gap. Hence it is natural to ask whether there are SDPs with a "hard" duality gap, i.e., SDPs with a duality gap, whose correct formulation has all integers.

4 Summary

We now give an overall comparison of all codes in Tables 8 and 9.

Table 8 shows how many problems were reduced, by how much, and the help codes.

	Reduction					Help codes					MM
	# reduced	red on n	red on m	extra free vars	nnz	1	-1	2	-2	3	
before	-	-	-	-	96903571	-	-	-	-	-	-
pd1	54	1.57%	6.90%	0	96063810	12	0	8	0	13	0
pd2	75	1.75%	7.94%	0	96021834	12	0	11	0	17	6
dd1	14	11.02%	0.00%	2293495	96850359	0	2	3	1	5	0
dd2	21	11.08%	0.00%	2315849	95425200	0	2	6	1	6	4
Sieve-SDP	61	3.49%	13.63%	0	90825196	14	0	8	1	20	0

Table 8: Reduction and help codes on all 197 problems

The second column shows how many problems were reduced. The third through sixth columns show "by how much" the problems were reduced. To explain these columns, let us fix an SDP in the primal form (P) with potentially several semidefinite block variables (some of which may be of order 1, i.e., nonnegative variables).

Let n_{before} and n_{after} be the total size of the semidefinite blocks before and after reduction. We define the reduction rate on n as

$$\frac{\sum n_{\text{before}} - \sum n_{\text{before}}}{\sum n_{\text{before}}},$$

where the summation is taken over all 197 problems.

Similarly, let m_{before} and m_{after} be the number of constraints in a problem before and after reduction. We define the reduction rate on m as

$$\frac{\sum m_{\text{before}} - \sum m_{\text{before}}}{\sum m_{\text{before}}},$$

where the summation is taken over all 197 problems.

Codes dd1 and dd2 add free variables, and the fifth column in table 8 shows how many.

The sixth column "nnz" shows the total number of nonzeros in the constraint matrices.

The last column "MM" shows how many times a code ran out of memory (or crashed): this happened with pd2 6 times and with dd2 4 times. To make sure that we report the results in a fair manner, we reran these codes on a machine with 24 GB RAM, and the results were the same.

Table 9 shows the running times. The first column shows the preprocessing time and the second shows the solution time by Mosek after preprocessing.

The last column "Ratio" gives an intuitive idea about how fast the preprocessors are:

$$\text{Ratio} = \frac{\text{preprocessing time}}{\text{solution time without preprocessing}}$$

	preprocessing	solving	Ratio
before	–	111940.02	–
pd1	695.60	111251.44	0.62%
pd2	8607.75	110546.09	7.69%
dd1	488.03	111901.02	0.44%
dd2	12069.64	111943.52	10.78%
our	896.03	111113.21	0.80%

Table 9: Preprocessing times

Given these tables we can summarize the findings of the paper. In all aspects Sieve-SDP is competitive with the other codes. In detail:

- It is second best considering the number of problems reduced.
- It is competitive considering "reduction on n"; methods dd1 and dd2 won, but those methods added a large number of free variables.
- It is competitive timewise: considering the time spent on preprocessing, we have

  $$\mathrm{dd1} < \mathrm{pd1} < Sieve\text{-}SDP \ll \mathrm{pd2} \ll \mathrm{dd2}.$$

  Notice, however, that dd1 reduced the smallest number of problems. Also note that Sieve-SDP spent 817.90s (91% of our total preprocessing time) on 4 big problems from the "finance" dataset, and only spent 78.13s on the remaining 193 problems.

  In fact, Sieve-SDP spent less than a second on 178 problems out of the 197; and it spent more than a minute only on 4 problems, the large finance problems.

  It spent less than one percent of the time on preprocessing, than it took for Mosek to solve the problems.
- It is competitive in recovering known optimal solutions; see Tables 1, 2, and 3.

In several aspects Sieve-SDP seems to be the best:

- It is best considering "reduction on m" and in reducing the number of nonzeros.
- It needs very little additional memory, precisely $O(nm)$. For details, and the Matlab code, see Appendix B.
- It is best in detecting infeasibility (this is indicated by help code 1). It is important that Sieve-SDP detects infeasibility without using any optimization solver, whereas the other methods rely on Mosek.
- It is very accurate and stable: it is as accurate as Cholesky factorization, which works in machine precision. Sieve-SDP is also easily implemented in a *safe mode*.
- It is the simplest: the core Matlab code consists of only 60 lines. Thus it seems easy to parallelize it, in particular, it is also likely to work very well using a machine with a GPU.

https://github.com/quoctd/SieveSDP

A Detailed results

In this section we give very detailed computational results on all problems.

The tuple $f; l; s;$ describes the size of variables of the problem, where

- the number f stands for the number of *free* variables;
- the number l stands for the number of *linear nonnegative* variables;
- the number s stands for the size of the semidefinite variable block, possibly with multiplicity.

For example, $3; 5; 6$ means that a problem has 3 free variables; 5 linear nonnegative variables; and a semidefinite matrix variable of order 6. The tuple $3; 5; 6, 5_3$ means that a problem has 3 free variables; 5 linear nonnegative variables; and four semidefinite matrix variable blocks which are of order $6, 5, 5, 5,$ respectively. The number m stands for the number of constraints.

In the column "red" we have 1 if the problem was reduced; 0 if it was not reduced; and "Infeas" if Sieve-SDP found it infeasible. The number t_{prep} is the time spent on preprocessing; the number t_{conv} is the time spent on converting from Mosek format to Sedumi format (for the codes pd1, pd2, dd1, dd2).

In the column "Infeas" we have a 1 if Mosek detects infeasibility after preprocessing, and 0 if it does not. The column obj(P, D) stands for the objective values (primal and dual, respectively). The column DIMACS contains the absolute value of the DIMACS error, whose absolute value is *largest.*

As before, MM means that a code ran out of memory, or crashed.

A.1 Detailed results on the "Permenter-Parrilo" problems

Note that this problem collection has 68 problems. From these 59 problems were reduced by at least one of the five methods.

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
1	CompactDim2R1	before	0; 3; 3	5				0	3.83e-06, 4.12e-06	7.07e-01	2.60	
		pd1	0; 3; 1	3	1	0.31	0.02	1	0.00e+00, 1.00e+00	7.07e-01	0.50	1
		pd2	0; 3; 1	3	1	0.09	0.00	1	0.00e+00, 1.00e+00	7.07e-01	0.41	1
		dd1			0	0.04	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			Infeas	0.04					0.00	1
2	CompactDim2R2	before	0; 0; 6, 3_3	14				0	4.87e-08, 5.24e-08	7.07e-01	1.19	
		pd1	0; 0; 1_3	2	1	0.07	0.00	1	Infeas, $+\infty$	-	0.55	1
		pd2	0; 0; 1_3	2	1	0.07	0.00	1	Infeas, $+\infty$	-	0.53	1
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			Infeas	0.00					0.00	1
3	CompactDim2R3	before	0; 0; 10, 6_3	27				0	1.50e+00, 1.50e+00	1.15e-07	0.71	
		pd1	0; 0; 1_3	2	1	0.11	0.00	1	Infeas, $+\infty$	-	0.51	1
		pd2	0; 0; 1_3	2	1	0.10	0.00	1	Infeas, $+\infty$	-	0.58	1
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			Infeas	0.01					0.00	1
4	CompactDim2R4	before	0; 0; 15, 10_3	44				0	1.50e+00, 1.50e+00	1.13e-07	1.79	
		pd1	0; 0; 1_3	2	1	0.19	0.00	1	Infeas, $+\infty$	-	1.57	1
		pd2	0; 0; 1_3	2	1	0.22	0.00	1	Infeas, $+\infty$	-	1.69	1
		dd1			0	0.02	0.00					
		dd2			0	0.05	0.00					
		Sieve-SDP			Infeas	0.02					0.00	1
5	CompactDim2R5	before	0; 0; 21, 15_3	65				0	1.50e+00, 1.50e+00	1.83e-07	1.91	
		pd1	0; 0; 1_3	2	1	0.26	0.00	1	Infeas, $+\infty$	-	1.57	1
		pd2	0; 0; 1_3	2	1	0.27	0.00	1	Infeas, $+\infty$	-	1.50	1
		dd1			0	0.02	0.00					
		dd2			0	0.05	0.00					
		Sieve-SDP			Infeas	0.03					0.00	1
6	CompactDim2R6	before	0; 0; 28, 21_3	90				0	1.50e+00, 1.50e+00	2.70e-07	1.86	
		pd1	0; 0; 1_3	2	1	0.32	0.00	1	Infeas, $+\infty$	-	1.55	1
		pd2	0; 0; 1_3	2	1	0.39	0.00	1	Infeas, $+\infty$	-	1.57	1
		dd1			0	0.03	0.00					
		dd2			0	0.07	0.00					
		Sieve-SDP			Infeas	0.04					0.00	1
7	CompactDim2R7	before	0; 0; 36, 28_3	119				0	1.50e+00, 1.50e+00	3.66e-07	1.90	
		pd1	0; 0; 1_3	2	1	0.41	0.00	1	Infeas, $+\infty$	-	1.54	1
		pd2	0; 0; 1_3	2	1	0.64	0.00	1	Infeas, $+\infty$	-	1.57	1
		dd1			0	0.04	0.00					
		dd2			0	0.07	0.00					
		Sieve-SDP			Infeas	0.06					0.00	1
8	CompactDim2R8	before	0; 0; 45, 36_3	152				0	1.50e+00, 1.50e+00	5.61e-07	1.92	
		pd1	0; 0; 1_3	2	1	0.57	0.00	1	Infeas, $+\infty$	-	1.55	1
		pd2	0; 0; 1_3	2	1	0.91	0.00	1	Infeas, $+\infty$	-	1.60	1
		dd1			0	0.04	0.00					
		dd2			0	0.08	0.00					
		Sieve-SDP			Infeas	0.08					0.00	1
9	CompactDim2R9	before	0; 0; 55, 45_3	189				0	1.50e+00, 1.50e+00	6.27e-07	1.97	
		pd1	0; 0; 1_3	2	1	0.84	0.00	1	Infeas, $+\infty$	-	1.53	1
		pd2	0; 0; 1_3	2	1	1.44	0.00	1	Infeas, $+\infty$	-	1.52	1
		dd1			0	0.03	0.00					
		dd2			0	0.15	0.00					
		Sieve-SDP			Infeas	0.12					0.00	1
10	CompactDim2R10	before	0; 0; 66, 55_3	230				0	1.50e+00, 1.50e+00	5.17e-07	1.21	
		pd1	0; 0; 1_3	2	1	0.99	0.02	1	Infeas, $+\infty$	-	0.70	1
		pd2	0; 0; 1_3	2	1	2.21	0.00	1	Infeas, $+\infty$	-	0.52	1
		dd1			0	0.03	0.00					
		dd2			0	0.15	0.00					
		Sieve-SDP			Infeas	0.15					0.00	1
11	Example1	before	0; 0; 3	2				0	0.00e+00, 0.00e+00	0.00e+00	1.56	
		pd1	0; 0; 2	1	1	0.07	0.01	0	0.00e+00, 0.00e+00	0.00e+00	1.50	
		pd2	0; 0; 2	1	1	0.05	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.55	
		dd1	5; 0; 1	2	1	0.07	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.57	
		dd2	5; 0; 1	2	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.54	
		Sieve-SDP	0; 0; 2	1	1	0.01		0	0.00e+00, 0.00e+00	0.00e+00	1.55	
12	Example2	before	0; 0; 3	2				0	3.33e-01, 3.33e-01	5.05e-02	1.59	
		pd1	0; 0; 2	1	1	0.06	0.01	0	1.00e+00, 1.00e+00	0.00e+00	1.54	2,3
		pd2	0; 0; 2	1	1	0.04	0.00	0	1.00e+00, 1.00e+00	0.00e+00	1.54	2,3
		dd1	3; 0; 2	2	1	0.04	0.00	0	4.73e-15, 1.82e-14	2.75e-14	1.67	2,3
		dd2	3; 0; 2	2	1	0.05	0.00	0	4.73e-15, 1.82e-14	2.75e-14	1.67	2,3
		Sieve-SDP	0; 0; 2	1	1	0.00		0	1.00e+00, 1.00e+00	0.00e+00	1.65	2,3
13	Example3	before	0; 0; 3	4				0	3.33e-01, 3.33e-01	6.90e-02	1.61	
		pd1	0; 0; 2	1	1	0.04	0.01	0	1.17e-07, 1.69e-07	5.14e-08	2.46	2,3
		pd2	0; 0; 2	1	1	0.06	0.01	0	1.17e-07, 1.69e-07	5.14e-08	2.40	2,3
		dd1	3; 0; 2	4	1	0.04	0.01	0	4.73e-15, 1.82e-14	2.75e-14	1.67	2,3
		dd2	3; 0; 2	4	1	0.05	0.00	0	4.73e-15, 1.82e-14	2.75e-14	1.63	2,3
		Sieve-SDP	0; 0; 2	1	1	0.01		0	1.17e-07, 1.69e-07	5.14e-08	2.47	2,3

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
14	Example4	before	0; 0; 3	3				0	0.00e+00, 1.74e-07	5.00e-01	2.96	
		pd1	0; 0; 1	1	1	0.06	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.17	1
		pd2	0; 0; 1	1	1	0.07	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.21	1
		dd1	5; 0; 1	3	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.60	2
		dd2	5; 0; 1	3	1	0.07	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.56	2
		Sieve-SDP			Infeas	0.00					0.00	1
15	Example6	before	0; 0; 8	8				0	1.00e+00, 1.00e+00	1.95e-08	1.84	
		pd1	0; 0; 5	4	1	0.07	0.00	0	1.00e+00, 1.00e+00	0.00e+00	1.58	
		pd2	0; 0; 5	4	1	0.05	0.00	0	1.00e+00, 1.00e+00	0.00e+00	1.54	
		dd1	26; 0; 4	8	1	0.04	0.00	0	1.00e+00, 1.00e+00	9.75e-09	1.63	
		dd2	26; 0; 4	8	1	0.05	0.00	0	1.00e+00, 1.00e+00	9.75e-09	1.60	
		Sieve-SDP	0; 0; 5	4	1	0.00		0	1.00e+00, 1.00e+00	0.00e+00	1.64	
16	Example7	before	0; 0; 5	3				0	0.00e+00, 0.00e+00	0.00e+00	1.55	
		pd1	0; 0; 4	2	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.51	
		pd2	0; 0; 4	2	1	0.05	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.55	
		dd1	14; 0; 1	3	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.56	
		dd2	14; 0; 1	3	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.52	
		Sieve-SDP	0; 0; 4	2	1	0.00		0	0.00e+00, 0.00e+00	0.00e+00	1.57	
17	Example9size20	before	0; 0; 20	20				1	0.00e+00, 3.39e-01	5.00e-01	2.05	
		pd1	0; 0; 1	1	1	0.06	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.21	
		pd2	0; 0; 1	1	1	0.07	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.18	
		dd1	209; 0; 1	20	1	0.19	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.55	-1
		dd2	209; 0; 1	20	1	0.24	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.54	-1
		Sieve-SDP			Infeas	0.00					0.00	1
18	Example9size100	before	0; 0; 100	100				1	0.00e+00, 3.43e-01	5.00e-01	2.15	
		pd1	0; 0; 1	1	1	0.08	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.20	
		pd2	0; 0; 1	1	1	0.22	0.00	1	0.00e+00, 1.00e+00	5.00e-01	1.28	
		dd1	5049; 0; 1	100	1	1.50	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.56	-1
		dd2	5049; 0; 1	100	1	3.24	0.00	0	0.00e+00, 0.00e+00	0.00e+00	1.56	-1
		Sieve-SDP			Infeas	0.01					0.00	1
19	RandGen6	before	0; 0; 320	140				0	3.95e-06, 3.24e-06	2.29e-05	24.83	
		pd1			0	3.53	0.98					
		pd2			0	16.38	0.98					
		dd1			0	0.74	0.98					
		dd2	19985; 0; 250	140	1	35.79	2.05	0	1.68e-07, 1.26e-11	8.00e-07	6.13	2,3
		Sieve-SDP	0; 0; 120	70	1	1.43		0	3.73e-06, 3.04e-06	9.17e-06	3.26	
20	RandGen7	before	0; 0; 40	27				0	9.42e-07, 4.22e-07	4.69e-06	1.63	
		pd1			0	0.04	0.02					
		pd2	0; 0; 28	14	1	0.10	0.03	0	9.85e-07, 4.53e-07	3.27e-06	1.66	
		dd1			0	0.04	0.02					
		dd2	649; 0; 18	27	1	0.09	0.03	0	2.65e-11, 4.69e-16	7.21e-11	1.84	2
		Sieve-SDP	0; 0; 28	14	1	0.02		0	9.85e-07, 4.53e-07	3.27e-06	1.73	
21	RandGen8	before	0; 0; 60	40				0	5.41e-09, 2.44e-09	9.31e-08	1.73	
		pd1			0	0.08	0.02					
		pd2			0	0.29	0.02					
		dd1			0	0.04	0.02					
		dd2	1269; 0; 33	40	1	0.33	0.03	0	2.15e-15, 2.78e-19	7.19e-14	1.73	
		Sieve-SDP	0; 0; 30	20	1	0.05		0	1.52e-09, 6.33e-10	9.04e-09	1.67	
22	copos_1	before	0; 0; 35	210				0	0.00e+00, 1.11e-08	4.40e-07	1.68	
		pd1			0	0.03	0.00					
		pd2	0; 0; 25	160	1	0.09	0.02	0	0.00e+00, -3.86e-10	2.12e-08	1.61	
		dd1			0	0.02	0.00					
		dd2			0	0.04	0.00					
		Sieve-SDP			0	0.04						
23	copos_2	before	0; 0; 120	1716				0	0.00e+00, 5.75e-11	1.69e-08	3.40	
		pd1			0	0.05	0.00					
		pd2	0; 0; 96	1524	1	0.60	0.11	0	0.00e+00, -7.97e-13	6.70e-11	2.53	
		dd1			0	0.04	0.00					
		dd2			0	0.16	0.00					
		Sieve-SDP			0	0.14						
24	copos_3	before	0; 0; 286	8008				0	0.00e+00, -4.93e-10	1.59e-07	51.49	
		pd1			0	0.11	0.01					
		pd2	0; 0; 242	7524	1	34.51	0.57	0	0.00e+00, -4.51e-11	1.26e-08	34.73	
		dd1			0	0.08	0.01					
		dd2			0	0.87	0.01					
		Sieve-SDP			0	0.59						
25	copos_4	before	0; 0; 560	27132				0	0.00e+00, -9.00e-11	7.20e-08	1630.41	
		pd1			0	0.51	0.06					
		pd2	0; 0; 490	26152	1	27.43	1.96	0	0.00e+00, -1.70e-10	6.56e-08	1113.73	
		dd1			0	0.34	0.06					
		dd2			0	4.54	0.06					
		Sieve-SDP			0	2.51						
26	cprank_1	before	9; 0; 1_9, 10, 9	46				0	-3.00e+00, -3.00e+00	3.50e-08	1.42	
		pd1			0	0.03	0.00					
		pd2			0	0.02	0.00					
		dd1	30; 0; 1_7, 8, 9	46	1	0.03	0.01	0	-3.00e+00, -3.00e+00	4.62e-08	0.95	
		dd2	30; 0; 1_7, 8, 9	46	1	0.05	0.01	0	-3.00e+00, -3.00e+00	3.87e-08	0.94	
		Sieve-SDP			0	0.03						

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
27	cprank_2	before	$1296; 0; 1_{81}, 82, 81$	3322				0	-9.00e+00, -9.00e+00	6.62e-08	13.48	
		pd1			0	0.08	0.00					
		pd2			0	0.17	0.00					
		dd1	$3456; 0; 1_{49}, 50, 81$	3322	1	0.13	0.28	0	-9.00e+00, -9.00e+00	6.65e-09	10.28	
		dd2	$3456; 0; 1_{49}, 50, 81$	3322	1	0.50	0.29	0	-9.00e+00, -9.00e+00	1.51e-09	9.75	
		Sieve-SDP			0	0.13						
28	hInfeas12	before	$0; 0; 6_2, 12$	43				0	-2.44e-09, -1.58e-09	1.80e+00	1.22	
		pd1			0	0.02	0.00					
		pd2	0; 0; 6, 2, 6	22	1	0.04	0.00	0	-7.98e-11, -7.03e-11	1.79e+00	1.33	
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			0	0.01						
29	horn2	before	0; 0; 4	7				0	0.00e+00, 6.28e-13	8.49e-13	0.74	
		pd1			0	0.01	0.00					
		pd2	0; 0; 2	3	1	0.03	0.00	0	0.00e+00, 0.00e+00	1.57e-16	0.70	
		dd1			0	0.01	0.00					
		dd2			0	0.01	0.00					
		Sieve-SDP			0	0.00						
30	horn3	before	0; 0; 10	28				0	0.00e+00, 1.46e-07	8.62e-07	0.81	
		pd1			0	0.01	0.00					
		pd2	0; 0; 6	16	1	0.03	0.00	0	0.00e+00, 3.53e-09	2.65e-08	0.74	
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			0	0.00						
31	horn4	before	0; 0; 20	84				0	0.00e+00, 1.13e-07	1.90e-06	0.81	
		pd1			0	0.02	0.00					
		pd2	0; 0; 14	60	1	0.05	0.00	0	0.00e+00, 7.11e-09	7.44e-08	0.81	2
		dd1			0	0.01	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			0	0.01						
32	horn5	before	0; 0; 35	210				0	0.00e+00, 1.07e-08	2.69e-07	0.79	
		pd1			0	0.02	0.00					
		pd2	0; 0; 25	160	1	0.06	0.01	0	0.00e+00, -2.28e-09	2.35e-07	0.74	
		dd1			0	0.01	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.02						
33	hornD2	before	0; 0; 4	3				0	-5.26e-08, 0.00e+00	5.26e-08	0.80	
		pd1			0	0.02	0.00					
		pd2			0	0.02	0.00					
		dd1			0	0.01	0.00					
		dd2	7; 0; 2	3	1	0.02	0.00	0	-1.88e-16, 0.00e+00	1.78e-15	0.73	
		Sieve-SDP			0	0.00						
34	hornD3	before	0; 0; 10	27				0	-5.58e-08, 0.00e+00	8.62e-07	0.80	
		pd1			0	0.01	0.00					
		pd2			0	0.02	0.00					
		dd1			0	0.01	0.00					
		dd2	34; 0; 6	27	1	0.03	0.00	0	2.18e-08, 0.00e+00	6.58e-08	1.00	
		Sieve-SDP			0	0.00						
35	hornD4	before	0; 0; 20	126				0	1.77e-07, 0.00e+00	1.02e-06	0.79	
		pd1			0	0.02	0.00					
		pd2			0	0.03	0.00					
		dd1			0	0.02	0.00					
		dd2	105; 0; 14	126	1	0.04	0.01	0	1.01e-08, 0.00e+00	8.98e-08	0.92	2
		Sieve-SDP			0	0.01						
36	hornD5	before	0; 0; 35	420				0	2.33e-08, 0.00e+00	1.83e-07	0.77	
		pd1			0	0.02	0.00					
		pd2			0	0.03	0.00					
		dd1			0	0.02	0.00					
		dd2	305; 0; 25	420	1	0.08	0.03	0	5.70e-10, 0.00e+00	2.02e-09	0.89	
		Sieve-SDP			0	0.03						
37	hybridLyap	before	$860; 0; 6, 108, 11_{10}$	3093				0	0.00e+00, 4.29e-07	1.27e-04	7.86	
		pd1	$860; 0; 6, 56, 11, 1_2, 11, 1_2, 11_2$	1607	1	0.14	0.09	0	0.00e+00, 3.94e-07	6.16e-05	1.35	
		pd2	$860; 0; 6, 34, 8, 1_2, 8, 1_2, 9, 7$	1173	1	1.03	0.05	0	0.00e+00, 4.14e-09	4.78e-07	0.91	2
		dd1			0	0.03	0.00					
		dd2			0	0.13	0.00					
		Sieve-SDP			0	0.12						
38	leverage_limit	before	$0; 18100; 151_{100}, 30_{100}$	68195				0	-8.75e+01, -8.75e+01	2.14e-05	223.18	
		pd1			0	2.33	0.19					
		pd2	$0; 18100; 151_{99}, 121, 30_{100}$	67700	1	142.81	8.73	0	-8.75e+01, -8.75e+01	5.63e-06	181.57	3
		dd1	$958500; 18100; 61_{100}, 30_{100}$	68195	1	4.23	8.09	0	-8.75e+01, -8.75e+01	6.46e-06	231.19	3
		dd2	$958500; 18100; 61_{100}, 30_{100}$	68195	1	689.57	8.19	0	-8.75e+01, -8.75e+01	8.03e-06	159.60	3
		Sieve-SDP	$0; 18100; 143_{97}, 141_3, 26_{98}, 25_2$	56196	1	321.64		0	-8.74e+01, -8.74e+01	6.57e-05	199.83	3
39	long_only	before	$0; 9000; 91_{100}, 30_{100}$	59095				0	-4.13e+01, -4.13e+01	1.73e-06	195.09	
		pd1			0	1.32	0.19					
		pd2	$0; 9000; 91_{99}, 61, 30_{100}$	58600	1	27.80	7.90	0	-4.13e+01, -4.13e+01	2.32e-06	131.77	3
		dd1	$229500; 9000; 61_{100}, 30_{100}$	59095	1	2.15	7.93	0	-4.13e+01, -4.13e+01	6.50e-06	152.63	3
		dd2	$229500; 9000; 61_{100}, 30_{100}$	59095	1	699.96	7.88	0	-4.13e+01, -4.13e+01	2.84e-06	194.93	3
		Sieve-SDP	$0; 8573; 83_{97}, 81_3, 26_{98}, 25_2$	46670	1	200.67		0	-4.13e+01, -4.13e+01	1.64e-06	117.57	3

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
40	sector_neutral	before	$0; 12000; 121_{100}, 30_{100}$	62392				0	-1.21e+02, -1.21e+02	5.02e-04	147.12	
		pd1			0	1.97	0.27					
		pd2	$0; 12000; 121_{99}, 91, 30_{100}$	61897	1	221.99	7.91	0	-1.21e+02, -1.21e+02	6.94e-05	191.66	3
		dd1	$549000; 12000; 61_{100}, 30_{100}$	62392	1	3.17	8.12	0	-1.21e+02, -1.21e+02	7.49e-05	156.23	3
		dd2	$549000; 12000; 61_{100}, 30_{100}$	62392	1	28.35	8.34	0	-1.21e+02, -1.21e+02	3.41e-04	112.22	3
		Sieve-SDP	$0; 12000; 121_{99}, 111, 30_{100}$	62247	1	69.21		0	-1.21e+02, -1.21e+02	4.78e-04	193.34	3
41	unconstrained	before	$0; 12000; 121_{100}, 30_{100}$	62095				0	-1.33e+02, -1.33e+02	2.41e-06	278.82	
		pd1			0	2.13	0.19					
		pd2	$0; 12000; 121_{99}, 91, 30_{100}$	61600	1	43.95	7.83	0	-1.33e+02, -1.33e+02	5.25e-06	177.75	3
		dd1	$549000; 12000; 61_{100}, 30_{100}$	62095	1	2.98	7.89	0	-1.33e+02, -1.33e+02	4.87e-05	316.37	-2
		dd2	$549000; 12000; 61_{100}, 30_{100}$	62095	1	656.00	7.91	0	-1.33e+02, -1.33e+02	4.61e-05	405.96	-2
		Sieve-SDP	$0; 12000; 113_{97}, 111_3, 26_{98}, 25_2$	50097	1	226.38		0	-1.28e+02, -1.28e+02	1.32e-05	180.73	3
42	unboundDim1R1	before	0; 2; 2	2				0	1.33e-09, -7.05e-10	4.38e-09	0.96	
		pd1			0	0.03	0.00					
		pd2			0	0.02	0.00					
		dd1			0	0.02	0.00					
		dd2			0	0.01	0.00					
		Sieve-SDP	0; 1; 1	1	1	0.01		0	0.00e+00, 0.00e+00	0.00e+00	0.84	
43	unboundDim1R2	before	$0; 0; 3, 2_2$	4				0	-9.26e-12, -8.65e-12	7.07e-01	1.61	
		pd1	$0; 0; 1_2$	1	1	0.06	0.01	0	0.00e+00, 0.00e+00	0.00e+00	0.86	2
		pd2	$0; 0; 1_2$	1	1	0.06	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.86	2
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.00		0	0.00e+00, 0.00e+00	0.00e+00	0.86	2
44	unboundDim1R3	before	$0; 0; 4, 3_2$	6				0	-4.85e-09, -4.71e-09	7.07e-01	1.47	
		pd1	$0; 0; 1_2$	1	1	0.08	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.84	2
		pd2	$0; 0; 1_2$	1	1	0.09	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.84	2
		dd1			0	0.05	0.00					
		dd2			0	0.04	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.00		0	0.00e+00, 0.00e+00	0.00e+00	0.98	2
45	unboundDim1R4	before	$0; 0; 5, 4_2$	8				0	-2.58e-08, -2.55e-08	7.07e-01	1.41	
		pd1	$0; 0; 1_2$	1	1	0.11	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.91	2
		pd2	$0; 0; 1_2$	1	1	0.12	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.93	2
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.00		0	0.00e+00, 0.00e+00	0.00e+00	0.88	2
46	unboundDim1R5	before	$0; 0; 6, 5_2$	10				0	-1.00e+00, -1.00e+00	9.88e-08	1.16	
		pd1	$0; 0; 1_2$	1	1	0.13	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.90	3
		pd2	$0; 0; 1_2$	1	1	0.15	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.88	3
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.00		0	0.00e+00, 0.00e+00	0.00e+00	0.85	3
47	unboundDim1R6	before	$0; 0; 7, 6_2$	12				0	-1.00e+00, -1.00e+00	2.15e-07	1.07	
		pd1	$0; 0; 1_2$	1	1	0.15	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.87	3
		pd2	$0; 0; 1_2$	1	1	0.18	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.85	3
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.01		0	0.00e+00, 0.00e+00	0.00e+00	0.85	3
48	unboundDim1R7	before	$0; 0; 8, 7_2$	14				0	-1.00e+00, -1.00e+00	5.11e-08	1.04	
		pd1	$0; 0; 1_2$	1	1	0.18	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.87	3
		pd2	$0; 0; 1_2$	1	1	0.22	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.85	3
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.01		0	0.00e+00, 0.00e+00	0.00e+00	0.83	3
49	unboundDim1R8	before	$0; 0; 9, 8_2$	16				0	-1.00e+00, -1.00e+00	5.43e-08	1.07	
		pd1	$0; 0; 1_2$	1	1	0.21	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.86	3
		pd2	$0; 0; 1_2$	1	1	0.24	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.88	3
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.01		0	0.00e+00, 0.00e+00	0.00e+00	0.85	3
50	unboundDim1R9	before	$0; 0; 10, 9_2$	18				0	-1.00e+00, -1.00e+00	6.50e-08	1.04	
		pd1	$0; 0; 1_2$	1	1	0.27	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.84	3
		pd2	$0; 0; 1_2$	1	1	0.28	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.97	3
		dd1			0	0.03	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.02		0	0.00e+00, 0.00e+00	0.00e+00	0.92	3
51	unboundDim1R10	before	$0; 0; 11, 10_2$	20				0	-1.00e+00, -1.00e+00	1.41e-07	1.08	
		pd1	$0; 0; 1_2$	1	1	0.26	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.88	3
		pd2	$0; 0; 1_2$	1	1	0.31	0.00	0	0.00e+00, 0.00e+00	0.00e+00	0.87	3
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$0; 0; 1_2$	1	1	0.02		0	0.00e+00, 0.00e+00	0.00e+00	0.83	3
52	vamos_5_34	before	0; 0; 52	721				0	0.00e+00, -6.15e-09	7.45e-08	1.29	
		pd1			0	0.04	0.01					
		pd2				crash						MM
		dd1			0	0.02	0.01					
		dd2			0	0.04	0.01					
		Sieve-SDP			0	0.05						

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
53	wei_wagner_F7_minus_4	before	0; 0; 8	31				0	0.00e+00, -9.60e-13	1.11e-11	0.92	
		pd1			0	0.01	0.00					
		pd2	0; 0; 5	14	1	0.03	0.00	0	0.00e+00, -5.80e-11	2.12e-10	0.75	
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			0	0.00						
54	wei_wagner_P7	before	0; 0; 8	32				0	0.00e+00, -1.46e-08	9.09e-08	0.91	
		pd1			0	0.03	0.00					
		pd2	0; 0; 4	10	1	0.04	0.00	0	0.00e+00, -3.02e-10	1.31e-09	0.91	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.00						
55	wei_wagner_W3Plus	before	0; 0; 8	31				0	0.00e+00, -6.06e-09	5.47e-08	0.98	
		pd1			0	0.03	0.00					
		pd2	0; 0; 3	6	1	0.03	0.00	0	0.00e+00, -4.77e-09	1.11e-08	0.94	
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP			0	0.00						
56	wei_wagner_W3_PlusE	before	0; 0; 9	38				0	0.00e+00, -9.18e-09	5.53e-08	0.99	
		pd1			0	0.02	0.00					
		pd2	0; 0; 5	15	1	0.04	0.00	0	0.00e+00, -7.21e-09	3.21e-08	0.98	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.00						
57	wei_wagner_nP_minus_1_24	before	0; 0; 12	64				0	0.00e+00, -5.50e-09	8.80e-08	1.01	
		pd1			0	0.02	0.00					
		pd2	0; 0; 6	21	1	0.04	0.00	0	0.00e+00, -1.08e-11	5.60e-11	0.94	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.01						
58	wei_wagner_nP_minus_9_12	before	0; 0; 12	64				0	0.00e+00, -3.92e-09	4.87e-08	1.07	
		pd1			0	0.02	0.00					
		pd2	0; 0; 5	15	1	0.04	0.00	0	0.00e+00, -4.11e-15	2.34e-14	0.94	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.01						
59	wei_wagner_vamos_12	before	0; 0; 16	103				0	0.00e+00, -1.59e-08	1.38e-07	1.12	
		pd1			0	0.03	0.00					
		pd2	0; 0; 13	74	1	0.05	0.01	0	0.00e+00, -2.54e-10	1.62e-09	0.95	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP			0	0.02						

A.2 Detailed results on the “Henrion-Toh” problems

Note that this dataset has 98 problems. From these 18 problems were reduced by at least one of the five methods.

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
1	sedumi-brown	before	925; 0; 56	461				0	-5.39e-09, 0.00e+00	2.87e-07	1.18	
		pd1	925; 0; 21	251	1	0.08	0.01	0	7.12e-10, 0.00e+00	1.63e-06	1.19	
		pd2	925; 0; 21	251	1	0.17	0.01	0	7.12e-10, 0.00e+00	1.63e-06	1.02	
		dd1			0	0.03	0.00					
		dd2			0	0.05	0.00					
		Sieve-SDP	925; 0; 21	251	1	0.07		0	7.12e-10, 0.00e+00	1.63e-06	1.00	
2	sedumi-conform3	before	630; 0; 56	285				0	2.05e-08, 0.00e+00	4.54e-07	0.98	
		pd1	630; 0; 53	273	1	0.05	0.02	0	2.51e-08, 0.00e+00	4.90e-07	0.96	
		pd2	630; 0; 53	273	1	0.10	0.02	0	2.51e-08, 0.00e+00	4.90e-07	1.01	
		dd1			0	0.02	0.00					
		dd2			0	0.04	0.00					
		Sieve-SDP	630; 0; 53	273	1	0.03		0	2.51e-08, 0.00e+00	4.90e-07	0.95	
3	sedumi-conform4	before	1890; 0; 84	454				0	-2.58e-08, 0.00e+00	5.57e-06	1.27	
		pd1	1890; 0; 81	442	1	0.10	0.04	0	-4.12e-08, 0.00e+00	4.44e-06	1.18	
		pd2	1890; 0; 81	442	1	0.21	0.04	0	-4.12e-08, 0.00e+00	4.44e-06	1.14	
		dd1			0	0.03	0.00					
		dd2			0	0.07	0.00					
		Sieve-SDP	1890; 0; 81	442	1	0.04		0	-4.12e-08, 0.00e+00	4.44e-06	1.14	
4	sedumi-fp210	before	66; 0; 66, 11_{10}	1000				0	3.75e-01, 3.75e-01	2.15e-07	1.67	
		pd1	66; 0; 11_{11}	285	1	0.11	0.04	0	3.75e-01, 3.75e-01	2.55e-08	1.35	
		pd2	66; 0; 11_{11}	285	1	0.29	0.06	0	3.75e-01, 3.75e-01	2.55e-08	1.08	
		dd1			0	0.04	0.00					
		dd2			0	0.08	0.00					
		Sieve-SDP	66; 0; 11_{11}	285	1	0.07		0	6.24e-08, 5.44e-08	8.28e-08	1.11	3
5	sedumi-fp23	before	0; 0; 28, 7_{13}	209				0	2.13e+02, 2.13e+02	3.97e-06	1.24	
		pd1	0; 0; 7_{14}	83	1	0.08	0.02	0	2.13e+02, 2.13e+02	9.96e-07	1.14	3
		pd2	0; 0; 7_{14}	83	1	0.11	0.02	0	2.13e+02, 2.13e+02	9.96e-07	1.16	3
		dd1			0	0.03	0.00					
		dd2			0	0.05	0.00					
		Sieve-SDP	0; 0; 7_{14}	83	1	0.03		0	2.13e+02, 2.13e+02	9.96e-07	1.13	3

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
6	sedumi-fp24	before	$0;0;105,14_{35}$	2379				0	1.95e+02, 1.95e+02	9.68e-08	5.58	
		pd1	$0;0;14_{36}$	559	1	0.34	0.21	0	1.95e+02, 1.95e+02	9.66e-11	1.50	
		pd2	$0;0;14_{36}$	559	1	0.94	0.23	0	1.95e+02, 1.95e+02	9.66e-11	1.58	
		dd1			0	0.06	0.00					
		dd2			0	0.26	0.00					
		Sieve-SDP	$0;0;14_{36}$	559	1	0.30		0	1.95e+02, 1.95e+02	9.66e-11	1.57	
7	sedumi-fp25	before	$0;0;28,7_{15}$	209				0	1.10e+01, 1.10e+01	6.63e-06	1.12	
		pd1	$0;0;7_{16}$	83	1	0.09	0.03	0	1.10e+01, 1.10e+01	1.39e-07	1.11	2,3
		pd2	$0;0;7_{16}$	83	1	0.14	0.03	0	1.10e+01, 1.10e+01	1.39e-07	1.19	2,3
		dd1			0	0.05	0.00					
		dd2			0	0.08	0.00					
		Sieve-SDP	$0;0;7_{16}$	83	1	0.04		0	1.10e+01, 1.10e+01	1.39e-07	1.24	2,3
8	sedumi-fp26	before	$0;0;66,11_{31}$	1000				0	2.68e+02, 2.68e+02	3.74e-08	2.35	
		pd1	$0;0;11_{32}$	285	1	0.21	0.17	0	2.68e+02, 2.68e+02	1.18e-07	1.24	
		pd2	$0;0;11_{32}$	285	1	0.68	0.19	0	2.68e+02, 2.68e+02	1.18e-07	1.25	
		dd1			0	0.06	0.01					
		dd2			0	0.43	0.01					
		Sieve-SDP	$0;0;11_{32}$	285	1	0.14		0	2.68e+02, 2.68e+02	1.18e-07	1.14	
9	sedumi-fp27	before	$0;0;66,11_{25}$	1000				0	3.90e+01, 3.90e+01	1.94e-10	2.13	
		pd1	$0;0;11_{26}$	285	1	0.16	0.11	0	3.90e+01, 3.90e+01	3.98e-09	1.30	
		pd2	$0;0;11_{26}$	285	1	0.46	0.12	0	3.90e+01, 3.90e+01	3.98e-09	1.20	
		dd1			0	0.05	0.01					
		dd2			0	0.29	0.01					
		Sieve-SDP	$0;0;11_{26}$	285	1	0.13		0	3.90e+01, 3.90e+01	3.98e-09	1.25	
10	sedumi-fp32	before	$0;0;165,45_{22}$	3002				0	-7.05e+00, -7.05e+00	2.64e-07	47.49	
		pd1	$0;0;45_4,9_3,45_{16}$	1286	1	2.30	0.56	0	-7.05e+00, -7.05e+00	2.49e-06	9.46	3
		pd2	$0;0;45_4,9_3,45_{16}$	1286	1	4.79	0.58	0	-7.05e+00, -7.05e+00	2.49e-06	9.54	3
		dd1			0	0.10	0.02					
		dd2			0	10.55	0.02					
		Sieve-SDP	$0;0;45_4,9_3,45_{16}$	1286	1	1.11		0	-7.05e+00, -7.05e+00	2.49e-06	9.61	3
11	sedumi-fp33	before	$0;0;21,6_{16}$	125				0	-1.01e+04, -1.01e+04	3.36e-07	1.31	
		pd1	$0;0;13,6_{16}$	105	1	0.09	0.03	0	-1.01e+04, -1.01e+04	2.98e-07	1.21	
		pd2	$0;0;13,6_{16}$	105	1	0.19	0.03	0	-1.01e+04, -1.01e+04	2.98e-07	1.25	
		dd1			0	0.04	0.00					
		dd2			0	0.14	0.00					
		Sieve-SDP	$0;0;14,6_{16}$	111	1	0.04		0	-1.18e+04, -1.18e+04	9.28e-02	1.14	-2
12	sedumi-fp34	before	$0;0;28,7_{16}$	209				0	1.72e+02, 1.72e+02	8.10e-07	1.12	
		pd1	$0;0;7,1_2,7_{14}$	83	1	0.11	0.03	0	1.72e+02, 1.72e+02	3.11e-07	1.08	
		pd2	$0;0;7,1_2,7_{14}$	83	1	0.11	0.03	0	1.72e+02, 1.72e+02	3.11e-07	1.08	
		dd1			0	0.03	0.00					
		dd2			0	0.07	0.00					
		Sieve-SDP	$0;0;7,1_2,7_{14}$	83	1	0.03		0	1.72e+02, 1.72e+02	3.11e-07	1.10	
13	sedumi-fp35	before	$0;0;35,20_8$	164				0	4.00e+00, 4.00e+00	5.76e-06	1.29	
		pd1	$0;0;20_8,10$	119	1	0.13	0.04	0	4.00e+00, 4.00e+00	5.66e-07	1.29	2,3
		pd2	$0;0;20_8,10$	119	1	0.31	0.04	0	4.00e+00, 4.00e+00	5.66e-07	1.29	2,3
		dd1			0	0.04	0.00					
		dd2			0	0.09	0.00					
		Sieve-SDP	$0;0;20_8,10$	119	1	0.05		0	4.00e+00, 4.00e+00	5.66e-07	1.26	2,3
14	sedumi-fp410	before	$1;0;6,3_4$	14				0	1.67e+01, 1.67e+01	5.98e-08	1.12	
		pd1	$1;0;4,3_4$	10	1	0.07	0.00	0	1.67e+01, 1.67e+01	1.40e-08	0.98	
		pd2	$1;0;4,3_4$	10	1	0.07	0.00	0	1.67e+01, 1.67e+01	1.40e-08	1.01	
		dd1			0	0.03	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP	$1;0;4,3_4$	10	1	0.01		0	2.71e+01, 2.71e+01	3.42e-08	0.97	3
15	sedumi-fp44	before	$0;0;4,3_2$	6				0	4.44e+02, 4.44e+02	4.64e-08	1.15	
		pd1	$0;0;3_3$	5	1	0.04	0.00	0	4.44e+02, 4.44e+02	3.51e-08	1.16	
		pd2	$0;0;3_3$	5	1	0.04	0.00	0	4.44e+02, 4.44e+02	3.51e-08	1.11	
		dd1			0	0.02	0.00					
		dd2			0	0.05	0.00					
		Sieve-SDP	$0;0;3_3$	5	1	0.00		0	4.44e+02, 4.44e+02	3.51e-08	1.10	
16	sedumi-fp46	before	$0;0;10,6_2$	27				0	6.70e-08, -2.54e-07	4.78e-07	1.03	
		pd1	$0;0;5,3,2$	11	1	0.11	0.00	0	1.54e-07, -2.20e-08	2.22e-07	1.01	
		pd2	$0;0;5,3,2$	11	1	0.14	0.00	0	1.54e-07, -2.20e-08	2.22e-07	0.93	
		dd1			0	0.02	0.00					
		dd2			0	0.02	0.00					
		Sieve-SDP	$0;0;5,3,2$	11	1	0.02		0	1.54e-07, -2.20e-08	2.22e-07	0.91	
17	sedumi-fp49	before	$1;0;6,3_4$	14				0	1.67e+01, 1.67e+01	5.98e-08	1.06	
		pd1	$1;0;4,3_4$	10	1	0.06	0.00	0	1.67e+01, 1.67e+01	1.40e-08	1.00	
		pd2	$1;0;4,3_4$	10	1	0.06	0.00	0	1.67e+01, 1.67e+01	1.40e-08	0.95	
		dd1			0	0.02	0.00					
		dd2			0	0.03	0.00					
		Sieve-SDP	$1;0;4,3_4$	10	1	0.01		0	2.71e+01, 2.71e+01	3.42e-08	0.93	3
18	sedumi-l4	before	$0;0;45$	152				0	3.70e-02, 3.70e-02	7.10e-08	1.04	
		pd1	$0;0;1$	1	1	0.17	0.00	1	0.00e+00, 1.00e+00	5.00e-01	0.61	1
		pd2	$0;0;1$	1	1	0.19	0.00	1	0.00e+00, 1.00e+00	5.00e-01	0.61	1
		dd1			0	0.03	0.00					
		dd2			0	0.04	0.00					
		Sieve-SDP			Infeas	0.04					0.00	1

A.3 Detailed results on the "moreSDPs" problems

Note that this problem collection has 31 problems. From these 8 problems were reduced by at least one of the five methods. There were 5 problems on which pd2 or dd2 ran out of memory, or crashed.

No.	name		f;l;s	m	red	t_{prep}	t_{conv}	Infeas	obj (P, D)	DIMACS	t_{sol}	help
1	diamond_patch	before	0; 0; 5477	5478				0	1.63e+01, 1.63e+01	3.56e-04	9875.07	
		pd1			0	27.72	0.05					
		pd2				MM						MM
		dd1			0	25.88	0.05					
		dd2			0	3053.88	0.05					
		Sieve-SDP			0	0.98						
2	e_moment_stable	before	0; 342; 171, 18_{17}	5984				0	-1.98e-01, -1.98e-01	1.14e-05	46.00	
		pd1	0; 342; 18_{18}	1139	1	0.53	0.15	0	-1.98e-01, -1.98e-01	8.44e-06	1.94	
		pd2	0; 342; 18_{18}	1139	1	0.81	0.13	0	-1.98e-01, -1.98e-01	8.44e-06	1.58	
		dd1			0	0.05	0.00					
		dd2			0	0.33	0.00					
		Sieve-SDP	0; 342; 18_{18}	1139	1	0.51		0	-1.98e-01, -1.98e-01	8.44e-06	1.49	
3	G60_mb	before	0; 0; 7000	7001				0	1.93e+03, 1.93e+03	2.69e-04	33699.25	
		pd1			0	107.66	10.87					
		pd2				MM						MM
		dd1			0	72.76	10.87					
		dd2				MM						MM
		Sieve-SDP			0	22.42						
4	maxG60	before	0; 0; 7000	7000				0	-1.52e+04, -1.52e+04	6.73e-07	5740.25	
		pd1			0	49.24	0.01					
		pd2				MM						MM
		dd1			0	47.25	0.01					
		dd2				MM						MM
		Sieve-SDP			0	1.06						
5	ice_2.0	before	0; 0; 8113	8113				0	6.81e+03, 6.81e+03	4.71e-07	18409.71	
		pd1			0	66.30	0.02					
		pd2				MM						MM
		dd1			0	71.78	0.02					
		dd2				MM						MM
		Sieve-SDP			0	1.40						
6	neu3	before	0; 2; 418	7364				0	7.10e-08, 1.12e-08	2.01e-06	164.58	
		pd1	0; 2; 87	1152	1	0.93	0.12	0	4.69e-08, 3.50e-08	1.93e-07	2.85	2
		pd2	0; 2; 87	1152	1	5.45	0.11	0	4.69e-08, 3.50e-08	1.93e-07	2.76	2
		dd1			0	0.15	0.02					
		dd2			0	2.28	0.02					
		Sieve-SDP	0; 2; 87	1152	1	2.49		0	4.69e-08, 3.50e-08	1.93e-07	2.87	2
7	neu3g	before	0; 0; 462	8007				0	4.58e-08, -2.89e-09	8.67e-07	161.55	
		pd1	0; 0; 87	1151	1	1.31	0.12	0	8.91e-08, 5.65e-08	2.91e-07	2.90	
		pd2	0; 0; 87	1151	1	10.54	0.12	0	8.91e-08, 5.65e-08	2.91e-07	3.10	
		dd1			0	0.18	0.04					
		dd2			0	2.63	0.04					
		Sieve-SDP	0; 0; 87	1151	1	2.36		0	8.91e-08, 5.65e-08	2.91e-07	2.83	
8	p_auss2_3.0	before	0; 0; 9115	9115				0	8.62e+03, 8.62e+03	2.37e-07	27459.23	
		pd1			0	89.73	0.02					
		pd2				MM						MM
		dd1			0	88.05	0.02					
		dd2				MM						MM
		Sieve-SDP			0	1.85						
9	rose13	before	0; 0; 105	2379				0	1.20e+01, 1.20e+01	2.52e-07	8.31	
		pd1	0; 0; 92	1911	1	0.10	0.15	0	1.20e+01, 1.20e+01	9.91e-07	5.28	
		pd2	0; 0; 80	1523	1	0.50	0.11	0	1.20e+01, 1.20e+01	2.11e-07	3.03	
		dd1			0	0.03	0.00					
		dd2			0	0.14	0.00					
		Sieve-SDP	0; 0; 92	1911	1	0.46		0	1.20e+01, 1.20e+01	9.91e-07	5.84	
10	rose15	before	0; 2; 135	3860				0	-2.84e-06, -2.68e-06	1.78e-05	20.90	
		pd1	0; 2; 121	3181	1	0.07	0.26	0	-2.35e-07, 4.05e-08	3.17e-05	12.57	3
		pd2	0; 2; 107	2593	1	0.65	0.18	0	1.35e-09, 1.36e-09	1.22e-07	6.14	2,3
		dd1			0	0.04	0.00					
		dd2			0	0.20	0.00					
		Sieve-SDP	0; 2; 121	3181	1	0.63		0	-2.35e-07, 4.05e-08	3.17e-05	12.72	3
11	taha1a	before	0; 0; 252, 56_3, 126_{10}	3002				0	-1.00e+00, -1.00e+00	9.36e-07	37.21	
		pd1	0; 0; 126, 56_3, 126_{10}	2001	1	10.68	0.78	0	-1.00e+00, -1.00e+00	1.21e-07	21.38	
		pd2	0; 0; 126, 56_3, 126_{10}	2001	1	18.31	0.73	0	-1.00e+00, -1.00e+00	1.21e-07	21.34	
		dd1			0	0.20	0.06					
		dd2			0	20.80	0.06					
		Sieve-SDP	0; 0; 126, 56_3, 126_{10}	2001	1	1.97		0	-1.00e+00, -1.00e+00	1.21e-07	21.41	
12	taha1b	before	0; 3; 286, 66_{20}	8007				0	-7.73e-01, -7.73e-01	1.58e-07	152.91	
		pd1	0; 3; 66_{21}	3002	1	14.26	0.89	0	-7.73e-01, -7.73e-01	2.95e-07	34.26	
		pd2	0; 3; 66_{21}	3002	1	17.57	0.85	0	-7.73e-01, -7.73e-01	2.95e-07	34.41	
		dd1			0	0.15	0.04					
		dd2			0	1.77	0.04					
		Sieve-SDP	0; 3; 66_{21}	3002	1	2.16		0	-7.73e-01, -7.73e-01	2.95e-07	34.91	
13	taha1c	before	0; 0; 462, 126_3, 252_{10}	6187				0	-1.00e+00, -1.00e+00	3.02e-07	294.99	
		pd1	0; 0; 252, 126_3, 252_{10}	4367	1	141.64	1.93	0	-1.00e+00, -1.00e+00	4.37e-07	180.33	
		pd2	0; 0; 252, 126_3, 252_{10}	4367	1	189.49	1.96	0	-1.00e+00, -1.00e+00	4.37e-07	180.95	
		dd1			0	0.73	0.25					
		dd2			0	166.27	0.25					
		Sieve-SDP	0; 0; 252, 126_3, 252_{10}	4367	1	12.02		0	-1.00e+00, -1.00e+00	4.37e-07	181.10	

B Core Matlab code

In this section we provide our core Matlab code with some comments. In our code we physically delete rows and columns of the A_i and of C only at the very end. During the exceution of the algorithm we only *mark* such rows, columns and constraints as deleted.

We use two arrays to keep track of what has been marked deleted:

(1) The m-vector `undeleted`, whose ith entry is 1 if constraint i *has not* been deleted, and 0 if it *has* been deleted.

(2) The sparse array $\texttt{I} \in \{0,1\}^{n\times(m+1)}$ with entries defined as follows.

(a) For all i and for $1 \le j \le m$

$$I(i,j) = \begin{cases} 1, & \text{if in } A_j \text{ the } i\text{th row and column are all zero or have been deleted} \\ 0, & \text{otherwise} \end{cases}$$

(b) For all i

$$I(i,m+1) = \begin{cases} 1 & \text{if in all } A_j \text{ the } i\text{th row and column have been deleted} \\ 0 & \text{otherwise} \end{cases}$$

```
function[Ared, bred, cred, info] = SieveSDP(A, b, c, epsilon, maxiter)
    % Inputs:
    %  A       : The cell array of m symmetric matrices of size n x n;
    %  b       : The vector of rhs in R^m, and b <= 0;
    %  c       : The objective coefficient matrix of size n x n;
    %  epsilon: An accuracy for safe mode, its default value is eps
    %  maxiter: The maximum number of iterations.
    % Outputs:
    %   Ared, bred, cred: The data of the problem after preprocessing
    %   info: A structure containing the information of preprocessing

    if nargin < 5, maxiter = intmax;    end
    if nargin < 4, epsilon = eps; end
    sqrtEPS = sqrt(epsilon);

    Ared = []; bred = []; cred = [];
    n    = size(c, 1); m = length(b);
    I    = true(n, m + 1); % initial nonzero indices of each constraint
    for i = 1:m, I(:, i) = any(A{i}, 2); end
    I    = sparse(I);

    undeleted       = ones(m, 1); % Keep track of deleted constraints
    not_done        = 1;          % not_done = 1 means preprocessing not done
    info.infeasible = 0;          % infeasibility detected?
    info.reduction  = 0;          % any reduction?
    constr_indices  = (1:m);
    constr_num      = m;
    iter            = 0;
    bn = -sqrtEPS*max(1, norm(b, inf)); % b < 0 if b < -sqrt(epsilon)*max{1, ||b||}
    bz = bn*sqrtEPS;         % b = 0 if -epsilon*max{1, ||b||} < b <= 0

    % Preprocessing
    while not_done
        not_done = 0;
        for ii = 1:constr_num
            i    = constr_indices(ii);
```

```
            At   = A{i}(I(:, i), I(:, i));  % get the nonzero submatrix
            Iaux = any(At, 2);
            if find(Iaux == false, 1),
                I(I(:, i), i) = Iaux;
                At            = At(Iaux, Iaux);
            end
            if isempty(At)
                if b(i) < bn, info.infeasible = 1; return; end
                    % Ai=0 and bi<0 => infeasible
                if b(i) > bz, undeleted(i) = 0; continue; end
                    % Ai=0 and bi=0 => reduce
            end
            if b(i) < bn
                [~, pd_check] = chol(At);
                if pd_check == 0, info.infeasible = 1; return; end
                    % Ai pd and bi<0 => infeasible
            else
                if b(i) > bz
                    [~, pd_check] = chol(At);
                    if pd_check == 0        % Ai pd and bi=0 => reduce
                        I(I(:, i), :) = false;
                        undeleted(i)  = 0;
                        not_done      = 1;
                    else
                        [~, nd_check] = chol(-At);
                        if nd_check == 0    % Ai nd and bi=0 => reduce
                            I(I(:, i), :) = false;
                            undeleted(i)  = 0;
                            not_done      = 1;
                        end
                    end
                end
            end
        end
        constr_indices = find(undeleted);
        constr_num     = length(constr_indices);
        iter           = iter + 1;
        if iter > maxiter, not_done = 0; end
    end

    % Doing reduction: Deleted rows/columns are marked in I(:, m + 1)
    % Now do physical deletion
    I_nonzero = I(:, m + 1);
    Ared      = cell(constr_num, 1);
    for ii = 1:constr_num
        i        = constr_indices(ii);
        Ared{ii} = A{i}(I_nonzero, I_nonzero);
    end
    bred = b(constr_indices);
    cred = c(I_nonzero, I_nonzero);
    if (nnz(I_nonzero) < n) || (constr_num < m), info.reduction = 1; end
end
```

C The DIMACS errors

For the sake of completeness in this section we describe the DIMACS errors, which are commonly used to measure the accuracy of an approximate solution X of (P) and of y of (D).

Define the operator $\mathcal{A} : \mathbb{R}^m \to \mathcal{S}^n$ and its adjoint as

$$\mathcal{A}(X) = (A_1 \bullet X, \ldots, A_m \bullet X), \tag{C.8}$$

$$\mathcal{A}^*(y) = \sum_{i=1}^{m} y_i A_i. \tag{C.9}$$

Suppose we are given an approximate solution X of (P) and an approximate solution y of (D). For brevity, define $Z = C - \mathcal{A}^*(y)$.

Then the DIMACS error measures are defined as follows:

$$\mathrm{err}_1 = \frac{\|\mathcal{A}(X) - b\|_2}{1 + \|b\|_\infty} \tag{C.10}$$

$$\mathrm{err}_2 = \max\left\{0, \frac{-\lambda_{\min}(X)}{1 + \|b\|_\infty}\right\} \tag{C.11}$$

$$\mathrm{err}_3 = \frac{\|\mathcal{A}^*(y) - C - Z\|_F}{1 + \|C\|_\infty} \tag{C.12}$$

$$\mathrm{err}_4 = \max\left\{0, \frac{-\lambda_{\min}(Z)}{1 + \|C\|_\infty}\right\} \tag{C.13}$$

$$\mathrm{err}_5 = \frac{b^T y - C \bullet X}{1 + |C \bullet X| + |b^T y|} \tag{C.14}$$

$$\mathrm{err}_6 = \frac{Z \bullet X}{1 + |C \bullet X| + |b^T y|} \tag{C.15}$$

In the above equations we use the following notation. If $M = (m_{ij}) \in \mathcal{S}^n$, then we write $\|M\|_F$ for the Frobenius norm of M and $\|M\|_\infty$ for the infinity norm of M, i.e.,

$$\|M\|_F = \sqrt{\sum_{i,j} m_{ij}^2}$$

$$\|M\|_\infty = \max_{i,j} |m_{ij}|.$$

We also write $\lambda_{\min}(M)$ for the smallest eigenvalue of M.

Acknowledgements The third author, Quoc Tran-Dinh, is supported in part by the NSF-grant No. DMS-1619884, USA. We are very grateful to to Erling Andersen at Mosek for running several SDPs, and explaining the results; to Joachim Dahl at Mosek for helpful discussions on converting SDPs, and for providing his conversion code; to Didier Henrion and Kim Chuan Toh for providing us with the Henrion-Toh dataset; to Frank Permenter for helpful comments; and to Hans Mittellmann for helping us with some of the large scale SDPs. We also thank Oktay Günlük for helping us to find the name Sieve-SDP.

References

[1] MOSEK ApS. Mosek optimization toolbox for matlab 8.0.0.94, 2017. 1

[2] Victor Baston. Extreme copositive quadratic forms. *Acta Arithmetica*, 15(3):319–327, 1969. 6

[3] Jonathan M. Borwein and Henry Wolkowicz. Facial reduction for a cone-convex programming problem. *J. Aust. Math. Soc.*, 30:369–380, 1981. 4

[4] Jonathan M. Borwein and Henry Wolkowicz. Regularizing the abstract convex program. *J. Math. Anal. App.*, 83:495–530, 1981. 2, 4

[5] Stephen Boyd, Mark T Mueller, Brendan O?Donoghue, Yang Wang, et al. Performance bounds and suboptimal policies for multi–period investment. *Foundations and Trends® in Optimization*, 1(1):1–72, 2014. 6, 9, 10, 12

[6] Samuel Burer and Renato DC Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003. 1

[7] Samuel Burer, Renato DC Monteiro, and Yin Zhang. Solving a class of semidefinite programs via nonlinear programming. *Mathematical Programming*, 93(1):97–122, 2002. 1

[8] Sam Burton, Cynthia Vinzant, and Yewon Youm. A real stable extension of the vamos matroid polynomial. *arXiv preprint arXiv:1411.2038*, 2014. 6

[9] Vris Cheung and Henry Wolkowicz. Sensitivity analysis of semidefinite programs without strong duality. Technical report, Optimization Online, 2014. 9

[10] Vris Cheung, Henry Wolkowicz, and Simon Schurr. Preprocessing and regularization for degenerate semidefinite programs. In David Bailey, Heinz H. Bauschke, Frank Garvan, Michel Théra, Jon D. Vanderwerff, and Henry Wolkowicz, editors, *Proceedings of Jonfest: a conference in honour of the 60th birthday of Jon Borwein*. Springer, 2013. 6, 9, 11

[11] Palahenedi Hewage Diananda. On non-negative forms in real variables some or all of which are non-negative. In *Proc. Cambridge Philos. Soc*, volume 58, pages 17–25, 1962. 6

[12] Dimitry Drusviyatsky, Gábor, and Henry Wolkowicz. Coordinate shadows of semi-definite and euclidean distance matrices. *SIAM J. Opt.*, 25(2):1160–1178, 2015. 2, 4

[13] Dmitriy Drusvyatskiy, Nathan Krislock, Yuen-Lam Voronin, and Henry Wolkowicz. Noisy euclidean distance realization: robust facial reduction and the pareto frontier. *arXiv preprint arXiv:1410.6852*, 2014. 2, 4

[14] Hamza Fawzi and Pablo A Parrilo. Self-scaled bounds for atomic cone ranks: applications to nonnegative rank and cp-rank. *Mathematical Programming*, 158(1-2):417–465, 2016. 6

[15] Henrik Friberg. Facial reduction heuristics and the motivational example of mixed integer conic optimization. Technical report, Optimization Online, 2016. 4

[16] K. Fujisawa, M. Fukuda, M. Kojima, K. Nakata, M. Nakata, and M. Yamashita. Sdpa (semidefinite programming algorithm) and sdpa-gmp user's manual – version 7.1.0. *Department of Mathematical and Computing Sciences, Tokyo Institute of Technology. Research Reports on Mathematical and Computing Sciences Series B-448*, 2008. 1, 4, 12, 14

[17] Katsuki Fujisawa, Masakazu Kojima, Kazuhide Nakata, and Makoto Yamashita. SDPA (semidefinite programming algorithm) user's manual version 6.2. 0. *Department of Mathematical and Computing Sciences, Tokyo Institute of Technology. Research Reports on Mathematical and Computing Sciences Series B: Operations Research*, 2002. 1

[18] Didier Henrion, Simone Naldi, and Mohab Safey ed Din. Exact algorithms for linear matrix inequalities. *SIAM J. Opt.*, 26(4):2512–2539, 2016. 4, 10

[19] Michal Kočvara and Michael Stingl. Pennon: A code for convex nonlinear and semidefinite programming. *Optimization methods and software*, 18(3):317–333, 2003. 1

[20] Nathan Krislock and Henry Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM J. Opt.*, 20:2679–2708, 2010. 2, 4

[21] Minghui Liu and Gábor Pataki. Exact duals and short certificates of infeasibility and weak infeasibility in conic linear programming. *Math. Program. Ser. A*, page to appear, 2017. 4, 10

[22] Yanli Liu, Ernest K Ryu, and Wotao Yin. A new use of Douglas-Rachford splitting and admm for identifying infeasible, unbounded, and pathological conic programs. *arXiv preprint arXiv:1706.02374*, 2017. 10

[23] Hans D Mittelmann. An independent benchmarking of sdp and socp solvers. *Mathematical Programming*, 95(2):407–430, 2003. 5, 7

[24] Gábor Pataki. The geometry of semidefinite programming. In Romesh Saigal, Lieven Vandenberghe, and Henry Wolkowicz, editors, *Handbook of semidefinite programming*. Kluwer Academic Publishers, also available from www.unc.edu/˜pataki, 2000. 3

[25] Gábor Pataki. A simple derivation of a facial reduction algorithm and extended dual systems. Technical report, Columbia University, 2000. 2, 4

[26] Gábor Pataki. Strong duality in conic linear programming: facial reduction and extended duals. In David Bailey, Heinz H. Bauschke, Frank Garvan, Michel Théra, Jon D. Vanderwerff, and Henry Wolkowicz, editors, *Proceedings of Jonfest: a conference in honour of the 60th birthday of Jon Borwein*. Springer, also available from http://arxiv.org/abs/1301.7717, 2013. 2, 4

[27] Gábor Pataki and H. Schmieta Stefan. The DIMACS library of mixed semidefinite-quadratic-linear programs. 6

[28] Frank Permenter and Pablo Parrilo. Partial facial reduction: simplified, equivalent sdps via approximations of the psd cone, to appear, mathematical programming a. Technical report, http://arxiv.org/abs/1408.4685, 2014. 2, 4, 5, 6, 7, 9, 10

[29] Michael Posa, Mark Tobenkin, and Russ Tedrake. Lyapunov analysis of rigid body systems with impacts and friction via sums-of-squares. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 63–72. ACM, 2013. 6

[30] Arie J Quist, Etienne de Klerk, Cornelis Roos, and Tamas Terlaky. Copositive realxation for genera quadratic programming. *Optimization methods and software*, 9(1-3):185–208, 1998. 6

[31] Jos Sturm. Using sedumi 1.02, a matlab toolbox for optimiation over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999. 1

[32] Shin-ichi Tanigawa. Singularity degree of the positive semidefinite matrix completion problem. *SIAM Journal on Optimization*, 27(2):986–1009, 2017. 4

[33] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997. 4

[34] Levent Tunçel. *Polyhedral and Semidefinite Programming Methods in Combinatorial Optimization*. Fields Institute Monographs, 2011. 2, 4

[35] Reha H. Tutuncu, Kim C. Toh, and Michael J. Todd. Solving semidefinite-quadratic-linear programming using sdpt3. *Mathematical Programming Ser. B*, 95:189–217, 2003. 1

[36] David G Wagner and Yehua Wei. A criterion for the half-plane property. *Discrete Mathematics*, 309(6):1385–1390, 2009. 6

[37] Hayato Waki. How to generate weakly infeasible semidefinite programs via Lasserre's relaxations for polynomial optimization. *Optim. Lett.*, 6(8):1883–1896, 2012. 6, 9, 10

[38] Hayato Waki and Masakazu Muramatsu. Facial reduction algorithms for conic optimization problems. *J. Optim. Theory Appl.*, 158(1):188–215, 2013. 2, 4

[39] Hayato Waki, Maho Nakata, and Masakazu Muramatsu. Strange behaviors of interior-point methods for solving semidefinite programming problems in polynomial optimization. *Computational Optimization and Applications*, pages 1–22, 2012. 5, 6, 9, 12

[40] Xin-Yuan Zhao, Defeng Sun, and Kim-Chuan Toh. A newton-cg augmented lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, 20(4):1737–1765, 2010. 1