# Sieve-SDP: A Simple Algorithm to Preprocess Semidefinite Programs

Yuzixuan Zhu

Joint work with Gábor Pataki and Quoc Tran-Dinh

University of North Carolina at Chapel Hill

SIAGA, July 2019

# Outline

- Basic Concepts
- Examples
- The Sieve Algorithm
- Computational Results

# Semidefinite Program (SDP)

$$\begin{aligned} \text{inf.} \;\; & C \cdot X \\ \text{s.t.} \;\; & A_i \cdot X = b_i \; (i = 1, ..., m) \\ & X \succeq 0 \end{aligned}$$

where

- $C, A_i, X \in \mathcal{S}^n, \; b_i \in \mathbb{R}, \; i = 1, ..., m$
- $A \cdot X := \text{trace}(AX) = \sum_{i,j=1}^{n} a_{ij} x_{ij}$
- $X \succeq 0$: $X \in \mathcal{S}_+^n$, i.e. $X$ is symmetric positive semidefinite (psd)

# Motivation

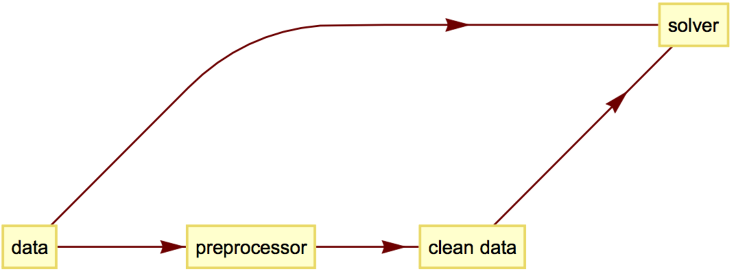
Softwares: SeDuMi, SDPT3, Mosek, ...

- Slow for problems that are large
- Error for problems without strict feasibility

We want to preprocess the problem to

- Reduce size by removing redundancy
- Detect lack of strict feasibility

before giving the problem to the solver.



# Example 1

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

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

$$X \succeq 0$$

# Example 1

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

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

$$X \succeq 0$$

$$\text{Suppose } X = (x_{ij})_{3\times 3} \text{ feasible } \Rightarrow x_{11} = 0$$

$$\Rightarrow x_{12} = x_{13} = 0$$

# Example 1

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

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

$$X \succeq 0$$

$$\begin{aligned} \text{Suppose } X = (x_{ij})_{3\times 3} \text{ feasible } &\Rightarrow x_{11} = 0 \\ &\Rightarrow x_{12} = x_{13} = 0 \\ &\Rightarrow x_{22} = -1 \\ &\Rightarrow \text{Infeasible!} \end{aligned}$$

# Example 2

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 0$$

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

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 1$$

$$X \succeq 0$$

# Example 2

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 0, \quad \text{removed}$$

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

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 1$$

$$X \succeq 0$$

# Example 2

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 0, \quad \text{removed}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \cdot X = 0, \quad \text{removed}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot X = 1$$

$$X \succeq 0$$

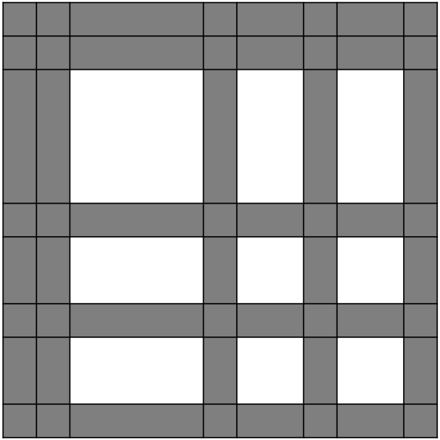
# Example 2

Before preprocessing: $X \in \mathcal{S}_+^4$; 3 constraints

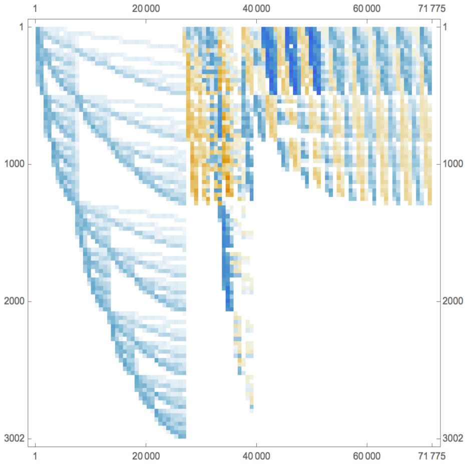
After preprocessing: $X \in \mathcal{S}_+^1$; 1 constraint: $1 \cdot X = 1$

# The Sieve structure

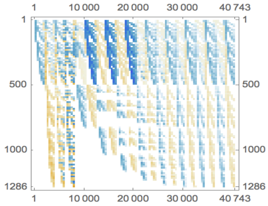
After reduction, the matrix looks like this:



# A large example




(a) An SDP with $m = 3002$ and $\sum n_i^2 = 71775$




(b) Reduced SDP with $m = 1286$ and $\sum n_i^2 = 40743$

# Basic steps

**Step 1.** Find a constraint of the form

$$\begin{pmatrix} D_i & 0 \\ 0 & 0 \end{pmatrix} \cdot X = b_i,$$

where $b_i \leq 0$ and $D_i \succ 0$ (checked by Cholesky factorization).

**Step 2.** If $b_i < 0$, stop. The SDP is infeasible.

**Step 3.** If $b_i = 0$, delete rows and columns corresponding to $D_i$; remove this constraint.

# Sieve-SDP is a facial reduction algorithm (FRA)

- Literature: Borwein-Wolkowicz 1981; Waki-Muramatsu 2013; Pataki 2013.
- The feasible region of an SDP is

$$\{X \in \mathcal{S}^n_+ : \ A_i \cdot X = b_i, \ i = 1, ..., m\},$$

  which is equivalent to

$$\{X \in F : \ A_i \cdot X = b_i, \ i = 1, ..., m\}$$

  for some $F$ face of $\mathcal{S}^n_+$.
- FRA iterates to reduce the cone ($F_{k+1} \subseteq F_k \subseteq \cdots \subseteq \mathcal{S}^n_+$).

# Other motivation: reformulations

Liu-Pataki 2015, 2017: if

$$\begin{aligned} \text{inf.}\ & C \cdot X \\ \text{s.t.}\ & A_i \cdot X = b_i \ (i = 1, ..., m) \\ & X \succeq 0 \end{aligned}$$

is infeasible, then it can be **reformulated** (using eros, and similarity transf) so Sieve-SDP trivially proves infeasibility.

# Other motivation: reformulations

Liu-Pataki 2015, 2017: if

$$\begin{aligned} \text{inf.} \ & C \cdot X \\ \text{s.t.} \ & A_i \cdot X = b_i \ (i = 1, ..., m) \\ & X \succeq 0 \end{aligned}$$

is infeasible, then it can be **reformulated** (using eros, and similarity transf) so Sieve-SDP trivially proves infeasibility.

This work: in many cases we do not even have to reformulate!

# Permenter-Parrilo (PP) preprocessing methods

- PP reduces the size of an SDP by solving linear programming subproblems
- Implemented for primal (p-) and dual (d-) SDPs
- Implemented using diagonal (-d1) and diagonally dominant (-d2) approximations

# Problem sets

Table: 5 datasets consisting of 771 SDP problems.

| dataset | source | # problems |
|---|---|---|
| Permenter-Parrilo (PP) | paper in MPA, 2017 | 68 |
| Mittelmann | Mittelmann website | 31 |
| Dressler-Illiman-de Wolff (DIW) | paper in 2017 | 155 |
| Henrion-Toh | Didier Henrion and Kim-Chuan Toh | 98 |
| Toh-Sun-Yang | papers in SIOPT/MPA | 419 |
| total | | 771 |

# Computational setup

# Computational setup

- Preprocess using Sieve-SDP and four PP methods (pd1, pd2, dd1, dd2).

# Computational setup

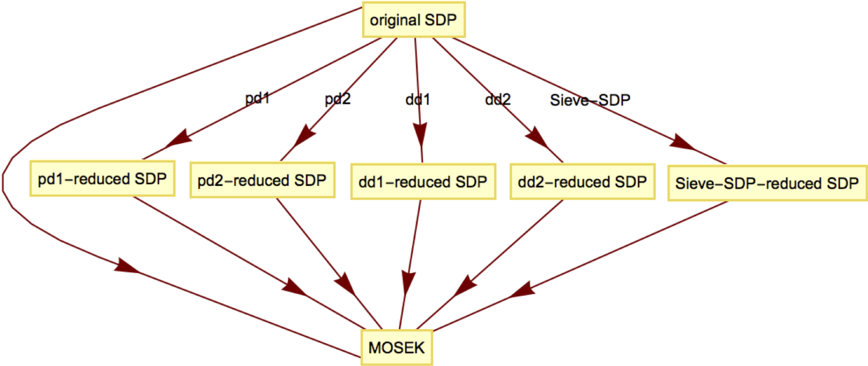
- Preprocess using Sieve-SDP and four PP methods (pd1, pd2, dd1, dd2).
- Use MOSEK to solve each problem before and after preprocessing.

# Computational setup

- Preprocess using Sieve-SDP and four PP methods (pd1, pd2, dd1, dd2).
- Use MOSEK to solve each problem before and after preprocessing.
- MATLAB R2015a on MacBook Pro with 8GB of RAM.

# Computational setup

- Preprocess using Sieve-SDP and four PP methods (pd1, pd2, dd1, dd2).
- Use MOSEK to solve each problem before and after preprocessing.
- MATLAB R2015a on MacBook Pro with 8GB of RAM.



# Comparison criteria

[1] http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/

# Comparison criteria

- Does preprocessing reduce a problem?

[1] `http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/`

# Comparison criteria

- Does preprocessing reduce a problem?
- Does it help to detect infeasibility?

[1] http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/

# Comparison criteria

- Does preprocessing reduce a problem?
- Does it help to detect infeasibility?
- Note: Sieve-SDP detects infeasibility without any optimization solver!

[1] `http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/`

# Comparison criteria

- Does preprocessing reduce a problem?
- Does it help to detect infeasibility?
- Note: Sieve-SDP detects infeasibility without any optimization solver!
- Does it help to recover the true objective value?

[1] `http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/`

# Comparison criteria

- Does preprocessing reduce a problem?
- Does it help to detect infeasibility?
- Note: Sieve-SDP detects infeasibility without any optimization solver!
- Does it help to recover the true objective value?
- Does it reduce solution inaccuracy defined by DIMACS errors[1]?

[1] http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/

# Comparison criteria

- Does preprocessing reduce a problem?
- Does it help to detect infeasibility?
- Note: Sieve-SDP detects infeasibility without any optimization solver!
- Does it help to recover the true objective value?
- Does it reduce solution inaccuracy defined by DIMACS errors[1]?
- Does it reduce solving time?

[1] `http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/`

# Recover true objective values?

Problem set "Compact" from Waki 2012

| problem | correct | w/o prep. | pd1/pd2 | dd1/dd2 | Sieve |
|---|---|---|---|---|---|
| 1 | Inf, $+\infty$ | 3.79e+06, 4.20e+06 | Inf, 1 | 3.79e+06, 4.20e+06 | Inf, - |
| 2 | Inf, $+\infty$ | 6.41e-10, 6.81e-10 | Inf, 2 | 6.41e-10, 6.81e-10 | Inf, - |
| 3 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 4 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 5 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 6 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 7 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 8 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 9 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| 10 | Inf, $+\infty$ | 1.5, 1.5 | Inf, 2 | 1.5, 1.5 | Inf, - |
| correct% | 100%, 100% | 0%, 0% | 100%, 0% | 0%, 0% | 100%, - |

Inf = Infeasible

# Recover true objective values?

Similar results on problem set “unbounded” from
Waki-Muramatsu-Nakata 2012

# Recover true objective values?

Similar results on problem set “unbounded” from Waki-Muramatsu-Nakata 2012

In their paper, they used SDPA-GMP (whic carries a few hundred digits of accuracy) to compute the correct solutions,

# SDP relaxation for polynomial optimization

- Polynomial optimization:

$$\begin{aligned} \min_{x\in\mathbb{R}^N} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \geq 0, \quad i = 1, ..., r. \end{aligned} \tag{poly-opt}$$

# SDP relaxation for polynomial optimization

- Polynomial optimization:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \geq 0, \quad i = 1, ..., r. \end{aligned} \tag{poly-opt}$$

- (poly-opt) has SDP relaxations (Lasserre 2001).

# SDP relaxation for polynomial optimization

- Polynomial optimization:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \geq 0, \quad i = 1, ..., r. \end{aligned} \qquad \text{(poly-opt)}$$

- (poly-opt) has SDP relaxations (Lasserre 2001).
- if SDP is infeasible, then it is useless.

# SDP relaxation for polynomial optimization

- Polynomial optimization:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \geq 0, \quad i = 1, ..., r. \end{aligned} \tag{poly-opt}$$

- (poly-opt) has SDP relaxations (Lasserre 2001).
- if SDP is infeasible, then it is useless.
- Solvers could spend a lot of time (not) proving infeasibility.

# SDP relaxation for polynomial optimization

- Polynomial optimization:

$$\begin{aligned} \min_{x \in \mathbb{R}^N} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \geq 0, \quad i = 1, ..., r. \end{aligned} \tag{poly-opt}$$

- (poly-opt) has SDP relaxations (Lasserre 2001).
- if SDP is infeasible, then it is useless.
- Solvers could spend a lot of time (not) proving infeasibility.
- Sometimes one can compute near feasible solutions: Henrion-Lasserre 2005; Magron-Lasserre 2019.

# Results of DIW dataset (polynomial optimization problems)

Table: Results of DIW dataset.

| prep. method | # reduced | # infeas detected | $n$ | $m$ | $t_{prep} + t_{sol}$ (s) |
|---|---|---|---|---|---|
| w/o prep. | - | - | 53,523 | 186,225 | (39 hrs $\approx$) 139,493 |
| pd1 | 155 | 56 | 1,450 | 3,278 | 1,743 |
| pd2 | 155 | 56 | 1,450 | 3,278 | 10,956 |
| dd1 | 0 | 0 | 53,523 | 186,225 | 139,541 |
| dd2 | 0 | 0 | 53,523 | 186,225 | 161,628 |
| Sieve-SDP | 155 | 59 | 1,385 | 3,204 | (22 min $\approx$) 1,319 |

- Increased the solving speed by more than 100 times!
- Infeasibility has been double-checked in exact arithmetic.

# An example from DIW dataset

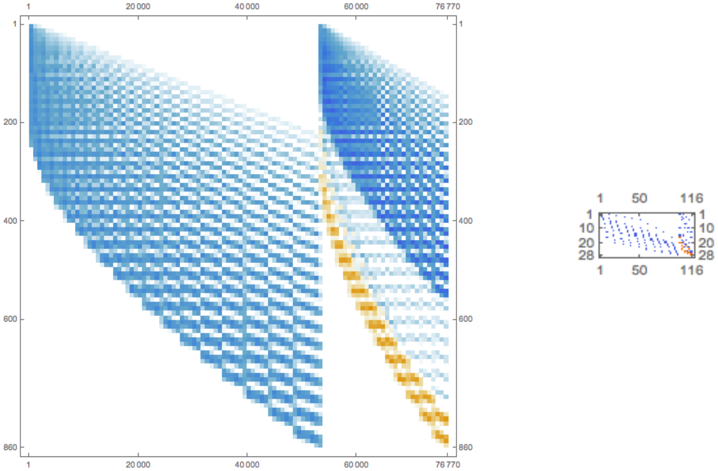



Figure: Size and sparsity before and after Sieve-SDP.

# Overall summary on all 771 problems: size reduction

Table: Overall size reduction.

| method | # reduced | red. on $n$ | red. on $m$ | extra free vars |
|---|---|---|---|---|
| pd1 | 209 | 15.47% | 17.79% | 0 |
| pd2 | 230 | 15.59% | 18.23% | 0 |
| dd1 | 14 | 6.74% | 0.00% | 2,293,495 |
| dd2 | 21 | 9.28% | 0.00% | 2,315,849 |
| Sieve-SDP | 216 | 16.55% | 20.66% | 0 |

red. on $n$: $\frac{\sum n_{\text{before}} - \sum n_{\text{after}}}{\sum n_{\text{before}}}$

red. on $m$: $\frac{\sum m_{\text{before}} - \sum m_{\text{after}}}{\sum m_{\text{before}}}$

# Overall summary on all 771 problems: helpfulness

Table: Overall helpfulness.

| method | # reduced | # infeas detected | # DIMACS error improved | # out of memory |
|---|---|---|---|---|
| pd1 | 209 | 67 | 74 | 0 |
| pd2 | 230 | 67 | 78 | 6 |
| dd1 | 14 | 0 | 2 | 0 |
| dd2 | 21 | 0 | 4 | 4 |
| Sieve-SDP | 216 | 73 | 74 | 0 |

# Overall summary on all 771 problems: time

Table: Preprocessing and solving times.

| method | $t_{prep}$ (hr) | $t_{sol}$ (hr) | $t_{prep}/t_{sol\_w/o}$ | time reduction |
|---|---|---|---|---|
| w/o | - | 75.67 | - | - |
| pd1 | 0.69 | 36.77 | 0.91% | 50.50% |
| pd2 | 6.48 | 36.57 | 8.56% | 43.12% |
| dd1 | 0.16 | 75.62 | 0.22% | -0.15% |
| dd2 | 10.00 | 75.56 | 13.21% | -13.16% |
| Sieve-SDP | 0.60 | 36.62 | 0.80% | 51.81% |

$$\text{time reduction: } \frac{t_{sol\_w/o} - (t_{prep} + t_{sol})}{t_{sol\_w/o}} \times 100\%.$$

# Summary

- Sieve-SDP is fast and stable.

# Summary

- Sieve-SDP is fast and stable.
- No need for an optimization solver: Only needs Cholesky factorization.

# Summary

- Sieve-SDP is fast and stable.
- No need for an optimization solver: Only needs Cholesky factorization.

# Paper and Code

- **Paper**: `https://arxiv.org/abs/1710.08954`

# Paper and Code

- **Paper**: `https://arxiv.org/abs/1710.08954`
- To appear, Math. Programming Computation

# Paper and Code

- **Paper**: `https://arxiv.org/abs/1710.08954`
- To appear, Math. Programming Computation
- **Code**: `https://github.com/unc-optimization/SieveSDP`

# Paper and Code

- **Paper**: `https://arxiv.org/abs/1710.08954`
- To appear, Math. Programming Computation
- **Code**: `https://github.com/unc-optimization/SieveSDP`
- Try Sieve-SDP in your research, and share your experience with us, please!

# Paper and Code

- **Paper**: `https://arxiv.org/abs/1710.08954`
- To appear, Math. Programming Computation
- **Code**: `https://github.com/unc-optimization/SieveSDP`
- Try Sieve-SDP in your research, and share your experience with us, please!

Thank you for your attention!