Exploiting Low-Rank Structure in Semidefinite Programming by Approximate Operator Splitting

Mario Souto, Joaquim D. Garcia and Álvaro Veiga

June 26, 2018
Outline

Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
Semidefinite Programming

▶ Primal:

\[
\begin{align*}
\text{minimize} & \quad \text{tr}(CX) \\
\text{subject to} & \quad \text{tr}(A_i X) = b_i, \quad i = 1, \ldots, m, \\
& \quad X \succeq 0,
\end{align*}
\]

▶ Dual:

\[
\begin{align*}
\text{maximize} & \quad b^T y \\
\text{subject to} & \quad \sum_{i=1}^{m} y_i A_i \preceq C.
\end{align*}
\]

▶ Problem data:

- \(A_1, \ldots, A_p, C \in \mathbb{S}^n;\)
- \(b_1, \ldots, b_m \in \mathbb{R}.\)
Semidefinite Programming - General form

▶ Primal:

\[
\begin{align*}
\text{minimize} & \quad \text{tr}(CX) \\
\text{subject to} & \quad A(X) = b, \\
& \quad G(X) \leq h, \\
& \quad X \succeq 0.
\end{align*}
\]

▶ Dual:

\[
\begin{align*}
\text{maximize} & \quad b^T y' + h^T y'' \\
\text{subject to} & \quad A^*(y') + G^*(y'') \leq C, \\
& \quad y'' \leq 0.
\end{align*}
\]

▶ Where:

- \( A(X) = [\text{tr}(A_1 X), \ldots, \text{tr}(A_m X)]^T; \)
- \( b = [b_1, \ldots, b_m]^T. \)
Why SDP matters?

- Subsumes most of convex optimization problems;

![Diagram showing the relationship between convex optimization, Semidefinite Programming, SOCP, LP, QP, Logdet, Exponential cone, and Geometric programming.](image)
Why SDP matters?

- Subsumes most of convex optimization problems;

- Several problems of interest lie in the SDP family;
Why SDP matters?

- Subsumes most of convex optimization problems;

![Diagram showing the relationship between Semidefinite Programming, Convex optimization, SOCP, LP, QP, Logdet, Exponential cone, and Geometric programming.]

- Several problems of interest lie in the SDP family;

- Establishes tight convex relaxations for several nonconvex problems.
Applications

- Control problems;

- Robust structural design (e.g. truss topology);

- Eigenvalue optimization problems;

- Relaxations for combinatorial problems (e.g. Max-Cut, graph coloring, traveling salesman, Max-Sat, ...);

- Optimal power flow relaxation;

- Machine Learning (matrix completion, robust PCA, kernel learning).
Why isn’t SDP widely used?

- Problem size grows very fast (quadratically on matrix side);
Why isn’t SDP widely used?

- Problem size grows very fast (quadratically on matrix side);

- Sparsity is not trivial to be exploited:
  - Changing with the adoption of chordal decomposition;
Why isn’t SDP widely used?

- Problem size grows very fast (quadratically on matrix side);

- Sparsity is not trivial to be exploited:
  - Changing with the adoption of chordal decomposition;

- Formulating the problem as a SDP may not always be straightforward:
  - Solved by modern modelling frameworks (JuMP.jl, cvxpy, Convex.jl);
Why isn’t SDP widely used?

- Problem size grows very fast (quadratically on matrix side);
- Sparsity is not trivial to be exploited:
  - Changing with the adoption of chordal decomposition;
- Formulating the problem as a SDP may not always be straightforward:
  - Solved by modern modelling frameworks (JuMP.jl, cvxpy, Convex.jl);
- State-of-the-art solvers are yet unable to solve large SDP problems.
Motivation - Low-rank structure

Any SDP with $m$ constraints admits a solution with rank at most $\sqrt{2m}$ (Barvinok-Pataki 1995/98);
Motivation - Low-rank structure

- Any SDP with $m$ constraints admits a solution with rank at most $\sqrt{2m}$ (Barvinok-Pataki 1995/98);

- In practice, several SDP problems admits even lower rank solutions;

- Interior points methods frequently compute the full rank solution;

- Low-rank structure is usually exploited as a matrix factorization (Burer-Monteiro 2003): $X = V^\top V$ where $V \in \mathbb{R}^{k \times n}$ and $k$ is the target rank.
Motivation - Low-rank structure

- Any SDP with $m$ constraints admits a solution with rank at most $\sqrt{2m}$ (Barvinok-Pataki 1995/98);

- In practice, several SDP problems admits even lower rank solutions;

- Interior points methods frequently compute the full rank solution;
Any SDP with $m$ constraints admits a solution with rank at most $\sqrt{2m}$ (Barvinok-Pataki 1995/98);

In practice, several SDP problems admits even lower rank solutions;

Interior points methods frequently compute the full rank solution;

Low-rank structure is usually exploited as a matrix factorization (Burer-Monteiro 2003):

$$X = V^T V \text{ where } V \in \mathbb{R}^{k \times n} \text{ and } k \text{ is the target rank}.$$
Goals

- Propose a novel first-order method for solving SDPs:
  - Based on the *primal-dual hybrid gradient*,
  - Providing both optimal primal and dual variables,
  - Admits inequalities without the use of slack variables.

Exploit the low-rank structure within the convex optimization framework;

Make available an open source SDP solver, called ProxSDP.
Goals

- Propose a novel first-order method for solving SDPs:
  - Based on the *primal-dual hybrid gradient*,
  - Providing both optimal primal and dual variables,
  - Admits inequalities without the use of slack variables.

- Exploit the **low-rank** structure within the convex optimization framework;
Goals

- Propose a novel first-order method for solving SDPs:
  - Based on the *primal-dual hybrid gradient*,
  - Providing both optimal primal and dual variables,
  - Admits inequalities without the use of slack variables.

- Exploit the **low-rank** structure within the convex optimization framework;

- Make available an open source SDP solver, called **ProxSDP**.
Outline

Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
The SDP problem can be solved by a Proximal Point Algorithm (Rockafellar 1976):

More specifically, a particular case of the Primal-Dual Hybrid Gradient (Chambolle-Pock 2010):

The resulting algorithm PD-SDP has a convergence rate of $O(1/N)$, optimal for nonsmooth problems (Nesterov 2004).
**Algorithm**  PD–SDP

**Given:** \( M, b \in \mathbb{R}^p, h \in \mathbb{R}^q \) and \( C \in \mathbb{S}^n \).

while \( \epsilon^k_{\text{comb}} > \epsilon_{\text{tol}} \) do

\[
X^{k+1} \leftarrow \text{proj}_{\mathbb{S}^n_+}(X^k - \alpha(M^*(y^k) + C'))
\]

\[
y^{k+1/2} \leftarrow y^k + \alpha M(2X^{k+1} - X^k)
\]

\[
y^{k+1} \leftarrow y^{k+1/2} - \alpha \text{proj}_{\leq h}(y^{k+1/2}/\alpha)
\]

end while

return \((X^{k+1}, y^{k+1})\)

- Spectral decomposition + Matrix multiplication + Truncations
Outline

Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
Computational bottleneck

- The computational complexity of each iteration of PD-SDP is $\mathcal{O}(n^3)$;

Exploiting low-rank structure
The computational complexity of each iteration of PD-SDP is $O(n^3)$;

The spectral decomposition can be prohibitive even for medium scale problems;
Computational bottleneck

- The computational complexity of each iteration of PD–SDP is $O(n^3)$;

- The spectral decomposition can be prohibitive even for medium scale problems;

- Can be reduced to $O(n^2 r)$, if one knows the target rank $r$ a priori to each iteration.
Computational bottleneck

- The computational complexity of each iteration of PD-SDP is $O(n^3)$ due to spectral decomposition;

- The spectral decomposition can be prohibitive even for medium scale problems;

- Can be reduced to $O(n^2 r)$, if one knows the target rank $r$ a priori to each iteration.
Low-rank approximation

- Truncated projection onto the positive semidefinite cone:

\[ \text{proj}_{S_n^+}(X, r) = \sum_{i=1}^{r} \max\{0, \lambda_i\} u_i u_i^T. \]

- The approximation error can be expressed as the sum of the eigenvalues that were left out by the truncated projection (Eckart–Young–Mirsky theorem 1936)

\[ \left\| \text{proj}_{S_n^+}(X) - \text{proj}_{S_n^+}(X, r) \right\|_F^2 = \sum_{i=r+1}^{n} \max\{\lambda_i, 0\} \leq (n - r) \max\{\lambda_r, 0\}. \]

- The approximate fixed-point iteration does converge as long as the error component is summable (Eckstein-Bertsekas 1992)
Algorithm LR-PD-SDP

Given: $\mathcal{M}, b \in \mathbb{R}^p, h \in \mathbb{R}^q, C \in \mathbb{S}^n$ and $r = 1$.

while $(n - r) \lambda_r > \epsilon_{tol}$ do

  while $\epsilon_{comb}^k > \epsilon_{tol}$ and $\epsilon_{comb}^k < \epsilon_{comb}^{k-\ell}$ do

    $X^{k+1} \leftarrow \text{proj}_{\mathbb{S}_+^n} (X^k - \alpha (\mathcal{M}^* (y^k) + C), r)$

    $y^{k+1/2} \leftarrow y^k + \alpha \mathcal{M} (2X^{k+1} - X^k)$

    $y^{k+1} \leftarrow y^{k+1/2} - \alpha \text{proj}_{\leq h} (y^{k+1/2} / \alpha)$

  end while

  $r \leftarrow 2r$

end while

return $(X^{k+1}, y^{k+1})$
Outline

Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
ProxSDP

- Open source solver developed in the Julia language;
- Solves general SDP problems (currently can be called from JuMP.jl);
- Fast performance for problems with low-rank structure;
- Provides both optimal primal and dual solutions.
Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
Massive MIMO

Binary Multiple Input Multiple Output (MIMO):

\[ y = Hx + \varepsilon, \]

- Transmitted symbols \( x \in \{ -1, +1 \}^n \);
- Received signal \( y \in \mathbb{R}^m \);
- Matrix of channel coefficients \( H \in \mathbb{R}^{m \times n} \);
- Additive Gaussian noise \( \varepsilon \in \mathbb{R}^m \) with variance \( \sigma^2 \).
MIMO detection SDR

- Write binary constraints using the SD relaxation
- Resulting formulation:

$$\begin{align*}
\text{minimize} & \quad \text{tr}(WX) \\
\text{subject to} & \quad \text{diag}(X) = 1, \\
& \quad X \succeq 0, \\
& \quad X_{n+1,n+1} = 1, \\
& \quad -1 \leq X \leq 1, \\
& \quad \text{rank}(X) = 1.
\end{align*}$$

- De-noising is exact if signal to noise ratio is sufficiently large (Ottersten-Jalden 2006).
## MIMO experiments

**Table:** MIMO detection with high SNR.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>p</th>
<th>CSDP</th>
<th>SCS</th>
<th>LR-PD-SDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>101</td>
<td>10201</td>
<td>114.3</td>
<td>1.6</td>
<td>0.3</td>
</tr>
<tr>
<td>200</td>
<td>201</td>
<td>40401</td>
<td>timeout</td>
<td>8.9</td>
<td>1.3</td>
</tr>
<tr>
<td>300</td>
<td>301</td>
<td>90601</td>
<td>timeout</td>
<td>39.6</td>
<td>3.1</td>
</tr>
<tr>
<td>400</td>
<td>401</td>
<td>160801</td>
<td>timeout</td>
<td>101</td>
<td>6.1</td>
</tr>
<tr>
<td>500</td>
<td>501</td>
<td>251001</td>
<td>timeout</td>
<td>136.1</td>
<td>8.8</td>
</tr>
</tbody>
</table>
### Graph equipartition problem from SDPLIB

<table>
<thead>
<tr>
<th>n</th>
<th>Rank</th>
<th>Instance</th>
<th>SCS</th>
<th>CSDP</th>
<th>PD-SDP</th>
<th>LR-PD-SDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>4</td>
<td>gpp124-1</td>
<td>29.8</td>
<td>0.6</td>
<td>8.4</td>
<td>0.6</td>
</tr>
<tr>
<td>124</td>
<td>4</td>
<td>gpp124-2</td>
<td>11.1</td>
<td>0.5</td>
<td>6.6</td>
<td>0.5</td>
</tr>
<tr>
<td>124</td>
<td>6</td>
<td>gpp124-3</td>
<td>9.3</td>
<td>0.6</td>
<td>5.3</td>
<td>0.5</td>
</tr>
<tr>
<td>124</td>
<td>6</td>
<td>gpp124-4</td>
<td>13.5</td>
<td>0.6</td>
<td>18.4</td>
<td>0.8</td>
</tr>
<tr>
<td>250</td>
<td>5</td>
<td>gpp250-1</td>
<td>155.6</td>
<td>2.4</td>
<td>32.2</td>
<td>2.6</td>
</tr>
<tr>
<td>250</td>
<td>7</td>
<td>gpp250-2</td>
<td>76.9</td>
<td>2.4</td>
<td>23.7</td>
<td>2.7</td>
</tr>
<tr>
<td>250</td>
<td>8</td>
<td>gpp250-3</td>
<td>61.8</td>
<td>2.2</td>
<td>29.3</td>
<td>3.5</td>
</tr>
<tr>
<td>250</td>
<td>8</td>
<td>gpp250-4</td>
<td>70.5</td>
<td>2.3</td>
<td>40.3</td>
<td>2.5</td>
</tr>
<tr>
<td>500</td>
<td>7</td>
<td>gpp500-1</td>
<td>timeout</td>
<td>25.9</td>
<td>150.2</td>
<td>9.7</td>
</tr>
<tr>
<td>500</td>
<td>8</td>
<td>gpp500-2</td>
<td>634.2</td>
<td>22.3</td>
<td>156.8</td>
<td>9.5</td>
</tr>
<tr>
<td>500</td>
<td>11</td>
<td>gpp500-3</td>
<td>405.37</td>
<td>16.4</td>
<td>117.2</td>
<td>11.9</td>
</tr>
<tr>
<td>500</td>
<td>13</td>
<td>gpp500-4</td>
<td>429.7</td>
<td>13.4</td>
<td>129.4</td>
<td>14.4</td>
</tr>
<tr>
<td>801</td>
<td>31</td>
<td>equalG11</td>
<td>timeout</td>
<td>81.1</td>
<td>timeout</td>
<td>29.4</td>
</tr>
<tr>
<td>1001</td>
<td>16</td>
<td>equalG51</td>
<td>timeout</td>
<td>164.6</td>
<td>timeout</td>
<td>55.6</td>
</tr>
</tbody>
</table>

**Table:** Comparison of convergence time, in seconds, for SDPLIB’s graph equipartition problem instances.
Introduction

Algorithm

Exploiting low-rank structure

SDP solver

Case studies

Conclusion
Achievements:

- Primal-dual method for solving SDP;

Future ideas:

- Explore properties of low-rank recovered solution;
- Combine proposed method with chordal sparsity techniques;
- Represent other cones;
Conclusion

Achievements:

- Primal-dual method for solving SDP;
- Low-rank structure is efficiently exploited;

Open-source SDP solver [ProxSDP](https://github.com/mariohsouto/ProxSDP.jl) is readily available.

Future ideas:

- Explore properties of low-rank recovered solutions;
- Combine proposed method with chordal sparsity techniques;
- Represent other cones;
Conclusion

Achievements:

- Primal-dual method for solving SDP;
- Low-rank structure is efficiently exploited;
- Open-source SDP solver [ProxSDP] is readily available, https://github.com/mariohsouto/ProxSDP.jl
Conclusion

▶ Achievements:
  
  o Primal-dual method for solving SDP;
  
  o Low-rank structure is efficiently exploited;
  
  o Open-source SDP solver [ProxSDP] is readily available, https://github.com/mariohsouto/ProxSDP.jl

▶ Future ideas:
  
  o Explore properties of low-rank recovered solution;
Conclusion

▶ Achievements:

- Primal-dual method for solving SDP;
- Low-rank structure is efficiently exploited;
- Open-source SDP solver [ProxSDP] is readily available, https://github.com/mariohsouto/ProxSDP.jl

▶ Future ideas:

- Explore properties of low-rank recovered solution;
- Combine proposed method with chordal sparsity techniques;
Conclusion

▶ Achievements:

- Primal-dual method for solving SDP;
- Low-rank structure is efficiently exploited;
- Open-source SDP solver [ProxSDP] is readily available, https://github.com/mariohsouto/ProxSDP.jl

▶ Future ideas:

- Explore properties of low-rank recovered solution;
- Combine proposed method with chordal sparsity techniques;
- Represent other cones;