Solving SDPs by using sparse PCA

Santanu S. Dey\textsuperscript{1}  Aleksandr M. Kazachkov\textsuperscript{2}  Andrea Lodi\textsuperscript{3}  Gonzalo Muñoz\textsuperscript{4}

\textsuperscript{1}Georgia Institute of Technology.  \textsuperscript{2}University of Florida.  \textsuperscript{3}École Polytechnique de Montréal.  \textsuperscript{4}Universidad de O’Higgins.

Thanks to A. M. Kazachkov for all the figures in the slides.

Feb 2021
Solving SDPs by using sparse PCA


Santanu S. Dey\textsuperscript{1}  Aleksandr M. Kazachkov\textsuperscript{2}  Andrea Lodi\textsuperscript{3}  Gonzalo Muñoz\textsuperscript{4}

\textsuperscript{1}Georgia Institute of Technology.
\textsuperscript{2}University of Florida.  \textsuperscript{3}École Polytechnique de Montréal.
\textsuperscript{4}Universidad de O’Higgins.

Thanks to A. M. Kazachkov for all the figures in the slides.

Feb 2021
1
The challenge – creating lightweight outer approximation of SDPs
Semi-definite programming

\[
\begin{align*}
\text{min} & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S_+^n, \\
\end{align*}
\]

where \( C \) and the \( A^i \)'s are \( n \times n \) matrices, \( \langle M, N \rangle := \sum_{i,j} M_{ij} N_{ij} \), and

Polynomial-time algorithm – but often challenging to solve in practice.

In many applications (Combinatorial optimization, non-convex quadratic problems), the SDP is a relaxation of a more challenging nonconvex problem, which needs to be solved using the (spacial) branch-and-bound algorithm. We do not want to maintain a SDP relaxation – since it would have to solved at each node — but maintain a linear programming relaxation.
Semi-definite programming

\[
\begin{align*}
\text{min} & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+, \\
\end{align*}
\]  
(SDP)

where \( C \) and the \( A^i \)'s are \( n \times n \) matrices, \( \langle M, N \rangle := \sum_{i,j} M_{ij} N_{ij} \), and

\[
S^n_+ = \{ X \in \mathbb{R}^{n \times n} | X = X^T, \ u^\top X u \geq 0, \ \forall u \in \mathbb{R}^n \}.
\]
Semi-definite programming

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+,
\end{align*}
\]

(SDP)

where \( C \) and the \( A^i \)'s are \( n \times n \) matrices, \( \langle M, N \rangle := \sum_{i,j} M_{ij} N_{ij} \), and

\[
S^n_+ = \{ X \in \mathbb{R}^{n \times n} \mid X = X^T, \ u^\top X u \geq 0, \ \forall u \in \mathbb{R}^n \}.
\]

- Polynomial-time algorithm – but often challenging to solve in practice.
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse principle component analysis

Experiments

Semi-definite programming

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i, \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+,
\end{align*}
\]

where \( C \) and the \( A^i \)'s are \( n \times n \) matrices, \( \langle M, N \rangle := \sum_{i,j} M_{ij}N_{ij} \), and \( S^n_+ = \{ X \in \mathbb{R}^{n \times n} \mid X = X^T, \ u^\top Xu \geq 0, \ \forall u \in \mathbb{R}^n \} \).

- **Polynomial-time algorithm** – but often challenging to solve in practice.
- In many applications (Combinatorial optimization, non-convex quadratic problems), the SDP is a relaxation of a more challenging nonconvex problem, which needs to be solved using the (spacial) branch-and-bound algorithm. **We do not want to maintain a SDP relaxation** – since it would have to solved at each node — but maintain a linear programming relaxation.
Semi-definite programming: solving via linear cutting-planes

1. Construct a linear programming relaxation:

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+ 
\end{align*}
\]  

(LP-Relax-of-SDP)
Solving SDPs by using sparse PCA

1. Construct a linear programming relaxation:

\[
\begin{align*}
    \min & \quad \langle C, X \rangle \\
    \text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
    & \quad X \in S^n_+ \\
\end{align*}
\]  

(LP-Relax-of-SDP)

2. Let optimal solution be \( \tilde{X} \). If \( \tilde{X} \in S^n_+ \), then we have solved the SDP.
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse priniciple component analysis

Experiements

Semi-definite programming: solving via linear cutting-planes

1. Construct a linear programming relaxation:

   \[
   \begin{align*}
   &\text{min} & & \langle C, X \rangle \\
   &\text{s.t.} & & \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
   & & & X \in S_+^n
   \end{align*}
   \]  
   (LP-Relax-of-SDP)

2. Let optimal solution be $\tilde{X}$. If $\tilde{X} \in S_+^n$, then we have solved the SDP.

3. Else,

   \[
   \tilde{X} = \sum_{i=1}^{n} \lambda_i v^i (v^i)^T
   \]
   and, say $\lambda_1 < 0$.
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse priniciple component analysis

Experiements

Semi-definite programming: solving via linear cutting-planes

1. Construct a linear programming relaxation:

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+ \\
\end{align*}
\] (LP-Relax-of-SDP)

2. Let optimal solution be $\tilde{X}$. If $\tilde{X} \in S^n_+$, then we have solved the SDP.

3. Else,

\[
\tilde{X} = \sum_{i=1}^{n} \lambda_i v^i (v^i)^T \quad \text{and, say } \lambda_1 < 0,
\]

4. Let $V = v^1 (v^1)^T$. Then

$$\langle V, \tilde{X} \rangle = \lambda_1 < 0, \quad \text{but } \langle V, X \rangle \geq 0 \quad \text{for all } X \in S^n_+$$

Go to Step 2.
Semi-definite programming: solving via linear cutting-planes

1. Construct a linear programming relaxation:

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad X \in S^n_+ \quad \text{(LP-Relax-of-SDP)}
\end{align*}
\]

2. Let optimal solution be \( \tilde{X} \). If \( \tilde{X} \in S^n_+ \), then we have solved the SDP.

3. Else,

\[
\tilde{X} = \sum_{i=1}^{n} \lambda_i v^i(v^i)^T \quad \text{and, say } \lambda_1 < 0,
\]

4. Let \( V = v^1(v^1)^T \). Then

\[
\langle V, \tilde{X} \rangle = \lambda_1 < 0, \quad \text{but } \langle V, X \rangle \geq 0 \quad \text{for all } X \in S^n_+
\]

5. Update, (LP-Relax-of -SDP) as:

\[
\begin{align*}
\min & \quad \langle C, X \rangle \\
\text{s.t.} & \quad \langle A^i, X \rangle \leq b_i \quad \forall i \in \{1, \ldots, m\} \\
& \quad \langle V, X \rangle \geq 0.
\end{align*}
\]

Go to Step 2.
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse principal component analysis

Experiements

Lets see an example of this at work...

▶ Box QP:

\[
\begin{align*}
\text{min} & \quad x^T Q x + c^T x \\
\text{s.t.} & \quad x \in [0, 1]^n
\end{align*}
\]
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse principal component analysis

Experiements

Lets see an example of this at work...

- **Box QP:**

  \[
  \begin{align*}
  \text{min} & \quad x^\top Q x + c^\top x \\
  \text{s.t.} & \quad x \in [0, 1]^n
  \end{align*}
  \]

- **SDP (+ McCormick) relaxation:**

  \[
  \begin{align*}
  \text{min} & \quad \langle Q, xx^T \rangle + c^\top x \\
  \text{s.t.} & \quad x \in [0, 1]^n
  \end{align*}
  \]
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse priniciple component analysis

Experiements

Lets see an example of this at work...

- **Box QP:**

\[
\begin{align*}
\text{min} & \quad x^T Q x + c^T x \\
\text{s.t.} & \quad x \in [0, 1]^n
\end{align*}
\]

- **SDP (+ McCormick) relaxation:**

\[
\begin{align*}
\text{min} & \quad \langle Q, xx^T \rangle + c^T x \\
\text{s.t.} & \quad x \in [0, 1]^n
\end{align*}
\]

\[
\begin{align*}
\text{min} & \quad \langle Q, X \rangle + c^T x \\
\text{s.t.} & \quad x \in [0, 1]^n, X = xx^T
\end{align*}
\]
Let's see an example of this at work...

**Box QP:**

\[
\begin{align*}
\text{min} & \quad x^\top Q x + c^\top x \\
\text{s.t.} & \quad x \in [0, 1]^n
\end{align*}
\]

**SDP (+ McCormick) relaxation:**

\[
\begin{align*}
\text{min} & \quad \langle Q, xx^T \rangle + c^\top x \\
\text{s.t.} & \quad x \in [0, 1]^n, X = xx^T
\end{align*}
\]

\[
\begin{align*}
\text{min} & \quad \langle Q, X \rangle + c^\top x \\
\text{s.t.} & \quad \max\{0, x_i + x_j - 1\} \leq X_{ij} \leq \min\{x_i, x_j\}, i, j \in [n] \quad \left[ \begin{array}{cc} 1 & x^\top \\ x & X \end{array} \right] \in S^n_+.
\end{align*}
\]

**spar125-025-1** (https://github.com/sburer/BoxQPinstances): 
\(n = 125\) and \(Q\) has 25\% non-zeros.
Gap closed vs. time

Eigenvector cut performance (spar125-025-1)

"Tailing off" at 75% gap closed
Solving SDPs by using sparse PCA

Dey, Kazachkov, Lodi, Muñoz

Creating lightweight outer approximation of SDPs

Sparse principal component analysis

Experiments

Gap closed vs. time

Eigenvector cut performance (spar125-025-1)

“Tailing off” at 75% gap closed

Time to solve the LP is about 94% of the time taken for each iteration

What if we can drastically reduce solving time?
Gap closed vs. time

Eigenvector cut performance (spar125-025-1)

“Tailing off” at 75% gap closed

Time to solve the LP is about 94% of the time taken for each iteration

What if we can drastically reduce solving time?
Why are the LPs becoming so difficult to solve?

- \( n = 125 \). So LP has 7875 variables.
Why are the LPs becoming so difficult to solve?

- $n = 125$. So LP has 7875 variables.

\[
\tilde{X} = \sum_{i=1}^{n} \lambda_i v_i (v_i)^T
\]

and, say $\lambda_1 < 0$.

Typically, the eigenvector ($v^1$) corresponding to the negative eigenvalue is dense.
Why are the LPs becoming so difficult to solve?

- $n = 125$. So LP has 7875 variables.

\[
\tilde{X} = \sum_{i=1}^{n} \lambda_i v_i (v_i)^T
\]

and, say $\lambda_1 < 0$,

Typically, the eigenvector $(v^1)$ corresponding to the negative eigenvalue is dense.

- Therefore, the inequality

\[
\langle V, X \rangle \geq 0,
\]

is a completely dense inequality in an LP which is already “largish”.

As these dense inequalities keep getting added, the LP solve times increase.

On the other hand, LP solvers love sparsity (Many linear algebra routine can exploit sparsity)!
Why are the LPs becoming so difficult to solve?

- $n = 125$. So LP has 7875 variables.

- $\tilde{X} = \sum_{i=1}^{n} \lambda_i v_i (v_i)^T$ and, say $\lambda_1 < 0$.

Typically, the eigenvector ($v^1$) corresponding to the negative eigenvalue is dense.

- Therefore, the inequality $\langle V, X \rangle \geq 0$,

is a completely dense inequality in an LP which is already “largish”.

- As these dense inequalities keep getting added, the LP solve times increase.
Why are the LPs becoming so difficult to solve?

- $n = 125$. So LP has 7875 variables.

- $\tilde{X} = \sum_{i=1}^{n} \lambda_{i} v^{i}(v^{i})^{T}$ and, say $\lambda_{1} < 0$,

Typically, the eigenvector ($v^{1}$) corresponding to the negative eigenvalue is dense.

- Therefore, the inequality

$$\langle V, X \rangle \geq 0,$$

is a completely dense inequality in an LP which is already “largish”.

- As these dense inequalities keep getting added, the LP solve times increase.

- On the other hand, LP solvers love sparsity (Many linear algebra routine can exploit sparsity)!
It is no enough to just explore linear programming relaxations of SDPs.
Conclusion

- It is no enough to just explore linear programming relaxations of SDPs.
- What we really want is \emph{sparse linear programming relaxations to SDPs}. 
Conclusion

- It is no enough to just explore linear programming relaxations of SDPs.
- What we really want is **sparse linear programming relaxations to SDPs**.
- Let's decide a sparsity level $k \ll n$: If $\tilde{X} \not\in S_+^n$, then find a vector $v$ such that

$$v^T \tilde{X} v < 0, \|v\|_0 \leq k,$$

then $V = vv^T$ will be sparse.

Questions:
1. How to find such “sparse eigenvectors”?
2. Will such sparse linear approximation of SDPs work?

It is no enough to just explore linear programming relaxations of SDPs.

What we really want is **sparse linear programming relaxations to SDPs**.

Lets decide a sparsity level $k << n$: If $\tilde{X} \not\in S_n^+$, then find a vector $v$ such that

$$v^\top \tilde{X} v < 0, \|v\|_0 \leq k,$$

then $V = vv^\top$ will be sparse.

Questions:

1. How to find such “sparse eigenvectors”?
Conclusion

- It is no enough to just explore linear programming relaxations of SDPs.
- What we really want is \textit{sparse linear programming relaxations to SDPs}.
- Let's decide a sparsity level $k << n$: If $\tilde{X} \not\in S^n_+$, then find a vector $v$ such that
  \[ v^T \tilde{X} v < 0, \|v\|_0 \leq k, \]
  then $V = vv^T$ will be sparse.
- Questions:
  1. How to find such "sparse eigenvectors"?
  2. Will such sparse linear approximation of SDPs work?
Conclusion

- It is no enough to just explore linear programming relaxations of SDPs.
- What we really want is **sparse linear programming relaxations to SDPs**.
- Let's decide a sparsity level $k << n$: If $\tilde{X} \not\in S^n_+$, then find a vector $v$ such that

$$v^T \tilde{X} v < 0, \|v\|_0 \leq k,$$

then $V = vv^T$ will be sparse.
- Questions:
  1. How to find such “sparse eigenvectors”?
  2. Will such sparse linear approximation of SDPs work?

2
Our key technique - Sparse PCA: a well studied problem in statistics/ML literature
Principal component analysis (PCA) and sparse PCA

Let $\bar{X} \in S^n_+$ (covariance matrix), the PCA problem:

$$w^* \in \arg\max_w \quad w^\top \bar{X} w$$

s.t.  \hspace{1cm} \|w\|_2 \leq 1 \quad \|w\|_0 \leq k$$

\{ PCA \}

Let $\bar{X} \in S^n_+$ (covariance matrix), the PCA problem:
Principal component analysis (PCA) and sparse PCA

Let $\bar{X} \in S^n_+$ (covariance matrix), the PCA problem:

$$w^* \in \underset{w}{\text{argmax}} \quad w^\top \bar{X} w \quad \text{s.t.} \quad \|w\|_2 \leq 1$$

- The optimal objective function of the above problem is the largest eigen value of $\bar{X}$.
- Given $\bar{X}$ is covariance matrix – the optimal solution of the above problem can be interpreted as the direction that captures the maximum variance.
- If $w^*$ is dense, this is no ideal from interpretability.
Principal component analysis (PCA) and sparse PCA

Let $\bar{X} \in S_+^n$ (covariance matrix), the PCA problem:

$$w^* \in \arg\max_w \quad w^\top \bar{X} w \quad \text{s.t.} \quad \|w\|_2 \leq 1$$

\[ \{ \text{PCA} \} \]

- The optimal objective function of the above problem is the largest eigenvalue of $\bar{X}$.
- Given $\bar{X}$ is covariance matrix – the optimal solution of the above problem can be interpreted as the direction that captures the maximum variance.
- If $w^*$ is dense, this is no ideal from interpretability.

$$\tilde{w} := \arg\max_w \quad w^\top \bar{X} w \quad \text{s.t.} \quad \|w\|_2 \leq 1 \quad \|w\|_0 \leq k$$

\[ \{ \text{Sparse PCA} \} \]
### How does SPCA help?

<table>
<thead>
<tr>
<th>Goal (given $\tilde{X}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find $v$ such that $v^T \tilde{X} v &lt; 0$, $|v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPCA (assuming $\bar{X} \in S^n_+$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve: $\max v^T \tilde{X} v : |v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>
How does SPCA help?

<table>
<thead>
<tr>
<th>Goal (given $\tilde{X}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find $v$ : such that $v^T \tilde{X} v &lt; 0$, $|v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPCA (assuming $\bar{X} \in S^n_+$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve : $\max v^T \bar{X} v : |v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>

$\tilde{X} \rightarrow$
### How does SPCA help?

<table>
<thead>
<tr>
<th>Goal (given $\tilde{X}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find $v$ : such that $v^T \tilde{X}v &lt; 0$, $|v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPCA (assuming $\tilde{X} \in S^n_+$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve : $\max v^T \tilde{X}v : |v|_2 \leq 1$, $|v|_0 \leq k$</td>
</tr>
</tbody>
</table>

$\tilde{X} \rightarrow -\tilde{X}$ (Not necessarily PSD)
How does SPCA help?

Goal (given $\tilde{X}$):

Find $v$ : such that $v^\top \tilde{X} v < 0, \|v\|_2 \leq 1, \|v\|_0 \leq k$

SPCA (assuming $\bar{X} \in S^n_+$) :

Solve : $\max v^\top \bar{X} v : \|v\|_2 \leq 1, \|v\|_0 \leq k$

$\tilde{X} \rightarrow -\tilde{X}$ ( Not necessarily PSD ) $\rightarrow -\tilde{X} + \lambda_{\text{max}}(\tilde{X}) I$
How does SPCA help?

<table>
<thead>
<tr>
<th>Goal (given $\tilde{X}$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find $\mathbf{v}$ such that $\mathbf{v}^\top \tilde{X} \mathbf{v} &lt; 0$, $|\mathbf{v}|_2 \leq 1$, $|\mathbf{v}|_0 \leq k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPCA (assuming $\bar{X} \in S_+^n$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve: $\max \mathbf{v}^\top \bar{X} \mathbf{v}$: $|\mathbf{v}|_2 \leq 1$, $|\mathbf{v}|_0 \leq k$</td>
</tr>
</tbody>
</table>

$\tilde{X} \rightarrow -\tilde{X}$ (Not necessarily PSD) $\rightarrow -\tilde{X} + \lambda_{\text{max}}(\tilde{X}) I$

Proposition (Sparse separation via SPCA)

- **If**
  $$\max \left\{ \mathbf{v}^\top \left( -\bar{X} + \lambda_{\text{max}}(\bar{X}) I \right) \mathbf{v} : \|\mathbf{v}\|_2 \leq 1, \|\mathbf{v}\|_0 \leq k \right\} \leq \lambda_{\text{max}}(\bar{X})$$
  then there is no sparse inequality.

- **If**
  $$\max \left\{ \mathbf{v}^\top \left( -\bar{X} + \lambda_{\text{max}}(\bar{X}) I \right) \mathbf{v} : \|\mathbf{v}\|_2 \leq 1, \|\mathbf{v}\|_0 \leq k \right\} > \lambda_{\text{max}}(\bar{X})$$
  then if $\mathbf{v}^*$ is the optimal solution then $\langle \mathbf{v}^*(\mathbf{v}^*)^\top, X \rangle \geq 0$ is a sparse separating hyperplane.
Sparse PCA is “mathematically hopeless” to solve — what about “in practice”?

- Approximation algorithm with multiplicative guarantee: [S. On Chan, D. Papailliooulos, A. Rubinstein, 2016], [M. Magdon-Ismail, (2017)] NP-hardness of approximation to within \((1 - \varepsilon)\), for some small constant \(\varepsilon > 0\).
Sparse PCA is “mathematically hopeless” to solve — what about “in practice”?

- Approximation algorithm with multiplicative guarantee: [S. On Chan, D. Papailiopoulos, A. Rubinstein, 2016], [M. Magdon-Ismail, (2017)] NP-hardness of approximation to within $(1 - \varepsilon)$, for some small constant $\varepsilon > 0$. The best approximation guarantee of $n^{1/3}$.
Sparse PCA is "mathematically hopeless" to solve — what about "in practice"?

- Approximation algorithm with multiplicative guarantee: [S. On Chan, D. Papailiopoulos, A. Rubinstein, 2016], [M. Magdon-Ismail, (2017)] NP-hardness of approximation to within $(1 - \varepsilon)$, for some small constant $\varepsilon > 0$. The best approximation guarantee of $n^{1/3}$. 

Sparse PCA is “mathematically hopeless" to solve — what about “in practice"?

- Approximation algorithm with multiplicative guarantee: [S. On Chan, D. Papailliopoulos, A. Rubinstein, 2016], [M. Magdon-Ismail, (2017)] NP-hardness of approximation to within $(1 - \varepsilon)$, for some small constant $\varepsilon > 0$. The best approximation guarantee of $n^{1/3}$.


- Exact/near exact mathematical programming methods: [SSD., Rahul Mazumder, Wang (2018)], [Yongchun Li, Weijun Xie (2020)]. Can scale up to matrices of size $1000 \times 1000$ in a few hours.
Sparse PCA is “mathematically hopeless” to solve — what about “in practice”?

- Approximation algorithm with multiplicative guarantee: [S. On Chan, D. Papailiopoulos, A. Rubinstein, 2016], [M. Magdon-Ismail, (2017)] NP-hardness of approximation to within \((1 - \varepsilon)\), for some small constant \(\varepsilon > 0\). The best approximation guarantee of \(n^{1/3}\).


- Exact/near exact mathematical programming methods: [SSD., Rahul Mazumder, Wang (2018)], [Yongchun Li, Weijun Xie (2020)]. Can scale up to matrices of size \(1000 \times 1000\) in a few hours.

ML community has come up with some fantastic heuristics:
- [X.-T. Yuan and T. Zhang (2013)] “Truncated power method”
- [M. Journee, Y. Nesterov, P. Richtarik, and R. Sepulchre, (2010)]
3
Experiments
Back to experimental results with $k = 0.25n$, One cut per iteration
Back to experimental results with $k = 0.25n$, One cut per iteration

Sparse cuts may eventually surpass the performance of dense cuts, avoiding tailing off, but when?

For smaller instances, this crossing point can be reached within an hour

For larger instances such as this, dense cuts have too much of a head start
Need to add multiple cuts in each iteration

After trying out various ideas, we settled on the following scheme:

- We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follows:
  - We add multiple cuts in each iteration generated as follow
After trying out various ideas, we settled on the following scheme:

- We add multiple cuts in each iteration generated as follows:
  - $\tilde{X}$, we find $v^0$ such that $(v^0)^\top \tilde{X} v^0 = \lambda^0 < 0$.

- Update $\tilde{X}_1 := \tilde{X} - \lambda^0 (v^0)^\top v^0$, and we find $v^1$ such that $(v^1)^\top \tilde{X}_1 v^1 = \lambda^1 < 0$.

- Repeat...
Need to add multiple cuts in each iteration

After trying out various ideas, we settled on the following scheme:

- We add **multiple cuts** in each iteration generated as follows:
  - \( \tilde{X} \), we find \( v^0 \) such that \( (v^0)^\top \tilde{X} v^0 = \lambda^0 < 0 \).
  - Update \( \tilde{X}^1 := \tilde{X} - \lambda^0 (v^0)^\top v^0 \), and we find \( v^1 \) such that \( (v^1)^\top \tilde{X}^1 v^1 = \lambda^1 < 0 \).
  - Repeat...

\( v^0, v^1, \ldots \) are used to generate cuts.
Need to add multiple cuts in each iteration

After trying out various ideas, we settled on the following scheme:

- We add **multiple cuts** in each iteration generated as follows:
  - \( \tilde{X} \), we find \( v^0 \) such that \( (v^0)^\top \tilde{X} v^0 = \lambda^0 < 0 \).
  - Update \( \tilde{X}^1 := \tilde{X} - \lambda^0 (v^0)^\top v^0 \), and we find \( v^1 \) such that \( (v^1)^\top \tilde{X}^1 v^1 = \lambda^1 < 0 \).
  - Repeat...
Need to add multiple cuts in each iteration

After trying out various ideas, we settled on the following scheme:

- We add **multiple cuts** in each iteration generated as follows:
  - \( \tilde{X} \), we find \( v^0 \) such that \( (v^0)\top \tilde{X} v^0 = \lambda^0 < 0 \).
  - Update
    \[
    \tilde{X}^1 := \tilde{X} - \lambda^0 (v^0)\top v^0,
    \]
    and we find \( v^1 \) such that \( (v^1)\top \tilde{X}^1 v^1 = \lambda^1 < 0 \).
  - Repeat...
  - \( v^0, v^1, \ldots \) are used to generate cuts.
More experiments (multiple cuts per iteration):

Percent gap closed with respect to QP optimum (spar125-025-1)
More experiments (multiple cuts per iteration):

Time to solve LP relaxation per iteration (spar125-025-1)

The LP is slowed by the large number of cuts being added, but it is still significantly faster than using dense cuts.
Strength of sparse cuts

Theorem (G. Blekherman, SSD., M. Molinaro, K. Shu, S. Sun (2020))

\[
\begin{align*}
& \max \quad \text{dist}_F(X, S_n^+) \\
& \text{s.t.} \quad \|X\|_F \leq 1 \\
& \quad \text{every } k \times k \\
& \quad \text{principal submatrix} \\
& \quad \text{of } X \text{ is PSD.}
\end{align*}
\]

\[
\leq \max \left\{ \frac{1 - k}{n}, \frac{(n - k)^{3/2}}{\sqrt{(n - k)^2 + (n - 1)k^2}} \right\}
\]
Strength of sparse cuts

Theorem (G. Blekherman, SSD., M. Molinaro, K. Shu, S. Sun (2020))

\[
\begin{align*}
\max & \quad \text{dist}_F(X, S^n_+), \\
\text{s.t.} & \quad \|X\|_F \leq 1, \\
& \quad \text{every } k \times k \\
& \quad \text{principal submatrix} \\
& \quad \text{of } X \text{ is PSD.}
\end{align*}
\]

\[
\leq \max \left\{ \frac{1-k}{n}, \frac{(n-k)^{3/2}}{\sqrt{(n-k)^2 + (n-1)k^2}} \right\}
\]

▶ Lets add some dense cuts in the first iteration.
More experiments (multiple cuts per iteration, dense cuts in first iteration):

Percent gap closed with respect to QP optimum (spar125-025-1)
More experiments (multiple cuts per iteration, dense cuts in first iteration):

Time to solve LP relaxation per iteration (spar125-025-1)

Max sparse cuts per iteration $K = 5n$

Sparsity level $k = 0.25(n+1)$
More experiments (multiple cuts per iteration, dense cuts in first iteration):

Percent gap closed with respect to SDP + McCormicks optimum (g05_100.0) QCQP instance!

Max sparse cuts per iteration $K = 5n$

Sparsity level $k = 0.25(n + 1)$
More experiments (multiple cuts per iteration, dense cuts in first iteration):

Percent gap closed with respect to SDP + McCormicks optimum (t2g10_5555)

QCQP instance!
Results

Results on 111 BoxQP instances for SPARSE, DENSE, and HYBRID. Results are averages over instances grouped by size, under a time limit of 1 hour.

<table>
<thead>
<tr>
<th>Instance group</th>
<th></th>
<th>SPARSE</th>
<th>DENSE</th>
<th>HYBRID</th>
<th>SPARSE</th>
<th>DENSE</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \in [20, 30]$</td>
<td>18</td>
<td>98.50</td>
<td>100.00</td>
<td>100.00</td>
<td>0.10</td>
<td>2.93</td>
<td>2.93</td>
</tr>
<tr>
<td>$n \in [40, 50]$</td>
<td>33</td>
<td>98.83</td>
<td>99.90</td>
<td>99.89</td>
<td>0.64</td>
<td>10.73</td>
<td>7.34</td>
</tr>
<tr>
<td>$n \in [60, 80]$</td>
<td>21</td>
<td>98.45</td>
<td>96.24</td>
<td>98.17</td>
<td>6.49</td>
<td>28.27</td>
<td>11.69</td>
</tr>
<tr>
<td>$n \in [90, 125]$</td>
<td>27</td>
<td>94.62</td>
<td>90.68</td>
<td>95.48</td>
<td>48.09</td>
<td>106.54</td>
<td>49.08</td>
</tr>
<tr>
<td>$n \in [200, 250]$</td>
<td>12</td>
<td>75.16</td>
<td>84.70</td>
<td>83.92</td>
<td>520.24</td>
<td>764.30</td>
<td>506.98</td>
</tr>
</tbody>
</table>

Commercial SDP solver (Mosek) for $n \in [200, 250]$ needs approximately 35 GB memory, needs > 1 hour to solve.
## Results

Results on 135 BiQ instances for SPARSE, DENSE, and HYBRID. Results are averages over instances grouped by size, under a time limit of 1 hour.

<table>
<thead>
<tr>
<th>Instance group</th>
<th>#</th>
<th>SPARSE</th>
<th>DENSE</th>
<th>HYBRID</th>
<th>SPARSE</th>
<th>DENSE</th>
<th>HYBRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \in [20, 90]$</td>
<td>18</td>
<td>98.70</td>
<td>99.47</td>
<td>99.81</td>
<td>1.33</td>
<td>15.14</td>
<td>7.26</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>31</td>
<td>94.92</td>
<td>82.53</td>
<td>95.00</td>
<td>31.33</td>
<td>91.35</td>
<td>34.82</td>
</tr>
<tr>
<td>$n \in [120, 150]$</td>
<td>41</td>
<td>90.18</td>
<td>89.35</td>
<td>92.61</td>
<td>125.87</td>
<td>262.56</td>
<td>132.43</td>
</tr>
<tr>
<td>$n \in [200, 250]$</td>
<td>45</td>
<td>54.72</td>
<td>65.72</td>
<td>64.06</td>
<td>479.61</td>
<td>830.75</td>
<td>519.96</td>
</tr>
</tbody>
</table>
Results on 151 MaxCut instances for SPARSE, DENSE, and HYBRID. Results are averages over instances grouped by size, under a time limit of 1 hour.

<table>
<thead>
<tr>
<th>Instance group</th>
<th>Gap closed (%)</th>
<th>Last LP time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SPARSE</td>
</tr>
<tr>
<td>$n = 60$</td>
<td>10</td>
<td>97.45</td>
</tr>
<tr>
<td>$n = 80$</td>
<td>30</td>
<td>93.61</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>99</td>
<td>79.36</td>
</tr>
<tr>
<td>$n \in [150,225]$</td>
<td>12</td>
<td>6.00</td>
</tr>
</tbody>
</table>
Thank You.

