# Semidefinite Programming by `Projective Cutting Planes`

Daniel Porumbel[1]

Conservatoire National des Arts et Métiers, Paris, France
daniel.porumbel@cnam.fr
http://cedric.cnam.fr/~porumbed/

**Abstract.** Seeking tighter relaxations of combinatorial optimization problems, semidefinite programming is a generalization of linear programming that offers better bounds and is still polynomially solvable. Yet, in practice, a semidefinite program is still significantly harder to solve than a similar-size Linear Program (LP). It is well-known that a semidefinite program can be written as an LP with infinitely-many cuts that could be solved by repeated separation in a `Cutting-Planes` scheme; this approach is likely to end up in failure. We proposed in [5] the `Projective Cutting-Planes` that upgrade the well-known separation sub-problem to the projection sub-problem: given a feasible $\mathbf{y}$ inside a polytope $\mathscr{P}$ and a direction $\mathbf{d}$, find the maximum $t^*$ so that $\mathbf{y} + t^*\mathbf{d} \in \mathscr{P}$. Using this new sub-problem, one can generate a sequence of both inner and outer solutions that converge to the optimum over $\mathscr{P}$. This paper shows that the projection sub-problem can be solved very efficiently in a semidefinite programming context, enabling the resulting `Projective Cutting-Planes` to compete very well with state-of-the-art semidefinite optimization software (refined over decades). Results suggest it may the fastest method for matrix sizes larger than $2000 \times 2000$.

**Keywords:** Semidefinite programming · Separation and projection sub-problem · Projective Cutting Planes

## 1 Introduction

We consider the following semidefinite optimization problem

$$(SDP)\begin{cases}\max\limits_{\mathbf{y}\in\mathbb{R}^n} & \mathbf{b}^\top\mathbf{y} & \text{(1a)}\\ s.t & \mathcal{A}^\top\mathbf{y} \preceq C & \text{(1b)}\\ & \mathbf{a}^\top\mathbf{y} \le c_a \ \forall(\mathbf{a}, c_a)\in\mathscr{C} & \text{(1c)}\end{cases}$$

where $\mathcal{A}^\top\mathbf{y} = \sum_{i=1}^k A_i y_i$; $A_1,\ A_2, \dots, A_k$ and $C$ are symmetric $n \times n$ matrices. The set $\mathscr{C}$ in (1c) contains simple linear constraints that may include $\mathbf{y} \ge \mathbf{0}$ (*i.e.*, one can enforce $y_i \ge 0$ by taking $a_i = -1$ and $a_j = 0 \forall j \ne i$ and $c_a = 0$). Adding a limited number of other linear constraints in (1c) would not have a huge impact on the techniques we propose, because we will also express the semidefinite positive (SDP) constraint (1b) through a set of linear constraints.

We rewrite (1a)-(1c) in the form below that is better suited to `Cutting-Planes`. Constraint (1b) is equivalent to (2d) when $\mathscr{D} = \mathbb{R}^n$, *i.e.*, when (2d) incorporates all cuts $\mathscr{D} = \mathbb{R}^n$, (2d) actually reduces to $X \succeq \mathbf{0}$ for $X = C - \mathcal{A}^\top \mathbf{y}$. Recall that $X \succeq \mathbf{0} \iff X \bullet \mathbf{d}\mathbf{d}^\top \geq 0\ \forall \mathbf{d} \in \mathbb{R}^n$, where $X \bullet \mathbf{d}\mathbf{d}^\top = \mathbf{d}^\top X \mathbf{d}$.

$$
(SDP\text{-}LP)\begin{cases} \max\limits_{\mathbf{y}\in\mathbb{R}^n} & \mathbf{b}^\top \mathbf{y} & (2a) \\ \quad s.t & X = C - \mathcal{A}^\top \mathbf{y} & (2b) \\ & \mathbf{a}^\top \mathbf{y} \leq c_a\ \forall (\mathbf{a}, c_a) \in \mathscr{C} & (2c) \\ & X \bullet \mathbf{d}\mathbf{d}^\top \geq 0\ \forall \mathbf{d} \in \mathscr{D} & (2d) \end{cases}
$$

In a `Cutting-Planes` scheme, both sets $\mathscr{C}$ and $\mathscr{D}$ could be generated on the fly, but this work-in-progress paper only addresses the case of a fixed (often empty) set $\mathscr{C}$. The main difficulty is to make $\mathscr{D}$ grow along the iterations, up to the point of finding the optimum solution of (1a)-(1c) using a reasonably-sized $\mathscr{D}$ in (2a)-(2d). We aim at large-scale SDP optimization, with a value of $n$ reaching a few thousands and a $k$ reaching a value of hundreds.

Interior Point Methods (IPMs) are a very popular approach for SDP optimization. These methods can offer everything one can desire in theory, but may be too slow for $n \geq 2000$, because they would have to solve huge Newton systems. Each iteration may easily end up in requiring computing a positive definite matrix of size $k \times k$ which requires $O(k^2n^2 + kn^3)$ operations, see, *e.g.*, [3, p. 66] or [8, p. 26]. The state of the art solver `Mosek` implemented one of the fastest IPM for SDP optimization.

Another very successful approach is the `ConicBundle` method [4,2] that reformulates the semidefinite program as an eigenvalue optimization problem, which is then solved by a subgradient method. This eigenvalue optimization problem arises as follows. The `ConicBundle` requires a constant trace constraint in the dual. Consider the expression $\mathcal{A}^\top \mathbf{y} = \sum_{i=1}^{k} A_i y_i$ from (2b) and suppose we have $A_k = I_n$ and $b_k > 0$. The optimal way to enforce $C - \mathcal{A}^\top \mathbf{y} \succeq \mathbf{0}$ and maximize the objective is to set $y_k = \lambda_{\min}\left(C - \sum_{i=1}^{k-1} A_i y_i\right)$. Separating this variable from the other $k-1$, one has to maximize this minimum eigenvalue function over the decision variables $y_1, y_2, \ldots, y_{k-1}$. Such methods have to maintain a cutting model that overestimates the concave non-smooth minimum eigenvalue function. The use of the term $A_k = I_n$ with $b_k > 0$ reduces to imposing $\texttt{trace}(Z) = b_k$ on the dual matrix $Z$ in the dual of (2a)-(2d) – and one can replace $k$ with a linear combination of $A_1, A_2, \ldots A_k$.

The `Projective Cutting-Planes` method proposed in this paper was deliberately designed to be as lightweight as possible, even more so than the bundle methods. Paradoxically, this is both a strength and a weakness. The weakness is that most proposed ideas are rather ad-hoc and they do not emerge in a structured manner from an established theory that has the size or strength to attract many people; this explains why the paper has few bibliographical references.

The advantage is that the new method is not placed on an existing research thread acknowledged for a long time in SDP optimization but (only) followed by highly-specialised SDP professionals. The number of people who can understand the new approach is rather large, extending (a bit) beyond the SDP field.

## 2 The general `Projective Cutting-Planes`

Let $\mathscr{P}$ be the feasible area of the semi-infinite LP (2a)-(2d). A `Cutting-Planes` algorithm constructs at each iteration $\texttt{it}$ an outer approximation $\mathscr{P}_{\texttt{it}}$ of $\mathscr{P}$, *i.e.*, a polytope $\mathscr{P}_{\texttt{it}}$ defined only by a subset of the constraints of $\mathscr{P}$, so that $\mathscr{P}_{\texttt{it}} \supseteq \mathscr{P}$. This generates a sequence of upper bounds $\mathbf{b}^\top \texttt{opt}(\mathscr{P}_{\texttt{it}})$ that decrease along the iterations $\texttt{it}$, converging to the optimal solution value $\mathbf{b}^\top \texttt{opt}(\mathscr{P})$; these bounds are associated with a series of outer non-feasible solutions $\texttt{opt}(\mathscr{P}_{\texttt{it}})$.

Any outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}})$ of (2a)–(2d) obtained this way can be turned into a feasible solution $Z$ of the dual of the main SDP program (1a)-(1c). It is enough to restrict $\mathscr{D}$ in (2d) to the current set $\overline{\mathscr{D}}$ of active constraints. In fact, each $\mathbf{d} \in \mathscr{D}$ generates a constraint (2d) has to be understood in conjunction with (2b) and implemented as $\left(A_1 \bullet \mathbf{dd}^\top\right) y_1 + \left(A_2 \bullet \mathbf{dd}^\top\right) y_2 + \ldots \left(A_k \bullet \mathbf{dd}^\top\right) y_k \leq C \bullet \mathbf{dd}^\top$. By LP duality, the objective value of the outer LP solution is the same as that of the dual LP solution $Z = \sum_{\mathbf{d}\in\overline{\mathscr{D}}} \gamma_{\mathbf{d}} \mathbf{dd}^\top \succeq \mathbf{0}$, where $\gamma_{\mathbf{d}}$ is the optimal dual value of constraint (2d) for $\mathbf{d}$; this $Z$ is also the dual SDP solution mentioned above. See also [8, Theorem 9] for more details on how a `Cutting-Planes` solving (2a)-(2d) can generate feasible dual SDP solutions along the iterations. But there is no built-in functionality in `Cutting-Planes` to generate inner feasible solutions.

To generate both inner and outer solutions (with regards to $\mathscr{P}$), `Projective Cutting-Planes` uses an iterative operation of projecting an interior point inside $\mathscr{P}$, as illustrated in Fig. 1. At each iteration $\texttt{it}$, an inner solution $\mathbf{y}_{\texttt{it}} \in \mathscr{P}$ is projected towards the direction $\mathbf{d}_{\texttt{it}}$ of the current optimal outer solution $\texttt{opt}(\mathscr{P}_{\texttt{it}-1})$, *i.e.*, we take $\mathbf{d}_{\texttt{it}} = \texttt{opt}(\mathscr{P}_{\texttt{it}-1}) - \mathbf{y}_{\texttt{it}}$. The projection sub-problem asks to determine $t^*_{\texttt{it}} = \max\{t : \mathbf{y}_{\texttt{it}} + t\mathbf{d}_{\texttt{it}} \in \mathscr{P}\}$. This requires finding the pierce (hit) point $\mathbf{y}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$ and a (first-hit) constraint of $\mathscr{P}$, which is added to the constraints of $\mathscr{P}_{\texttt{it}-1}$ to construct $\mathscr{P}_{\texttt{it}}$. At next iteration $\texttt{it}+1$, `Projective Cutting-Planes` takes a new interior point $\mathbf{y}_{\texttt{it}+1}$ on the segment joining $\mathbf{y}_{\texttt{it}}$ and $\mathbf{y}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$ and projects it towards direction $\mathbf{d}_{\texttt{it}+1} = \texttt{opt}(\mathscr{P}_{\texttt{it}}) - \mathbf{y}_{\texttt{it}+1}$. Regarding the theoretical convergence proof, see Remark 1 (p. 13) in appendix.

A central question in practice is to choose the projection base: given $\mathbf{y}_{\texttt{it}}$ and $\mathbf{y}_{\texttt{it}} + t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$, how should one choose the point $\mathbf{y}_{\texttt{it}+1} = \mathbf{y}_{\texttt{it}} + \alpha \cdot t^*_{\texttt{it}}\mathbf{d}_{\texttt{it}}$? Using $\alpha = 1$ would make `Projective Cutting-Planes` very aggressive. But previous work on combinatorial optimization LP relaxations (see, *e.g.*, Sections 2.2.2 or 3.2.1 of [7]) show that such a choice may only produce better feasible solutions in the beginning, but need more iterations in the long run. Such variant may be, however, useful if we do not need a very tight gap $\texttt{ub}_{\texttt{it}} - \texttt{lb}_{\texttt{it}} = \mathbf{b}^\top \texttt{opt}(\mathscr{P}_{\texttt{it}}) - \mathbf{b}^\top \mathbf{y}_{\texttt{it}}$; for instance, if we solve a relaxation of a combinatorial optimization problem, we can stop when $\lfloor \texttt{ub}_{\texttt{it}} \rfloor = \lfloor \texttt{lb}_{\texttt{it}} \rfloor$. Based on previous work, we decided

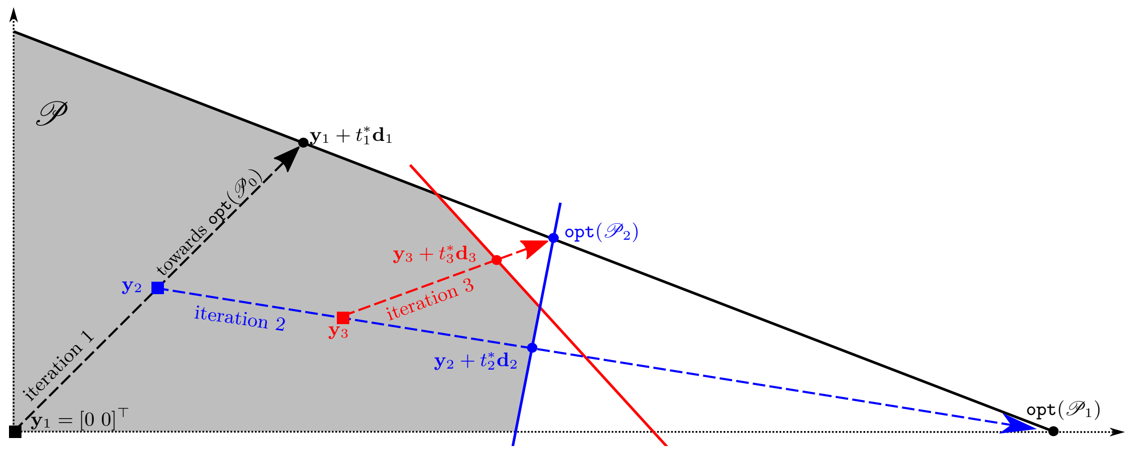


**Fig. 1.** Example of `Projective Cutting-Planes` execution (3 iterations). At the first iteration, one may projects $\mathbf{y}_1 = \mathbf{0}$ towards the optimal solution of an initial default polytope $\mathscr{P}_0$ that may contain only (very loose) bounds on $\mathbf{y}$. The projection sub-problem returns $t_1^*$ and the first-hit constraint represented by the black exterior solid line. At iteration $\mathtt{it} = 2$, if we use $\alpha = 0.5$, the midpoint $\mathbf{y}_2$ between $\mathbf{y}_1$ and $\mathbf{y}_1 + t^*\mathbf{d}_1$ is projected towards the optimal outer solution $\mathtt{opt}(\mathscr{P}_1)$ — at iteration 1, the outer approximation $\mathscr{P}_1 \supset \mathscr{P}$ contains the largest triangle. This generates a second facet (blue solid line) that is added to the facets of $\mathscr{P}_1$ to construct $\mathscr{P}_2$. The third projection in red takes the midpoint $\mathbf{y}_3$ between the blue square and the blue circle (last pierce point) and projects it towards $\mathtt{opt}(\mathscr{P}_2)$.

to use a rather conservative step length $\alpha = 0.3$; better choices may exist. In this work-in-progress paper, we start at the very first iteration with $\mathbf{y}_1 = \mathbf{0}_k$, but we have already studied other options that will be submitted for publication in a longer paper.

To determine $t^*_{\mathtt{it}} = \max\{t : \mathbf{y}_{\mathtt{it}} + t\mathbf{d}_{\mathtt{it}} \in \mathscr{P}\}$, one also has to find a *first-hit constraint* satisfied with equality by $\mathbf{y}_{\mathtt{it}} + t^*_{\mathtt{it}}\mathbf{d}_{\mathtt{it}}$. This projection sub-problem implicitly solves the separation sub-problem for all points $\mathbf{y}_{\mathtt{it}} + t\mathbf{d}_{\mathtt{it}}$ with $t \in \mathbb{R}_+$, because the above first-hit constraint separates all solutions $\mathbf{y}_{\mathtt{it}} + t\mathbf{d}_{\mathtt{it}}$ with $t > t^*_{\mathtt{it}}$ and proves $\mathbf{y}_{\mathtt{it}} + t\mathbf{d}_{\mathtt{it}} \in \mathscr{P}\ \forall t \in [0, t^*_{\mathtt{it}}]$.

By generalizing the separation sub-problem, the projection sub-problem may seem computationally far more expensive, but we will see this is not necessarily the case. Section 3 presents a few SDP techniques that bring us very close to designing a projection algorithm as fast as the separation one. Numerical experiments (Section 4) confirm that, in general, the projection sub-problem is not the most important computational bottleneck of the overall method.

The new method is reminiscent of an Interior Point Method (IPM) by the way it generates a sequence of interior points that converge to the optimal solution. An IPM moves from solution to solution by advancing along a Newton direction at each iteration, in an attempt to solve first order optimality conditions [1].

Advancing along a Newton direction is not really equivalent to performing a projection, because a projection advances up to the pierce point, while a Newton step in an IPM does not even execute all iterations to fully solve the first order conditions (for the current barrier term). An IPM tries to generate well-centered dual solutions that stay in the proximity of a central path; this is reminiscent of the trajectory of feasible solutions $\mathbf{y}_1,\ \mathbf{y}_2,\ \mathbf{y}_3,\ldots$ constructed by `Projective Cutting-Planes`.

The remaining (few) SDP customizations needed to adapt `Projective Cutting-Planes` to an SDP context are listed in Remark 4 (p. 14 in appendix).

Before presenting the SDP projection (Section 3), recall the semi-infinite LP (2a)-(2d) may contain two sets of constraints that may be generated on the fly: (2c) end (2d). When necessary, one may have to solve two projection sub-problems, a non-SDP one with regards to (2c) and an SDP one with regards to (2d). The space limitation does not enable us to advance more on this idea, but, to our knowledge, such questions are out of reach for other SDP algorithms.

## 3 The SDP projection algorithm

The `Projective Cutting-Planes` was initially designed and tested independently of any SDP concept, for the purpose of solving LP relaxations in combinatorial optimization. This is the first time we solve the projection sub-problem over the SDP cone: what is the maximum $t^*$ so that $X + t^*D \succeq \mathbf{0}$? In our context, $X$ is the SDP matrix $X = C - \mathcal{A}^\top \mathbf{y}_{in} \succeq \mathbf{0}$ associated to the current inner point $\mathbf{y}_{in}$ of (2a)-(2d) and $D = -\mathcal{A}^\top(\mathbf{y}_{out} - \mathbf{y}_{in})$, where $\mathbf{y}_{out}$ is the current outer point. We also have to determine a first-hit vector $\mathbf{v} \in \mathbb{R}^n$ so that $(X + t^*D) \bullet \mathbf{v}\mathbf{v}^\top = 0$. In practice, we may easily encounter numerical problems and this equality will always be satisfied within a certain tolerance. On the other hand, the value $D \bullet \mathbf{v}\mathbf{v}^\top$ should be really significantly lower than 0. When this is the case, advancing any $\epsilon > 0$ beyond $t^*$ leads to $(X + (t^* + \epsilon)D) \bullet \mathbf{v}\mathbf{v}^\top < 0$.

*Property 1.* We will see that the projection $X \to D$ can be calculated more rapidly if $D$ belongs to the image of $X$. This means that each column (and row) of $D$ can be written as a linear combination of the columns (or rows, resp.) of $X$. We can equivalently say that the null space of $X$ is included in the null space of $D$; thus, $X\mathbf{d} = 0 \implies D\mathbf{d} = 0\ \forall \mathbf{d} \in \mathbb{R}^n$. We will show below in cases A) and B) how it is easier to project when this property holds; if possible, `Projective Cutting-Planes` should thus adapt its own evolution to seek this property.

Two matrices $X$ and $X'$ are congruent if there is some non-singular $M$ such that $X' = MXM^\top$. It is well known (see, for example, [6, Prop 1.2.3.]) that two congruent matrices have the same SDP status: $X \succeq \mathbf{0} \iff X' \succeq \mathbf{0}$.

*Property 2.* (congruent expansion) We say that $X' \in \mathbb{R}^{n' \times n'}$ with $n' > n$ is a congruent expansion of $X \in \mathbb{R}^{n \times n}$ if and only if we can write $X' = MXM^\top$, for some $M \in \mathbb{R}^{n' \times n}$ of *full rank* $n$. $X$ has the same SDP status as $X'$.

*Proof.* We show both implications below.

1. $X \succeq \mathbf{0} \implies X' \succeq \mathbf{0}$. Assume the contrary for the sake of contradiction: $\exists \mathbf{v}' \in \mathbb{R}^{n'}$ such that $\mathbf{v}'^\top X' \mathbf{v}' < 0$. This implies $\mathbf{v}'^\top M X M^\top \mathbf{v}' < 0$, equivalent to $X \not\succeq \mathbf{0}$, contradiction.
2. $X' \succeq \mathbf{0} \implies X \succeq \mathbf{0}$ Assume the contrary: $\exists \mathbf{v} \in \mathbb{R}^n$ such $\mathbf{v}^\top X \mathbf{v} < 0$. We can surely write $\mathbf{v}^\top = \mathbf{v}'^\top M$ for some $\mathbf{v}' \in \mathbb{R}^{n'}$ because $M$ has full rank. This means $\mathbf{v}'^\top M X M^\top \mathbf{v}' < 0$, equivalent to $\mathbf{v}'^\top X' \mathbf{v}' < 0$, contradiction. □

Using these concepts we are ready to address the projection algorithm. We will distinguish four cases noted A), B), C) and D). Given $X \succeq \mathbf{0}$, we have to find:

$$\max \{t : X + tD \succeq \mathbf{0}\}. \tag{3}$$

---

A) This case is characterized by $X \succ \mathbf{0}$, *i.e.*, $X$ is non-singular; Prop 1 surely holds because the image of a non-singular $X$ is $\mathbb{R}^n$. We apply the Cholesky decomposition to determine the unique non-singular $K$ such that $X = KK^\top$. We then solve $D = KD'K^\top$ in variables $D'$ by back substitution; this may require $O(n^3)$ in theory, but `Matlab` is able to compute it much more rapidly in practice because $K$ is triangular. Let us re-write (3) as:

$$\max \left\{t : KI_nK^\top + tKD'K^\top \succeq \mathbf{0}\right\}. \tag{4}$$

This is equivalent (by congruence according to Prop 2) to

$$\max \{t : I_n + tD' \succeq \mathbf{0}\}. \tag{5}$$

The sought step length is $t^* = -\frac{1}{\lambda_{\min}(D')}$, or $t^* = \infty$ if $\lambda_{\min}(D') \geq 0$.

We still have to find a first-hit cut $\mathbf{v} \in \mathbb{R}^n$; in fact, technically, the first-hit cut will be $\left(A_1 \bullet \mathbf{v}\mathbf{v}^\top\right) y_1 + \left(A_2 \bullet \mathbf{v}\mathbf{v}^\top\right) y_2 + \cdots + \left(A_k \bullet \mathbf{v}\mathbf{v}^\top\right) y_k \leq C \bullet \mathbf{v}\mathbf{v}^\top$.

If $\mathbf{v}$ is an eigenvector of $K(I_n + t^*D')K^\top$ with an eigenvalue of 0, this means $\mathbf{v}^\top K(I_n + t^*D')K^\top \mathbf{v} = 0$. Thus, $\mathbf{u} = K^\top \mathbf{v}$ is eigenvector of $I_n + t^*D'$ with an eigenvalue of 0. This latter eigenvector $\mathbf{u}$ can be computed when determining $\lambda_{\min}(D') < 0$ above, because if the eigenvalue of $\mathbf{u}$ with regards to $D'$ is $\lambda_{\min}(D')$ its eigenvalue with regards to $I_n + t^*D'$ is 0 (since recall $t^* = -\frac{1}{\lambda_{\min}(D')}$). The sought $\mathbf{v}$ solves $K^\top \mathbf{v} = \mathbf{u}$ and it can rapidly be computed by back-substitution. We have $\mathbf{u}^\top D' \mathbf{u} < 0 \implies \mathbf{v}^\top KD'K^\top \mathbf{v} < 0 \implies \mathbf{v}^\top D \mathbf{v} < 0$. We thus have $\mathbf{v}^\top (X + t^*D)\mathbf{v} = 0$ and $\mathbf{v}^\top (X + (t^* + \epsilon)D)\mathbf{v} < 0$ for any $\epsilon > 0$. This proves $\mathbf{v}$ is a first-hit cut.

---

B) In this case Prop 1 is still satisfied, but $X$ has rank $c < n$. This means $X$ contains $c$ independent rows (and columns by symmetry), referred to as *core* rows (or columns); the other dependent rows (or columns) are *non-core* positions. Using the LDL decomposition of $X$, we will factorize $X = K_{nc}K_{nc}^\top$, where $K_{nc} \in \mathbb{R}^{n \times c}$. The image of $K_{nc}$ is equal to the image of $X$. Since Prop 1 is satisfied, we will see we can still solve $D = K_{nc}D'K_{nc}^\top$ in variables $D'$. A first intuition is to notice that we can project $X \to D$ only over the core rows and columns, because the non-core positions are dependent on the core ones.

But the most difficult task is to determine these core positions. We first apply the LDL decomposition and write $X = L\texttt{diag}(\mathbf{p})L^\top$ with $\mathbf{p} \geq \mathbf{0}_n$. The contribution of each $p_i$ in $L\texttt{diag}(\mathbf{p})L^\top$ is actually $p_i L_i L_i^\top$, where $L_i$ is column $i$ of $L$ ($\forall i \in [1..n]$). If all $n \times n$ elements of $p_i L_i L_i^\top$ are below some precision parameter, we consider $i$ is a non-core position; otherwise, it is a core position. By reducing all non-core positions $p_i$ to zero, we can say that all $n-c$ non core columns of $L$ vanish in the decomposition $X = L\texttt{diag}(\mathbf{p})L^\top$. After removing these vanished $n-c$ columns from $L$ and the corresponding zeros from $\mathbf{p}$, we can write $X = L\texttt{diag}(\mathbf{p})L^\top = L\texttt{diag}(\mathbf{p})^{\frac{1}{2}}\texttt{diag}(\mathbf{p})^{\frac{1}{2}}L^\top = K_{nc}K_{nc}^\top$ with $K_{nc} \in \mathbb{R}^{n\times c}$.

We next solve $D = K_{nc}D'K_{nc}^\top$ in variables $D'$. For this, we first reduce this system to work on $c \times c$ matrices, *i.e.*, we transform it into $D_{cc} = K_{cc}D'K_{cc}$ where $K_{cc}$ is $K_{nc}$ restricted to the $c$ core rows and $D_{cc}$ is $D$ restricted to the $c \times c$ core rows and columns. To solve this square system, we apply back-substitution twice and this is very fast because $K_{cc}$ is lower triangular. If the resulting solution $D'$ also satisfies $D = K_{nc}D'K_{nc}^\top$, then we are surely in case B). We obtained a reduced-size version of (5) working in the space of $c \times c$ matrices:

$$\max\left\{t : I_c + tD' \succeq \mathbf{0}\right\}. \tag{6}$$

And the maximum value of $t$ is here: $t^* = -\frac{1}{\lambda_{\min}(D')}$, or $t^* = \infty$ if $\lambda_{\min}(D') \geq 0$.

We finally determine a first-hit vector $\mathbf{v}_c \in \mathbb{R}^c$ over the core rows and columns exactly like in (the last paragraph describing) case A). To lift $\mathbf{v}_c$ to a hit vector $\mathbf{v} \in \mathbb{R}^n$, we construct $\mathbf{v}$ by inheriting the core positions from $\mathbf{v}_c$ and filling the non-core positions with zeros.

---

C) We still use the decomposition $X = K_{nc}K_{nc}^\top$ computed above but we suppose that the system $D = K_{nc}D'K_{nc}^\top$ has no solution in variables $D'$. This also means Prop 1 is not satisfied: $D$ does not belong to the image of $K_{nc}$ or $X$.

We will express all columns of $D$ as a linear combination of: (i) the columns of $K_{nc}$ and (ii) a set of $m$ columns of $D$ named active (independent) columns. We first apply the QR decomposition on matrix $[K_{nc}\ D] \in \mathbb{R}^{n\times(c+n)}$ and write $[K_{nc}\ D] = QR$, where $Q \in \mathbb{R}^{n\times(c+n)}$ and $R \in \mathbb{R}^{(c+n)\times(c+n)}$ is upper triangular. In fact, the standard QR factorization returns a matrix $Q \in \mathbb{R}^{n\times n}$ and a matrix $R \in \mathbb{R}^{n\times(c+n)}$, but we artificially extend $Q$ with $c$ null columns and $R$ with $c$ null rows to simplify notations. Let us focus on the first $c$ columns of $R$. Since $K_{nc}$ is full rank, the matrix $R$ restricted to the first $c$ columns will be full rank; since it is upper triangular, this means $R_{jj} \neq 0$ for all $j \in [1..c]$.

Now focus on row $c+i$ of $R$ for each $i \in [1..n]$. If all elements of this row are zero, column $c+i$ of $Q$ it has no contribution in the product $QR$; this also means that column $c+i$ of $[K_{nc}\ D]$ can be expressed as a combination of the first $c+i-1$ columns of $Q$. We call this column of $Q$ non-active, being dependent on the columns of $K_{nc}$ and on the active columns found while scanning the columns $[c+1, c+i-1]$ of $Q$.

Let $N$ denote the matrix $Q$ restricted to its $m$ active columns detected above. The size of $N$ provides a new way to detect case B): if $N$ were empty with $m = 0$, we would have been in case B). When $N$ is non-empty, we can decompose $X$

and $D$ as follows:

$$X = \underbrace{[K_{nc}\ N]}_{c\ +m} \begin{bmatrix} I_c & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} K_{nc}^\top \\ N^\top \end{bmatrix} \tag{7}$$

$$D = \underbrace{[K_{nc}\ N]}_{c\ +\ m} \underbrace{\begin{bmatrix} F & G^\top \\ G & E \end{bmatrix}}_{D_{c+m}} \begin{bmatrix} K_{nc}^\top \\ N^\top \end{bmatrix} \tag{8}$$

$$= K_{nc} F K_{nc}^\top + NEN^\top + K_{nc} G^\top N^\top + NGK_{nc}^\top \tag{9}$$

The hardest computational task is computing $D_{c+m}$. A straightforward approach may be quite slow. We prefer to exploit again the information determined by the QR decomposition. We will modify both sides of the factorization $[K_{nc}\ D] = QR$ to make it similar to (8). To transform $Q$ into $[K_{nc}\ N]$, we write $Q = [Q_{nc}\ Q_{n,c+1..n}]$, *i.e.*, we split its first $c$ columns $Q_{nc}$ from the last $n$ columns $Q_{n,c+1..n}$. We can write $K_{nc} = Q_{nc} R_{cc}$, where $R_{cc}$ is the $c \times c$ top-left part of $R$. Since this system is full rank, we obtain $Q_{nc} = K_{nc} R_{cc}^{-1}$. We can thus write $Q = [K_{nc}\ Q_{n,c+1..c+n}] \begin{bmatrix} R_{cc}^{-1} & \mathbf{0} \\ \mathbf{0} & I_n \end{bmatrix}$. Replacing this $Q$ in $[K_{nc}\ D] = QR$, we obtain $[K_{nc}\ D] = [K_{nc}\ Q_{n,c+1..n}] \begin{bmatrix} R_{cc}^{-1} & \mathbf{0} \\ \mathbf{0} & I_n \end{bmatrix} R$. We now compute the last $n$ columns of the right multiplication and denote the result by $C \in \mathbb{R}^{(n+c)\times n}$. If we also restrict $[K_{nc}\ D]$ to its last $n$ columns (*i.e.*, to $D$), the above QR factorization becomes:

$$D = [K_{nc}\ Q_{n,c+1..c+n}]C \tag{10}$$

The matrix $N$ is simply $Q_{n,c+1..c+n}$ restricted to the $m$ active columns identified above – recall a non-active column $c+i$ of $Q$ has no contribution in the $QR$ product since row $i$ of $R$ is null. The left factor $[K_{nc}\ Q_{n,c+1..c+n}]$ in (10) is thus reduced to $[K_{nc}\ N]$ by removing the non-active columns of $Q$. The associated $C$ factor in (10) is also reduced to some $\overline{C}$ by removing its null rows that come from the null rows of $R$. We can re-write (10) as $D = [K_{nc}\ N] \cdot \overline{C}$. We finally determine $D_{c+m}$ from (8) by solving $D_{c+m} \begin{bmatrix} K_{nc}^\top \\ N^\top \end{bmatrix} = \overline{C}$. We recall $D_{c+m}$ has the form:

$$D_{c+m} = \begin{bmatrix} F & G^\top \\ G & E \end{bmatrix}$$

The case C) under discussion here is characterized by the fact that $G = \mathbf{0}$. Applying (9), this means $D$ has the form $D = K_{nc} F K_{nc}^\top + NEN^\top$, where $N$ is orthogonal to $K_{nc}$ by (the QR decomposition) construction.[1] Using the congruence expansion Property 2 on (7)-(8), the SDP status of $X + t \cdot D$ is the same as that of

$$\begin{bmatrix} I_c & 0 \\ 0 & 0 \end{bmatrix} + t \cdot \begin{bmatrix} F & \mathbf{0} \\ \mathbf{0} & E \end{bmatrix}. \tag{11}$$

[1] As a side remark, we can show that $XD$ belongs to the column image of $X$ in this case C). Since $XD = K_{nc} K_{nc}^\top (K_{nc} F K_{nc}^\top + NEN^\top)$ and $K_{nc}^\top N = 0$, we have $X\mathbf{d} = 0 \implies K_{nc} K_{nc}^\top \mathbf{d} = 0 \implies K_{nc}^\top \mathbf{d} = 0 \implies XD\mathbf{d} = 0,\ \forall \mathbf{d} \in \mathbb{R}^n$.

Any hit-vector $\mathbf{v}' \in \mathbb{R}^{c+m}$ for the projection problem $\left[\begin{smallmatrix} I_c & 0 \\ 0 & 0 \end{smallmatrix}\right] \to \left[\begin{smallmatrix} F & \mathbf{0} \\ \mathbf{0} & E \end{smallmatrix}\right]$ can be lifted to a hit-vector $\mathbf{v} \in \mathbb{R}^n$ for the original projection $X \to D$ by finding a solution $\mathbf{v}$ of the underdetermined system $\mathbf{v}' = \left[\begin{smallmatrix} K_{nc}^\top \\ N^\top \end{smallmatrix}\right] \mathbf{v}$. We can hereafter only focus on projecting $\left[\begin{smallmatrix} I_c & 0 \\ 0 & 0 \end{smallmatrix}\right] \to \left[\begin{smallmatrix} F & \mathbf{0} \\ \mathbf{0} & E \end{smallmatrix}\right]$. Case C) is split in two cases:

**C.1)** if $E \succeq \mathbf{0}$, the above projection reduces to determining $\max\{t : I_c + tF \succeq \mathbf{0}\}$ and this is solved using (3) as in case A).

**C.2)** if $E \not\succeq \mathbf{0}$, the sought $t^*$ is 0. It is straightforward to see in (11) that any $t > 0$ would generate in this case a non SDP matrix.

---

D) If all above cases fail, we still apply the logic of (11) and solve the projection by finding the maximum $t$ such that:

$$\underbrace{\begin{bmatrix} I_c & 0 \\ 0 & 0 \end{bmatrix}}_{X_{c+m} \in \mathbb{R}^{(c+m)\times(c+m)}} + t \cdot \underbrace{\begin{bmatrix} F & G \\ G & E \end{bmatrix}}_{D_{c+m} \in \mathbb{R}^{(c+m)\times(c+m)}} \succeq \mathbf{0}. \tag{12}$$

We first present a tricky case. If there is some $i \in [c+1..m]$ and some $j \in [1..c]$ such that the diagonal element $(i,i)$ of $D_{c+m}$ is zero while its non-diagonal element $(i,j)$ is non-zero, then any $t > 0$ leads to $X_{c+m} + tD_{c+m} \not\succeq \mathbf{0}$. We return $t^* = 0$, but there is no hit vector $\mathbf{v}$ such that $(X_{c+m} + tD_{c+m}) \bullet \mathbf{v}\mathbf{v}^\top < 0 \forall t > 0$. Because if we reduce the whole projection to rows and columns $i$ and $j$, there is no vector $\mathbf{v} \in \mathbb{R}^2$ such that $\left[\begin{smallmatrix} 1 & t \\ t & 0 \end{smallmatrix}\right] \bullet \mathbf{v}\mathbf{v}^\top < 0$ for any $t > 0$ no matter how small.

If all possibilities discussed up to here fail, we solve the projection in two steps: (1) find a small $t_1$ such that $X_{c+m} + t_1 D_{c+m} \succeq \mathbf{0}$ and (2) solve the projection $(X_{c+m} + t_1 D_{c+m}) \to D_{c+m}$. In this second step, $D_{c+m}$ belongs to the image of $X_{c+m} + t_1 D_{c+m}$ (Prop 1 satisfied) and we will use case A) or B). However, finding $t_1$ may require a limited number of (costly) repeated separations. This case is virtually never needed in the experiments presented in this paper and we explore it further in appendix (Remark 2, p. 13).

## 4 Numerical results

There is unfortunately no well-established benchmark for testing SDP algorithms and no universally-accepted methodology to measure their performance. Most testing has been carried out in rather disparate contexts. Since we consider the most general SDP programs (no sparsity and no particular combinatorial structure behind the involved matrices), we simply generated the instances as follows. First, we constructed $\frac{k}{2}$ eigenvectors meant to become $0-$eigenvalue eigenvectors (with an eigenvalue of 0) for the matrices $A_1, A_2, \ldots, A_k$ and $C$; each such eigenvector is inserted in each of these matrices with a probability of 0.8. Once $n_0$ such $0-$eigenvalue eigenvectors are fixed for a given matrix, we construct at random $n - n_0$ orthogonal eigenvectors (that together constitute a basis of $\mathbb{R}^n$). In a first instance set, we generate the eigenvalues of these $n - n_0$ eigenvectors randomly between 9 and 10 for $A_1, A_2, \ldots, A_k$ and between 30 and 50 for $C$.

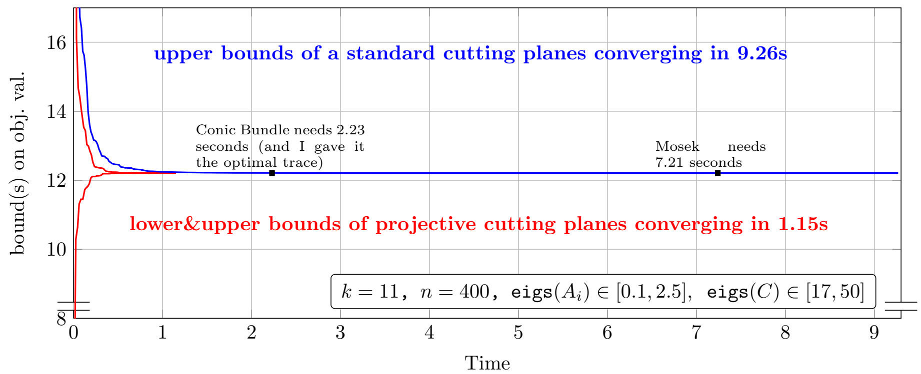


**Fig. 2.** A sample run comparing the main software considered in this paper.

In a second instance set, these eigenvalues have larger variations (indicated by Column 3 of Table 2). We set $\mathbf{b} = \mathbf{1}$ when not stated otherwise.

Figure 2 illustrates a comparison between the new method, the standard `Cutting-Planes`, the `ConicBundle` and the `Mosek` solver. This figure confirms the standard `Cutting-Planes` is too slow. `Mosek` is not very fast for such a low $k$ and large $n$. The `ConicBundle` needs a bit more than 2 seconds, around twice as much as `Projective Cutting-Planes`. In this paper we stop `Projective Cutting-Planes` when the `ub-lb` gap is below 0.00001, but notice that after 0.33 seconds this gap was already hardly noticeable on this figure. A rather loose gap may be satisfactory when we solve a relaxation of combinatorial optimization problem that has an integer optimum. Since the `ConicBundle` reformulates (2a)-(2d) as an eigenvalue optimization problem, it needs as input the trace of the optimal dual solution of (2a)-(2d); we offered it this artificial advantage by inserting matrix $A_{k+1} = I_n$; see full details in appendix (Remark 3, p. 14).

Table 1 reports the wall running times of `Projective Cutting-Planes`, `ConicBundle` and `Mosek` on the first instance set. These are the most time-consuming operations observed on our standard laptop (described by Remark 5).

(a) Determine $X$ and $D$ at each iteration. This operation has complexity $O(kn^2)$ while many calculations of the projection algorithm have a complexity of $O(n^3)$. Yet these latter calculations use very strongly-optimized `Matlab` routines, while computing $X$ or $D$ can not benefit from such routines, since this is not a very classical matrix operation. We are almost certain we will improve this situation in future versions of the software.

(b) Solve the projection sub-problem $X \to D$. Table 1 show that this may often represent less than 10% of the total running time. To our surprise, the operation from Point (a) is often more computationally expensive.

(c) Solve the LP corresponding to the outer approximation of the feasible SDP area (2a)-(2d). This step is relatively insignificant for $k < 50$, but it becomes expensive as $k$ is increased towards 100. The speed of `Projective Cutting-Planes` for a (much) larger $k$ is dependent on the LP solver (`cplex`); any future progress in linear programming may bring positive consequences.

| k \ n | 400 | 600 | 800 | 1000 | 1200 | 2000 (gap $ub$-$lb$) |
|---|---|---|---|---|---|---|
| 20 | $\frac{2.8(0.7/0.6/0.3)}{9.5/10.6}$ | $\frac{7.1(2.4/1.6/0.4)}{18.2/34}$ | $\frac{11(4/2.4/0.3)}{57.5/80}$ | $\frac{26(10/7/0.5)}{125/134}$ | $\frac{48(18/13/0.5)}{220/265}$ | $\frac{\text{gap closed}}{\text{n.a.}/0.044}$ |
| 30 | $\frac{7.8(2.6/1.6/0.9)}{26.8/15.3}$ | $\frac{22(9/3/0.2)}{119/43}$ | $\frac{46(20/8/2.4)}{134/97}$ | $\frac{55(25/10/1.1)}{155/211}$ | $\frac{125(55/23/2)}{156/405}$ | $\frac{\text{gap closed}}{\text{n.a.}/0.16}$ |
| 40 | $\frac{18(6/2/4)}{108/20}$ | $\frac{26(10/3/2)}{127/57}$ | $\frac{65(30/8/4)}{307/134}$ | $\frac{169(83/21/6)}{391/267}$ | $\frac{253(123/34/5)}{664/479}$ | $\frac{\text{gap closed}}{\text{n.a.}/0.27}$ |
| 50 | $\frac{69(20/5/25)}{130/25}$ | $\frac{65(30/8/3)}{160/80}$ | $\frac{189(88/17/23)}{410/169}$ | $\frac{297(151/31/15)}{863/391}$ | $\frac{462(237/49/15)}{770/712}$ | $\frac{0.00023}{\text{n.a.}/0.94}$ |
| 60 | $\frac{76(22/4/28)}{128/28}$ | $\frac{178(68/11/47)}{136/85}$ | $\frac{239(112/19/28)}{658/403}$ | $\frac{520(270/53/37)}{1081/736}$ | $\frac{614(318/60/23)}{1244/975}$ | $\frac{0.0021}{\text{n.a.}/5.05}$ |
| 70 | $\frac{106(34/5/38)}{150/35}$ | $\frac{212(70/11/72)}{474/102}$ | $\frac{628(248/33/187)}{411/735}$ | $\frac{570(304/45/45)}{2283/842}$ | $\frac{1297(674/103/87)}{2669/1152}$ | $\frac{0.017}{\text{n.a.}/25.5}$ |

**Table 1.** `Projective Cutting-Planes` compared to `ConicBundle` [2] and `Mosek`. For each instance, we provide the total wall running time (seconds) under the form $\frac{p(a/b/c)}{e/f}$, where $p$ is the total of wall time of `Projective Cutting-Planes`, $a$ is the time of computing $X$ and $D$, $b$ counts the projection time, $c$ is the time of the LP solver for the outer approximation of (2a)-(2d); $e$ is the total `ConicBundle` time and $f$ it the total `Mosek` time. The last column concerns a huge instance size and is different: it provides for each algorithm the gap $\texttt{ub} - \texttt{lb}$ reported after 1000 seconds. In fact, we only report this for our method (the numerator) and `Mosek`; "n.a." means not available for the `ConicBundle`, because it does not compute such intermediate $\texttt{ub} - \texttt{lb}$ values.

Table 1 is not meant to show that `Projective Cutting-Planes` is clearly superior to all other alternatives on all or most instances. While we aim at being very competitive in speed, this work is not a competition paper; we find such quest quite absurd. The three compared algorithms rely on different philosophies. The speed of `Projective Cutting-Planes` depends on the way `Matlab` implements certain basic operations (like Cholesky or QR factorizations, back-substitution, matrix multiplication, etc). Most of these building blocks have a theoretical complexity of $O(n^3)$ but their running time in `Matlab` (version 2018) seem closer to $O(n^2)$. This explains why the last column of Table 1 suggest that `Projective Cutting-Planes` is the most competitive method for $n \geq 2000$.

Table 2 next page compares `Projective Cutting-Planes` with the `ConicBundle` on the second benchmark set with instances of more varied sizes and of a different nature (regarding the spectrum of the $A_i$'s or the non-negativity of $\mathbf{y}$). Switching to $\mathbf{y} \geq 0$ may heavily reduce the number of `Projective Cutting-Planes` iterations because most of the elements of the optimal $\mathbf{y}$ may be zero. In some cases, even if $k$ reaches a value of thousands, the associated LPs remain very easy in practice because many of the $\mathbf{y}$ variables may remain zero at optimality when $\mathbf{y} \geq \mathbf{0}$. The last four rows of this table suggest `Projective Cutting-Planes` is the best method for very large SDP programs.[2]

We presented up to here only the most relevant benchmarking information we could present in a 12-pages paper. But the results reported in this work-in-

[2] We provided the optimal trace to the `ConicBundle` in the run from the last column. Since the optimal trace is unknown in advance, we determined it from the `ConicBundle` run from the next-to-last column where we only used a bounded trace constraint (as in Remark 3, p. 14). We are fully conscious that a better implementation of this optimal trace constraint may speed up the `ConicBundle`.

| Instance | | | | Projective Cutting-Planes | | | | | | ConicBundle | | Mosek |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | k | Eigs $A_i$'s | Eigs C | Itera-tions | All time | Compute $X$ & $D$ | Proj time | LP time (cplex) | Send data to LP | Trace unknown | Trace provided | |
| 800 | 80 | [-20, 100] | [0,100] | 1108 | 410 | 179 | 44 | 70 | 102 | 1051 | 94 | 320 |
| 600 | 40 | [-20, 100] | [0,100] | 155 | 17 | 4 | 6 | 1 | 3 | 148 | 22 | 72 |
| 400 | 100 | [-20, 100] | [0,100] | 2075 | 572 | 94 | 13 | 384 | 71 | 490 | 42 | 60 |
| Huge instances below have $\mathbf{y} \geq 0$, a random $\mathbf{b}$ and $\frac{n}{5}$ fixed null eigenvectors for all $A_i$'s and $C$ | | | | | | | | | | | | |
| 200 | 2000 | [40, 100] | [10,40] | 31 | 11 | 5 | 0.2 | 0.2 | 5 | timeout | | 717 |
| 200 | 3000 | [40, 100] | [10,40] | 70 | 49 | 27 | 0.4 | 0.7 | 18 | timeout | | 1346 |
| 4000 | 20 | [20,25] | [20,25] | 8 | 76 | 17 | 44 | 0 | 11 | timeout | | timeout |
| 5000 | 20 | [20,25] | [20,25] | 7 | 139 | 27 | 87 | 0 | 18 | timeout | | timeout |

**Table 2.** Seven runs of `Projective Cutting-Planes`, `ConicBundle` and `Mosek` on more varied instances. The last four instances have $\mathbf{y} \geq \mathbf{0}$; such linear constraints on $\mathbf{y}$ simplify the problem for `Projective Cutting-Planes`, but this may be a non-trivial change for `ConicBundle` (or other algorithms that do not embed the SDP problem in a lightweight LP over $\mathbf{y}$).

progress article are not a perfect measure of the final potential of the projection idea. This work represens the most initial version of the proposed method, submitted for the very first time to peer review. We must confess such software can not be perfect, because it was not thoroughly tested. Perhaps other SDP algorithms out there invested 1000 times more coding and software testing resources. However, it is quite safe and easy to check the correctness of a lower bound $\mathbf{b}^\top \overline{\mathbf{y}}$ reported by `Projective Cutting-Planes`: it is enough to check that the minimum eigenvalue of $C - \mathcal{A}^\top \overline{\mathbf{y}}$ is not-negative. It is very difficult to have errors in the upper bounds either, because any $\mathbf{d} \in \mathbb{R}^n$ provides a valid cut (2d) and each upper bound is simply computed by the LP solver that optimizes over all cuts (2d) provided all along the iterations.

## 5 Conclusion and prospects

We used `Projective Cutting-Planes` ideas [5] to propose a fast method for optimizing (very) large SDP programs. Many ideas go beyond SDP optimization, because the considered SDP program is incorporated in a more general (and yet very simple) LP. For example, the `Cutting-Planes` logic for solving this LP enables one to easily insert some initial linear constraints (2c) in the main SDP problem (2a)-(2d). If these linear constraints are prohibitively-many, they could even be generated on the fly by solving a second projection sub-problem in a purely LP context. We plan to implement this idea on a robust SDP problem in which the coefficients of the nominal constraints (2c) can vary according to some robust rules and produce prohibitively-many robust cuts – a projection algorithm for this robust LP is already available [7, Sec. 2.1]. It is quite easy to adapt `Projective Cutting-Planes` to perform certain re-optimization tasks like the following: after solving a (2a)-(2d) program, solve the same program again after adding a new LP (or SDP) constraint. We are not aware of other methods that can adapt so easily to address such questions.

## References


1. Gondzio, J.: Interior point methods 25 years later. European Journal of Operational Research **218**(3), 587–601 (2012)
2. Helmberg, C.: The conicbundle library for convex optimization, `www-user.tu-chemnitz.de/~helmberg/ConicBundle/`
3. Helmberg, C.: Semidefinite programming for combinatorial optimization. Ph.D. thesis, Technische Universität Berlin (2000), `https://www-user.tu-chemnitz.de/~helmberg/home.html`
4. Helmberg, C., Rendl, F.: A spectral bundle method for semidefinite programming. SIAM Journal on Optimization **10**(3), 673–696 (2000)
5. Porumbel, D.: `Projective Cutting-Planes`. SIAM Journal on Optimization **30**(1), 1007–1032 (2020)
6. Porumbel, D.: Demystifying the characterizations of sdp matrices in mathematical programming (2022), `cedric.cnam.fr/~porumbed/papers/sdp.pdf`
7. Porumbel, D.: Further experiments and insights on `Projective Cutting-Planes`. INFORMS Journal on Computing **34**(5) (2022)
8. Sivaramakrishnan, K.K.: Linear programming approaches to semidefinite programming problems. Ph.D. thesis, Rensselaer Polytechnic Institute (2002), `https://homepages.rpi.edu/~mitchj/phdtheses/kartik/rpithes.pdf`


## A More insights into the design and the implementation of `Projective Cutting-Planes`

While the key element of this work is the projection sub-problem, the overall implementation depend on many other (down-to-earth) factors. The main paper presented only the most important guidelines for understanding `Projective Cutting-Planes`, but it is not possible to discuss all nuts and bolts of the method. This appendix provides a number of remarks that completes the description of certain components of `Projective Cutting-Planes`.

*Remark 1.* In theory, the feasible area $\mathscr{P}$ of (2a)-(2d) is not a polytope. But if we consider in (2d) only constraints $\mathbf{d}$ with a finite number of digits, this feasible area becomes a polytope. As long as the amount of memory available on Earth is finite, the infinite number of SDP cuts is actually finite when one solves (2a)–(2d) with an earthly computer. A `Projective Cutting-Planes` iteration `it` either returns a new cut never discovered before or stops by proving $\mathtt{opt}(\mathscr{P}_{\mathtt{it}})$ is optimal (with $t^* = 1$). Considering a finite number of potential SDP cuts, the algorithm will converge in a finite number of iterations. We could go into more technical questions on convergence proofs, but such techniques are not directly relevant to the core of our algorithms. □

*Remark 2.* Projecting $X \to D$ is equivalent to projecting $X_{c+m} \to D_{c+m}$ using (12) as discussed at point D) of the projection algorithm (p. 9). It may be faster to use the smaller matrices $X_{c+m}$ and $D_{c+m}$ of order $c + m < n$. On the other hand, not working with the original matrices may lead to more numerical problems. We here limit the presentation to the case in which we apply repeated separation on the original matrices $X$ and $D$: we have to use repeated separation to determine the SDP status of $X + tD$ for various values of $t$.

We consider a user-provided list of separation points $t_1,\ t_2,\ t_3,\ldots$ to be tried so that $0 < t_1 < t_2 < t_3 \ldots$; for each such $t_i$, we solve the separation sub-problem by determining the minimum eigenvalue of $X + t_i D$. If this value is negative, $X + t_i D$ does not belong to the SDP cone. We now split case D) in two sub-cases:

**D.1)** If $X + t_1 D \not\succeq \mathbf{0}$ we return $t^* = 0$. It is important to have a $t_1$ value (very) close to 0. We basically consider that there is no space inside the SDP cone to perform any positive step towards $D$ only because there is no space to perform a step of $t_1$. By using a $t_1$ close to $10^{-6}$ we avoid many numerical problems when solving case D.1) this way.

**D.2)** If $X + t_1 D \succeq \mathbf{0}$, we return $t^* = t_1 + t_2^*$, where $t_2^*$ is the step length returned by projecting $(X + t_1 D) \rightarrow D$. But the advantage of this new projection is that $D$ will belong to the image of $X + t_1 D$, except in very pathological cases. This way, Property 1 is very likely to hold and we can solve the projection using cases A) or B). □

*Remark 3.* To provide a constant trace constraint for the `ConicBundle` in the dual of (2a)-(2d), we insert into the primal (2a)-(2d) an additional $A_{k+1} = I_n$ alongside a $b_{k+1}$ equal to the trace value. Since the trace is unknown in our experiments, we can only provide it by solving the instance beforehand. `Projective Cutting-Planes` could have also exploited such information to produce more interior feasible solutions. However, in all `ConicBundle` experiments with an unknown trace, we indicated to `ConicBundle` that this trace is $\leq 1000$. For this, we added a row and column of zeros to all matrices $A_1, A_2, \ldots A_{k+1}$ and $C$, putting a 1 only at position $(n+1, n+1)$ of $A_{k+1}$ so that $A_{k+1}$ becomes $I_{n+1}$, and $b_{k+1} = 1000$. We are fully conscious that a better implementation of this optimal trace constraint may speed up the `ConicBundle`. □

*Remark 4.* The most important customizations of `Projective Cutting-Planes` that were not fully described in the main body of the paper (due to space limitation) are the following.

- In the very beginning there is no default constraint that `Projective Cutting-Planes` may use to construct a very first outer approximation of (2a)-(2d) or a very first outer solution. We inserted an artificial initial box to have such a first outer approximation. This box only limits each variable $y_i$ to the interval [-100000,100000], which is more than enough for our instances. In the very beginning, while the current $\mathbf{y}$ still touches the box, we use standard `Cutting-Planes` (this never took an important amount of time compared to overall running time).
- Recall that in cases A) and B) we computed the minimum eigenvalue of $D'$ in (5), or respectively, (6). We described how that minimum eigenvalue produces a first-hit cut. We noticed that in practice it may be useful to go to the second minimum eigenvalue and use it to compute a second-hit cut

using exactly the same calculations as for the first-hit cut. We certainly do this only if this second minimum eigenvalue is still negative.

– We normalize certain cuts we eventually send to the LP solver (`cplex`). For each $i \in [1..k]$ the coefficient $i$ of decision variable $y_i$ comes from the term $\mathbf{v}^\top A_i \mathbf{v}$, where $\mathbf{v}$ is the first-hit vector returned by the projection algorithm. When the maximum resulting coefficient in absolute value is greater than 100000, we divide all coefficients of the cut by that maximum coefficient. $\square$

*Remark 5.* The code was implemented in `Matlab` (version `r2018b`) on a mainstream laptop clocked at 1.90GHz with an Intel i7-8665U processor with 4 cores. The number of threads can go up to 8 using a hyper-threading technology. We used the default `Matlab` configuration that allows up to 4 threads (a `maxNumCompThreads` value of 4). We chose `Matlab` because preliminary experiments suggest it provides the fastest matrix eigenvalue routines for $n \geq 1000$. We used a Linux Mint operation system; the Linux kernel version is 4.15.0. The LP solver is `cplex` version 12.10. $\square$