A Preprocessing Algorithm for Semidefinite Programming

Preston Faulk

April 5, 2016

Abstract

We describe and implement a preprocessing algorithm in MATLAB, FP, for semidefinite programming which is capable of detecting infeasibility or detecting and removing redundancy. The reduction algorithm results in improved performance using an SDP solver, SeDuMi, as measured by DIMACS error bounds and timing. We compare FP to another preprocessing algorithm, from [4], by comparing results on two sets of SDP's: PP and Henrion. [4] performs better than FP on the PP set but FP performs better than [4] on the Henrion set.

1 Intro to Semidefinite Programming

1.1 Review of Linear Programming

Before we define Semidefinite programming, we will begin by discussing linear programming because semidefinite programming follows as a natural extension of linear programming. From this point forward, we will refer to linear programming and linear programs as LP's and semidefinite programming and semidefinte programs as SDP's. The canonical form of an LP is:

$$\begin{aligned} &\min\ c'x \\ &\text{s.t. } Ax = b \\ &x \geq 0 \end{aligned}$$

Where $A \in \mathbf{R}^{m\times n}$, and $b \in \mathbf{R}^n$. x is a vector of decision variables, A is a matrix of parameters and, c, and b are vectors of parameters. In this formulation, solving the LP involves minimizing the objective function: $f(x) = c'x$ subject to constraints on the variables specified by A. There are several algorithms for solving LP's such as the Simplex Method and Interior Point Methods which are highly efficient. LP's can be applied to various phenomena from network optimization, shortest path determination, and resource allocation. However, LP's are limited in the problems they can model. Often, a problem can be best modeled using an SDP.

1.2 Definition of Positive Semidefinite Matrices

Before defining an SDP, it is necessary to first define the notion of positive (semi)defiinte matrices.

Definition: A symmetric matrix $A \in \mathbf{R}^{n \times n}$ is called **positive semidefinite** if $\forall x \in \mathbf{R}^n : x^T Ax \geq 0$. Similarly, A symmetric matrix $A \in \mathbf{R}^{n \times n}$ is called **positive definite** if $\forall x \in \mathbf{R}^n : x^T Ax > 0$ with $x \neq \vec{0}$

However, another equivalent definition which we will utilize during our algorithm is:

Definition: A symmetric matrix $A \in \mathbf{R}^{n \times n}$ is called **positive definite** if its cholesky factorization is unique.

1.3 Review of Semidefinite Programming

Now that we have a definition of PSD matrices, we can make a formal definition of an SDP in standard form:

$$\begin{gathered} min\ C \bullet X \\ s.t.\ A_i \bullet X = b_i \text{ for } i = 1, ..., m \\ X \succeq 0 \end{gathered}$$

Where $A, C, X \in \mathbf{R}^{n \times n}$, and $b \in \mathbf{R}^m$. A, C, and b represent the "data" of the problem and X represents the decision variables. The inner product $\bullet$ is defined as:

$$C \bullet X = \sum_{i,j}^{n} C_{i,j} X_{i,j}$$

SDP's are important because many problems in operations research can be formulated as SDP's.

1.3.1 SDP as a generalization of LP

The formulation of SDP's extend naturally from LP's. In LP's, the goal is to optimize a linear objective function over variables. In SDP's the goal is to optimize a linear objective function over vectors of variables. In SDP's, the PSD constraint replaces the non-negative constraint from LP's. In fact, LP's can be formulates as SDP's if each A_i and C are diagonal matrices where the diagonal elements of each A_i are the elememts from the i^{th} row of A from the LP formulation and the diagonal of C in the SDP is replaced with C from the LP. The PSD constraint on X is satisfied because a diagonal matrix with nonnegative diagonal entries has all nonnegative eigenvalues hence X is always positive semidefinite.

1.3.2 Examples

To demonstrate, we will present an SDP. Consider the SDP defined by:

$$A_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix} \; A_2 = \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} \; C = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix} \; b = \begin{bmatrix} 11 \\ 19 \end{bmatrix}$$

In the canonical form described above, this SDP looks like:

$$\begin{gathered} min \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix} \bullet X \\ subject\ to: \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix} \bullet X = 11 \\ \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} \bullet X = 19 \\ X \succeq 0 \end{gathered}$$

This SDP is equivalent to the following which resembles the form of an LP with the exception that the nonnegative constraint is replaced with the positive semidefiniteness constraint.

$$\begin{gathered} min\ x_{1,1} + 4x_{1,2} + 6x_{1,3} + 9x_{2,2} + 7x_{3,3} \\ \text{s.t. } x_{1,1} + 2x_{1,3} + 6x_{2,2} + 14x_{2,3} + 5x_{3,3} = 11 \\ 4x_{1,2} + 16_{1,3} + 6x_{2,2} + 4x_{3,3} = 19 \\ \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \succeq 0 \end{gathered}$$

1.3.3 SeDuMi

LP's have very efficient algorithms for finding solutions; namely the simplex method and interior point methods. There are many solvers designed to solve LP's. There are also several solvers for SDP's as well. The one which will be discussed and utilized during our discussion will be SeDuMi. Sedumi is an add-on for MATLAB, which lets you solve optimization with linear, quadratic, and semidefiniteness constraints. Specifically, we will be utilizing the semidefiniteness constraints as a solver for SDP's [5]. To demonstrate how SeDuMi works for SDP's, we will solve the example problem from the previous section using SeDuMi. In order to call SeDuMi on the SDP we must formulate it into SeDuMi format. SeDuMi format for SDP's consists of arrays representing the data from the A_i's, b, and c as well as an argument, K, whose fields contain the semidefinte constraints. A is an array whose i^{th} row contains the entries of A_i. b is a vector containing the m values for each b_i. c is a vectore containg the entries for C from the SDP. K can have the fields f, q, l, and s which stand for free, quadratic, lorentz, and semidefinite respectively. In our example, the data is presented in SeDuMi format here:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 3 & 7 & 1 & 7 & 5 \\ 0 & 2 & 8 & 2 & 6 & 0 & 8 & 0 & 4 \end{bmatrix}$$
$$b = \begin{bmatrix} 11 \\ 19 \end{bmatrix}$$
$$c = \begin{bmatrix} 1 & 2 & 3 & 2 & 9 & 0 & 3 & 0 & 7 \end{bmatrix}$$
$$K.s = 3$$

Then in MATLAB, the call to SeDuMi is:

```
[x,y,info] = sedumi(A,b,c,K);
```

Which produces the following output:

```
SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, Adaptive Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 2, order n = 4, dim = 10, blocks = 2
nnz(A) = 9 + 0, nnz(ADA) = 4, nnz(L) = 3
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            3.92E+02 0.000
  1 :   7.19E+00 2.87E+01 0.000 0.0733 0.9900 0.9900   1.52  1  1  1.6E+00
  2 :   1.29E+01 5.71E+00 0.000 0.1987 0.9000 0.9000   1.55  1  1  2.9E-01
  3 :   1.38E+01 1.46E+00 0.000 0.2555 0.9000 0.9000   1.23  1  1  6.9E-02
  4 :   1.39E+01 3.10E-02 0.000 0.0212 0.9900 0.9900   1.09  1  1  1.4E-03
  5 :   1.39E+01 6.78E-04 0.000 0.0219 0.9900 0.9900   1.00  1  1  3.0E-05
  6 :   1.39E+01 5.49E-05 0.357 0.0810 0.9900 0.9900   1.00  1  1  2.4E-06
  7 :   1.39E+01 1.53E-05 0.058 0.2791 0.9000 0.9245   0.99  1  1  6.7E-07
  8 :   1.39E+01 5.63E-07 0.161 0.0367 0.9900 0.9900   1.00  1  1  2.3E-08
  9 :   1.39E+01 9.95E-08 0.000 0.1768 0.9000 0.9092   1.00  2  2  4.2E-09

iter seconds digits       c*x               b*y
  9      0.1   8.4  1.3902227871e+01  1.3902227821e+01
|Ax-b| =   3.3e-08, [Ay-c]_+ =   0.0E+00, |x|=  1.9e+00, |y|=  6.6e-01

Detailed timing (sec)
   Pre          IPM          Post
5.995E-03    9.700E-02    2.002E-03
Max-norms: ||b||=19, ||c|| = 9,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 1.72745.
x =

    1.0559
    0.3692
    0.8683
    0.3692
```

```
    0.1291
    0.3036
    0.8683
    0.3036
    0.7140


y =

    0.4847
    0.4511


info =

         iter: 9
    feasratio: 0.9975
         pinf: 0
         dinf: 0
       numerr: 0
       timing: [0.0070 0.0970 0.0020]
      wallsec: 0.1060
       cpusec: 0.2184
```

As is evident, SeDuMi returns x, y, and $info$. x and y are solutions to the primal and dual SDP respictively in vector format rather than matrix. Info contains information relevant to the performace of the solver on this particular instance. Iter is the number of iterations performed. feasratio is a measure of feasibility determined by SeDuMi. A feasratio close to -1 indicates infeasibility, and a feasratio close to 1 indicates feasibility. pinf and dinf are indicators of feasibility in the primal and dual respectively. numerr is an indicator for whether SeDuMi encountered numerical error. Then timing, wallsec, and cpusec are measures of different timing information. Much of this information will be relevant in our later discussion and analysis.

2 Preprocessing Algorithms

SeDuMi is usually reliable for solving a variety of problems in an efficient manor. But sometimes with large problems or special circumstances SeDuMi has trouble either with accuracy or with timing. So the implementation of algorithms which can reduce the size of the problem or detect infeasibility before running SeDuMi are valuable from a practical standpoint. We will present our preprocessing algorithm and compare it to an existing preprocessing algorithm from Permenter and Parrilo [4].

2.1 Our Algorithm

To motivate our algorithm, we will present an example to demonstrate. Consider the following SDP system which is trivially infeasible:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \bullet X = 0$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \bullet X = -1$$
$$X \succeq 0$$

To see that it is infeasible, suppose that $X = (x_{i,j})^3_{i,j=1}$ is feasible in it. Then $x_{1,1} = 0$; hence the first row and column of X are zero by psdness, so the second constraint implies $x_{2,2} = -1$, which is a contradiction. Here, the internal structure of the system itself proves its infeasibility.

This underlying structure can be found in SDP's of larger scale. For example consider the following SDP system from [8]:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \bullet X = 0 \ ,$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \bullet X = 0 \ ,$$
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \bullet X = 1 \ ,$$
$$X \succeq 0$$

From the first constraint we can conclude that $x_{2,2} = 0$ and furthermore the second row and column of X are all zero. Hence we can delete the second row and column from the problem altogether which results in the following reduced SDP:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \bullet X = 0 \ ,$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \bullet X = 0 \ ,$$
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \bullet X = 1 \ ,$$
$$X \succeq 0$$

From here, there are two things to notice. One is that from the second constraint using the same logic as before, we could delete the first row and column from the problem. However something more convenient also presents itself. In the third constraint we have something impossible, no X can satisfy this constraint hence this SDP is infeasible.

2.2 General Explanation

Since SDP's given to SeDuMi are given in SeDuMi format, our algorithm begins by formatting it from SeDuMi format to a form which is easier for our algorithm to handle. We begin by taking each row of A in SeDuMi format and reshaping it into a square matrix which is exactly the A_i from the standard form we defined earlier. Then we store each A_i in a cell array in MATLAB. Then once they are in this convenient form, our algorithm involves checking each A_i to see if it has the structure:

$$\begin{bmatrix} A_i' & 0 \\ 0 & 0 \end{bmatrix} \bullet X = b_i$$

Where A_i' is positive definite or negative definite as determined by the Cholesky factorization. If $A_i' \succ 0$ and $b_i < 0$ or $A_i' \prec 0$ and $b_i > 0$ we declare infeasible. If $A_i' \succ 0$ and $b_i = 0$ or $A_i' \prec 0$ and $b_i = 0$ we delete those rows and columns from each A_i corresponding to those in A_i' and return to the beginning and continue to seek reductions or infeasibility.

2.3 Pseudocode

In order to see this more explicitly, I have included the following pseudocode which describes the algorithm:

```
while (not done)
    for i=1:m
        A_i' := A_i with zero rows/columns deleted
        if A_i' ≻ 0 and b_i < 0
            declare infeasible, break
        end
        if A_i' ≺ 0 and b_i > 0
            declare infeasible, break
        end
        if A_i' ≻ 0 and b_i == 0
            delete rows/columns of A_i' from each A_i, go to beginning
        end
        if A_i' ≺ 0 and b_i == 0
            delete rows/columns of A_i' from each A_i, go to beginning
        end
```

```
    end
end
```

This does not include the code for taking the SDP in SeDuMi format and converting it to a form which is suitable for the algorithm or converting it back into SeDuMi. In order to determine whether A_i' is positive or negative definiite we use the cholesky factorization. So for example, to check if $A \succ 0$ or $A \prec 0$ we call the MATLAB cholesky function as below:

```
[~,pdcheck] = chol(A);
or
[~,ndcheck] = chol(-A);
```

If pdcheck $= 0$ then $A \succ 0$, if ndcheck $= 0$ then $A \prec 0$. Also all arrays are kept in sparse format to reduce memory usage. The actual MATLAB code can be found online at (link add later).

3 Application of Algorithm

In order to evaluate the effectiveness of our algorithm we tested it on several families of instances. We look at several measurements to determine effectiveness including timing, DIMACS error bounds, and size of the instance before and after preprocessing. We will often be comparing our results to the results of [4] on common instances.

3.1 Description of Examples

The set of examples we will call the PP set includes 56 instances described in [4]. They originate from [8],[7],[2],[1],[6], and [3]. The set of examples we will call the Henrion set includes 90 instances from Didier Henrion.

3.2 Results

The results will focus on comparisons between our algorithm (FP) and [4]'s algorithm with diagonal and diagonally dominant approximations, denoted 'd' and 'dd' respectively. 'd' is faster than 'dd' but usually results in fewer reductions as described in their paper. We will consider a number of metrics. In order to see how well each algorithm reduces problems, we will often look at n_pre and n_post for each example. Here, n represents the size of each square A_i before and after reductions. For feasibility, we will consider SeDuMi's output called feasratio which as described earlier inidicated infeasibility if close to -1 and feasibility if close to 1. Then for accuracy of SeDuMi, we will consider the DIMACS error bounds which are described in (source for dimacs). These error bounds are included in SeDuMi output. We will always consider the greatest such error bound for each instance. As before this will be considered before and after preprocessing. Along with the DIMACS error bounds, we will define the criteria for what

it means for preprocessing to help an instance, this will be described in detail later. Finally we will consider timing. This will mean comparing the timing of SeDuMi before preprocessing, timing of preprocessing, and timing of SeDuMi after preprocessing. More details and explanation will follow in the discussion of the results.

The following table summarizes the results for FP vs PP concerning reductions of n and feasibility. We used 'd' or 'dd' according to the specifications in [4]. For those instances in which we detected infeasibility, n post FP reflects n at the detection of infeasibility during the algorithm.

Instance Name	Src.	n Pre	n Post FP	n Post PP	FP find inf	Sedumi inf Pre	Sedumi inf Post FP	Sedumi inf Post PP
Example1	[8]	3	2	2	no	1.0000	1.0000	1.0000
Example2	[8]	3	2	2	no	0.0000	1.0000	1.0000
Example3	[8]	3	2	2	no	0.4108	0.5517	0.5517
Example4	[8]	3	1	1	yes	-0.4351	-	-1.0000
Example5	[8]	10	10	10	no	0.9990	0.9990	0.9990
Example6	[8]	8	5	5	no	1.0011	1.0000	1.0000
Example7	[8]	5	4	4	no	1.0000	1.0000	1.0000
Example9size20	[8]	20	19	1	yes	-1.0130	-	-1.0000
Example9size100	[8]	100	99	1	yes	-1.0066	-	-1.0000
CompactDim2r1	[7]	6	5	1	yes	0.2004	-	-.9942
CompactDim2r2	[7]	15	9	3	yes	-0.1245	-	-.9942
CompactDim2r3	[7]	28	13	3	yes	0.8687	-	-.9942
CompactDim2r4	[7]	45	17	3	yes	1.0209	-	-.9942
CompactDim2r5	[7]	66	21	3	yes	1.0135	-	-.9942
CompactDim2r6	[7]	91	25	3	yes	1.0072	-	-.9942
CompactDim2r7	[7]	120	29	3	yes	0.9995	-	-.9942
CompactDim2r8	[7]	153	33	3	yes	1.1192	-	-.9942
CompactDim2r9	[7]	190	37	3	yes	0.9969	-	-.9942
CompactDim2r10	[7]	231	41	3	yes	0.9979	-	-.9942
unboundDim1r1	[2]	4	2	2	no	1.0000	1.0000	1.0000
unboundDim1r2	[2]	7	2	2	no	0.2753	1.0000	1.0000
unboundDim1r3	[2]	10	2	2	no	0.2365	1.0000	1.0000
unboundDim1r4	[2]	13	2	2	no	-0.5195	1.0000	1.0000
unboundDim1r5	[2]	16	2	2	no	0.9716	1.0000	1.0000
unboundDim1r6	[2]	19	2	2	no	1.0310	1.0000	1.0000
unboundDim1r7	[2]	22	2	2	no	1.0999	1.0000	1.0000
unboundDim1r8	[2]	25	2	2	no	1.0364	1.0000	1.0000
unboundDim1r9	[2]	28	2	2	no	1.1254	1.0000	1.0000
unboundDim1r10	[2]	31	2	2	no	1.0366	1.0000	1.0000
horn2	[1]	4	4	2	no	1.0000	1.0000	1.0000
horn3	[1]	10	10	6	no	1.0102	1.0102	1.0126
horn4	[1]	20	20	14	no	1.0296	1.0296	.9924
horn5	[1]	35	35	25	no	1.0432	1.0432	1.0823
hornD2	[1]	4	4	2	no	1.0000	1.0000	1.0000
hornD3	[1]	10	10	6	no	0.9935	0.9935	1.1106
hornD4	[1]	20	20	14	no	0.9112	0.9112	.8834
hornD5	[1]	35	35	25	no	1.0360	1.0360	1.0596
vamos534	[6]	52	52	41	no	0.9854	0.9854	1.0002
weiwagnerF7minus4	[6]	8	8	5	no	1.0000	1.0000	1.1192
weiwagnernPminus124	[6]	12	12	6	no	1.1050	1.1050	1.0002
weiwagnernPminus912	[6]	12	12	5	no	0.9988	0.9988	1.0000
weiwagnerP7	[6]	8	8	4	no	0.9641	0.9641	1.0004
weiwagnervamos12	[6]	16	16	13	no	0.9998	0.9998	1.0003
weiwagnerW3PlusE	[6]	9	9	5	no	0.9865	0.9865	1.0000
weiwagnerW3Plus	[6]	8	8	3	no	0.8628	0.8628	1.0000
hinf12	[3]	24	24	14	no	.4690	.4690	.3110
hinf13	[3]	30	30	17	no	.8006	.8006	.8006

The following table summarizes the results for the maximum DIMACS error bounds before and after preprocessing for FP vs PP on the PP set. Once again, 'd' or 'dd' was chosen according to [4]

Instance Name	Src.	Max Error Pre	PP Max Error Post	FP Max Error Post
Example1	[8]	1.6362e-15	7.7605e-13	7.7605e-13
Example2	[8]	8.1400e-02	8.3926e-13e-13	8.3826e-13
Example3	[8]	1.7897e-05	2.3856e-05	2.3856e-05
Example4	[8]	5.0000e-01	triv inf	-
Example5	[8]	1.0922e-10	-	-
Example6	[8]	1.9796e-09	1.9796e-09	8.3803e-13
Example7	[8]	5.4940e-12	604599e-13	6.4599e-13
Example9size20	[8]	0.5000e-01	2.3776e-13	-
Example9size100	[8]	0.5000e-01	1.9997e-13	-
CompactDim2r1	[7]	3.9390e-01	5.0000e-01	-
CompactDim2r2	[7]	3.0580e-01	5.0000e-01	-
CompactDim2r3	[7]	4.9620e-08	5.0000e-01	-
CompactDim2r4	[7]	2.6701e-08	5.0000e-01	-
CompactDim2r5	[7]	6.5196e-08	5.0000e-01	-
CompactDim2r6	[7]	3.3071e-08	5.0000e-01	-
CompactDim2r7	[7]	3.5925e-07	5.0000e-01	-
CompactDim2r8	[7]	1.2397e-07	5.0000e-01	-
CompactDim2r9	[7]	3.7499e-07	5.0000e-01	-
CompactDim2r10	[7]	1.4372e-07	5.0000e-01	-
unboundDim1r1	[2]	6.8724e-13	6.8724e-13	3.7469e-13
unboundDim1r2	[2]	1.7000e-03	3.7448e-13	3.7448e-13
unboundDim1r3	[2]	3.6800e-02	3.7448e-13	3.7448e-13
unboundDim1r4	[2]	2.2700e-02	3.7448e-13	3.7448e-13
unboundDim1r5	[2]	2.2164e-08	3.7448e-13	3.7448e-13
unboundDim1r6	[2]	1.2919e-08	3.7448e-13	3.7448e-13
unboundDim1r7	[2]	1.3228e-08	3.7448e-13	3.7448e-13
unboundDim1r8	[2]	2.2885e-08	3.7448e-13	3.7448e-13
unboundDim1r9	[2]	2.2225e-08	3.7448e-13	3.7448e-13
unboundDim1r10	[2]	4.1245e-08	3.7448e-13	3.7448e-13
horn2	[1]	3.6465e-13	1.2561e-15	-
horn3	[1]	1.4302e-08	5.7441e-12	-
horn4	[1]	3.4129e-08	1.2340e-08	-
horn5	[1]	1.4071e-07	1.1803e-08	-
hornD2	[1]	4.0856e-14	5.8422e-12	-
hornD3	[1]	1.0738e-11	1.4893e-11	-
hornD4	[1]	1.8433e-10	5.7353e-10	-
hornD5	[1]	4.7737e-10	1.7059e-10	-
vamos534	[6]	1.6726e-09	1.2079e-09	-
weiwagnerF7minus4	[6]	1.1020e-12	2.5532e-09	-
weiwagnernPminus124	[6]	1.6584e-08	8.7937e-12	-
weiwagnernPminus912	[6]	7.6937e-09	1.5104e-12	-
weiwagnerP7	[6]	8.5155e-10	3.2403e-12	-
weiwagnervamos12	[6]	1.3060e-09	1.0943e-12	-
weiwagnerW3PlusE	[6]	1.6578e-10	6.6802e-15	-
weiwagnerW3Plus	[6]	4.9601e-09	2.5074e-13	-
hinf 12	[3]	2.5320e-01	2.9130e-01	-
hinf 13	[3]	2.2500e-02	can't replicate	-

The following table summarizes the overall results for all 146 instances with respect to reductions, infeasibility detection, and Help. We say we Helped an instance if the maximum DIMACS error for sedumi before preprocessing is greater than 10^{-6} and the maximum DIMACS error for sedumi after preprocessing is such that: maxErrorPost$< \frac{1}{10}$ maxErrorPre. Of the 146 instances here, 23 were eligible for being helped. Of the 23, 9 were from the Henrion set while 14 were fron the PP set.

	Number Reduced	Reduced %	Infeasibility Detected	Infeasibility %	Helps*	Helps
FP	49 out of 146	33.56%	15 out of 146	10.27%	6 of 146	4.11%
PP-dd	51 out of 146	34.93%	0 out of 146	0.00%	4 out of 146	2.74%
PP-d	41 out of 146	28.08%	0 out of 146	0.00%	4 out of 146	2.74%

The following table summarizes the results for all 90 henrion instances.

	Number Reduced	Reduced %	Infeasibility Detected	Infeasibility %	Helps*	Helps %
FP	17 out of 90	18.89%	2 out of 90	2.22%	1 of 90	1.11%
PP-dd	11 out of 90	12.22%	0 out of 90	0%	0 out of 90	0.00%
PP-d	13 out of 90	14.44%	0 out of 90	0%	0 out of 90	0.00%

The following table summarizes the results for all 56 PP instances.

	Number Reduced	Reduced %	Infeasibility Detected	Infeasibility %	Helps*	Helps %
FP	32 out of 56	57.14%	13 out of 56	23.21%	5 of 56	8.93%
PP-dd	40 out of 56	71.43%	0 out of 56	0%	4 out of 56	7.14%
PP-d	28 out of 56	50.00%	0 out of 56	0%	4 out of 56	7.14%

Next we look at a weighted and unweighted average for reduction percentages. For each instance, let p_i, and the weighted average be defined by:

$$p_i = \frac{n_{pre} - n_{post}}{n_{pre}}$$
$$\text{WA} = \frac{\sum n_{pre_i} * p_i}{\sum n_{pre_i}}$$

The results for these calculations are summarized in the table below:

	WA Total	WA Henrion	WA PP	Avg.	Avg. Henrion	Avg. PP
FP	.1202	.0747	.4259	.1645	.0565	.3380
PP-dd	.0302	.0073	.1838	.1740	.0290	.4072
PP-d	.0544	.0151	.3185	.1653	.0357	.3735

The next table summarizes the time analysis:

	SeDuMi Time w/o Preprocessing	Time with Preprocessing and SeDuMi
FP	1168.3 sec	312.4 sec
PP-dd	1168.3 sec	4144.1 sec
PP-d	1168.3 sec	278.5 sec

The next table shows the reduction in numberof non-zero entries using the same technique as for reduction percentages with a weighted and unweighted average:

	WA Total	WA Henrion	WA PP	Avg.	Avg. Henrion	Avg. PP
FP	.1847	.0629	.1880	.2279	.0779	.4691
PP-dd	.0015	.0184	.0011	.2250	.0481	.5093
PP-d	.0039	.0456	.0028	.2105	.0602	.4522

The final table shows for each preprocessing algorithm, how many times the optimal value before and after preprocessing differs by more than 10^{-5}:

	Occurances
FP	16
PP-dd	22
PP-d	24

4 Conclusion

Overall the results were comparable to [4]. From the PP set, in the instances from [8], [7], and [2] we either matched Permenter/Parrilo on reductions or declared infeasibility. In [1] ,[6], and [3] however we do not find any reductions when Permenter/Parrilo do. Looking at only the Henrion set however, we do better in total number of instances reduced, weighted reduction percentage, unweighted reduction percentage, infeasibility detection, and help percentage. It is clear that Permenter/Parrilo perform well on the instances reported in their paper. However, [8] is a collection of SDP instances yet not all of them appear in [4]. Only the instances where Permenter/Parrilo found reductions were reported. Those instances included in [8] but not in [4] were included in our 56 instances PP set but not in the first two large tables. The Henrion set consists of 90 hard SDP's arising from polynomial optimization. Neither FP nor PP-d nor PP-dd do particularly well on them in comparison to the PP set but FP does better in every category. The most complex part of FP is the cholesky factorization. PP finds reductions by solving several LP's and is much more complex than FP. On their own, both FP and PP do quite well but the most important result is that FP does as well as PP even though FP is a simple reduction algorithm in comparison to PP.

References

[1] Pablo A. Parrilo Grigoriy Blekherman and Rekha R. Thomas. Semidefinite optimization and convex algebraic geometry. *SIAM*, 2013.

[2] Maho Nakata Hayato Waki and Masakazu Muramatsu. Strange behaviors of interior-point methods for solving semidefinite programming problems in polynomial optimization. *Computational Optimization and Applications*, 53(3):823–844, 2012.

[3] Gábor Pataki. The dimacs library of semidefinite-quadratic-linear programs available at http://dimacs.rutgers.edu/challenges/seventh/instances. 1999.

[4] Frank Permenter and Pablo Parrilo. Partial facial reduction: simplified, equivalent sdps via approximations of the psd cone. 2014.

[5] Jos F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization)ver symmetric cones. 1999.

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

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

[8] Simon Schurr Yuen-Lam Cheung and Henry Wolkowicz. Preprocessing and regularization for degenerate semifefinite programs. In *Computational and Analytical Mathematics*, pages 251–303. Springer, 2013.