An oracle-based projection and rescaling algorithm for linear semi-infinite feasibility problems and its application to SDP and SOCP

Masakazu Muramatsu[*] Tomonari Kitahara[†] Bruno F. Lourenço[‡]
Takayuki Okuno[§] Takashi Tsuchiya[¶]

September 27, 2018

Abstract

We point out that Chubanov's oracle-based algorithm for linear programming [5] can be applied almost as it is to linear semi-infinite programming (LSIP). In this note, we describe the details and prove the polynomial complexity of the algorithm based on the real computation model proposed by Blum, Shub and Smale (the BSS model) which is more suitable for floating point computation in modern computers. The adoption of the BBS model makes our description and analysis much simpler than the original one by Chubanov [5]. Then we reformulate semidefinite programming (SDP) and second-order cone programming (SOCP) into LSIP, and apply our algorithm to obtain new complexity results for computing interior feasible solutions of homogeneous SDP and SOCP.

Keywords: Linear semi-infinite programming, Projection and rescaling algorithm, Oracle-based algorithm, Semidefinite programming.

1 Introduction

Let T be a (possibly infinite) index set and $\mathcal{A} = \{ \boldsymbol{a}_t \mid t \in T \} \subseteq \mathbb{R}^m$. In this paper, we consider the problem of finding a solution to the following homogeneous linear semi-infinite system:

$$\langle D \rangle \text{ find } \boldsymbol{y} \text{ s.t. } \boldsymbol{a}_t^T \boldsymbol{y} > 0 \ (t \in T).$$

Without loss of generality, we assume that $\boldsymbol{a}_t \neq \boldsymbol{0}$ for every $t \in T$. The objective of this article is to show that the recent oracle-based algorithm proposed by Chubanov [5] can be adapted to solve $\langle D \rangle$ and to work out the details of this extension. The advantage of this more general setting is that we can solve, for instance, certain homogeneous feasibility problems over nonpolyhedral cones. In fact, we will apply our algorithm to semidefinite programming (SDP) and second-order cone programming (SOCP) in Section 5.

The problem $\langle D \rangle$ is a special case of linear semi-infinite programming (LSIP) [13, 9, 17] which minimizes or maximizes a linear objective function with infinitely many linear constraints. When $|T| < \infty$, $\langle D \rangle$ becomes the linear feasibility problem that Chubanov [5] dealt with. We emphasize that the main novelty here is the case where $|T| = \infty$.

[*]Department of Computer and Network Engineering, The University of Electro-Communications 1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585 Japan. (E-mail: MasakazuMuramatsu@uec.ac.jp)

[†]Kyushu University (E-mail: tomonari.kitahara@econ.kyushu-u.ac.jp)

[‡]Department of Mathematical Informatics, Graduate School of Information Science & Technology, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan. (E-mail: lourenco@mist.i.u-tokyo.ac.jp)

[§]Riken Center for Advanced Intelligence Project (E-mail: takayuki.okuno.ks@riken.jp)

[¶]National Graduate Institute for Policy Studies 7-22-1 Roppongi, Minato-ku, Tokyo 106-8677, Japan. (E-mail: tsuchiya@grips.ac.jp)

In this paper, we assume the existence of an oracle for $\langle D\rangle$ having the following input/output:

Oracle($\boldsymbol{y}$)
Input: a nonzero vector $\boldsymbol{y} \in \mathbb{R}^m$
Output: $t \in T$ and $\boldsymbol{a}_t$ such that $\boldsymbol{a}_t^T \boldsymbol{y} \leq 0$, or
declare that $\forall t \in T, \boldsymbol{a}_t^T \boldsymbol{y} > 0$.

We show that Chubanov's oracle-based algorithm [5] can be applied almost as it is to $\langle D\rangle$ together with the oracle above. The algorithm described here either

(i) returns a solution of $\langle D\rangle$,

(ii) returns a certificate that there is no solution for $\langle D\rangle$,

(iii) declare that the maximum volume of parallelogram spanned by vectors in a certain bounded area in the feasible set is less than a positive constant ϵ. (See Section 2 for details.)

Furthermore, our algorithm calls the oracle polynomially many times, hence the running time is polynomial if the oracle is also polynomial.

We use the real computation model proposed by Blum, Shub and Smale [1], while Chubanov [5] uses the standard bit computation model. This means that the meaning of polynomial complexity is different from that used by Chubanov [5]. There are significant differences between the real computation model and the bit computation model. For example, it is known that determining whether an SDP has an optimal solution falls both in P and NP in the real computation model (Ramana [22]), while the status is not known in the bit computation model.

By adopting the real computation model, we can also avoid the discussion on bit size, which makes our analysis much more transparent. As a result, our algorithm can be regarded as a basic scheme to solve convex programming, having a few common properties with the ellipsoid method [14, 19]. Namely, both algorithms work in the variable space and their complexity can be written in terms of an separation oracle. We will discuss on this point in Section 6.

For applications of our method to convex programming, in the latter part of this paper, we consider to solve feasibility problems for SDP and SOCP using the proposed algorithm. We reformulate them into the shape of $\langle D\rangle$, and show that oracles needed are polynomial in the real computation model. Therefore, polynomial algorithms based on projection and rescaling for computing an interior-feasible solution of homogeneous SDP or SOCP are established. We will compare the complexity of the algorithm developed here with some recent projection and rescaling algorithms [20, 18].

From the point of view of semi-infinite programming, a novel point here is that we do not assume any topology on T. This stands in constrast to the usual setting in semi-infinite programming, where it is usually assumed that T is compact and/or $\boldsymbol{a}_t$ is continuous under some topology.

1.1 Previous works on projection and rescaling algorithms

The algorithm described here has its origins in a previous work by Chubanov [4], which proposed a polynomial algorithm to compute an interior feasible solution of homogeneous linear programming problems. Chubanov's original algorithm was extended and improved along several different directions [16, 23, 20, 15, 18, 8, 21, 24, 12]. Nevertheless, a common point among most of those variants is that they are divided in two parts, a *basic procedure* and a *main algorithm*. The basic procedure searches for an interior feasible solution, and when it cannot find one, returns a solution called *cut generating vector (CGV)*. The main algorithm then uses the cut generating vector to rescale the problem so that the rescaled problem can be dealt with by the basic procedure again. The idea is that the rescaling makes the problem easier, so it becomes more likely that the basic procedure will find an interior feasible solution. Typically, the basic procedure will also make use of the projection operator. For example, in many variants, it is necessary to orthogonally project a point onto an appropriate linear subspace. Due to this combination of projection and rescaling steps, these algorithms are sometimes called *projection and rescaling algorithms*.

We will now discuss some of the extensions and enhancements. Roos [23] proposed an improved basic procedure which generates sharper cuts and also produced some preliminary numerical results. Improved basic procedures were also proposed by Zhang and Roos [24] and by Gutman [12]. Li, Roos and Terlaky [16] extended Chubanov's algorithm to the case of linear feasibility problems with a single non-homogeneous equality. Peña and Soheili [20] proposed an algorithm which computes an interior feasible solution of homogeneous symmetric cone system using projection and rescaling. An extension to homogeneous feasibility problems over second order cones was proposed by Kitahara and Tsuchiya in [15]. Then, Lourenço, Kitahara, Muramatsu, and Tsuchiya [18] extended Chubanov's algorithm [4] to symmetric cone programming in a different way than Peña and Soheili's algorithm and gave a different complexity result. A summary of the differences between both approaches can be found in Section 1 of [18]. Recently, Peña and Soheili discussed computational aspects of projection and rescaling algorithms for the linear programming case and they presented an extensive set of numerical experiments [21].

In this paper, we take a closer look at another algorithm proposed by Chubanov [5], which solves $\langle D \rangle$ for the case $|T| < \infty$. The algorithm in [5] is quite similar to the one in [4], but an important distinction is that it makes use of an oracle. The advantage of using an oracle is that the algorithm works only in the space of the variable $\boldsymbol{y}$ and, under appropriate assumptions, has polynomial complexity even if, say, the number of inequalities is exponential in m. This aspect of the algorithm can be quite useful and, in fact, it was used by Fujishige [8] to propose an algorithm for minimization of submodular functions.

In Section 3.2 of Dadush et al.[6], the same problem $\langle D \rangle$ equipped with the same oracle was dealt with, and a variant of projection and rescaling algorithms was proposed. Their algorithm and the algorithm proposed in this paper share the same idea to apply a projection and rescaling algorithm to linear semi-infinite programming. However, as the directions of branch are different, there exist some differences. Below we describe them.

The first difference is the condition measures on which the algorithms rely. The algorithm in [6] is based on Goffin's measure of a full-dimensional cone Σ defined by

$$\rho_\Sigma = \sup \{ \, r \mid B(\boldsymbol{x}, r) \subseteq \Sigma, \|\boldsymbol{x}\| = 1 \, \},$$

where $B(\boldsymbol{x}, r)$ is the open ball whose center is $\boldsymbol{x}$ and radius is r. In contrast, the analysis of our algorithm is based on the maximum volume of parallelograms spanned by vectors contained in a bounded area of the cone. See Section 2 for the exact definition of this condition measure. For example, if Σ is the positive orthant of the m dimensional space, then the maximum volume of parallelograms is 1, while $\rho_\Sigma = m^{-1/2}$. On the other hand, when Σ is the second-order cone in the three dimensional space, the former is $\sqrt{6}/8$ while the latter $1/\sqrt{2}$. In this case, the former is less than the latter. To the extent of the authors' knowledge, any useful connection between the two condition measures is not known.

The two algorithms are different in complexity, too. Table 1 shows them. Here, BP and MA are abbreviations of the basic procedure and the main algorithm, respectively. The algorithm in [6] needs $O(m^2)$ iterations of their basic procedure and $O(m \log \rho_\Sigma^{-1})$ rescaling process, whereas our algorithm needs $O(m^3)$ iterations of the basic procedure and $O(\log \epsilon)$ rescaling process. In case of $\epsilon = \rho_\Sigma^{-1}$, the orders of total arithmetic operations needed become identical.

	Dadush et al. [6]	proposed
bd. for length of CGVs	$O(m^{-1})$	$O(m^{-3/2})$
# of Iterations within BP	$O(m^2)$	$O(m^3)$
# of arith. op. within BP	$O((C_o + m^2)m^2)$	$O((C_o + m^2)m^3)$
# of rescaling	$m \log \rho_\Sigma^{-1}$	$\log \epsilon^{-1}$
Total complexity of arith. op.	$O((C_o + m^2)m^3 \log \rho_\Sigma^{-1})$	$O((C_o + m^2)m^3 \log \epsilon^{-1})$
Space complexity of BP	$O(m^3)$	$O(m^2)$
Space complexity of MA	$O(m^2)$	$O(m \min(m, \log_2 \epsilon^{-1}))$

Table 1: Differences in complexity between [6] and the proposed algorithm

As is shown in the last two rows of Table 1, there exist differences in space complexity, too. In the basic procedure, we keep $\{\, x_t \,|\, t \in T_+ \,\}$ where $|T_+| \le m+1$ and an $(m+1)\times(m+1)$ matrix G, whereas the algorithm in [6] keeps $O(m^2)$ variables and the same number of vectors of size m. Also in the main algorithm, the proposed algorithm has a possibility to reduce the space complexity that is depending on the number of rescaling. See Section 4 for the details.

1.2 Organization of this paper

The paper is organized as follows. In Section 2, we give some preliminary observations on LSIP. Sections 3 and 4 describe the basic procedure and the main algorithm, respectively, giving detailed proofs of their complexities. In Section 5, we reformulate the problem of computing interior feasible solutions of a homogeneous SDP and SOCP into LSIP, and apply our algorithm with suitable oracles, respectively. We establish the total complexities of the algorithm including that of the oracles. We compare the result of SDP with some existing results by Peña and Soheili [20] and Lourenço, et al [18]. Finally, we give concluding remarks in Section 6.

2 Preliminary observations

Associated with $\langle D\rangle$, we consider the following problem:

$$\langle P\rangle \left\{ \begin{array}{l} \text{find a finite number of positive weights } \{\, x_t \in \mathbb{R}_{++} \,|\, t \in T_+ \,\} \\ \text{where } T_+ \subseteq T \text{ and } |T_+| < \infty \text{ such that } \sum_{t\in T_+} \boldsymbol{a}_t x_t = \boldsymbol{0}. \end{array} \right.$$

Theorem 1. *It is impossible that both $\langle P\rangle$ and $\langle D\rangle$ have feasible solutions simultaneously.*

Proof. Let us assume that $\{\, x_t \,|\, t \in T_+ \,\}$ and $\boldsymbol{y} \in \mathbb{R}^m$ are feasible solutions of $\langle P\rangle$ and $\langle D\rangle$, respectively. Then we have a contradiction because

$$0 < \sum_{t\in T_+} x_t(\boldsymbol{a}_t^T \boldsymbol{y}) = (\sum_{t\in T_+} x_t \boldsymbol{a}_t)^T \boldsymbol{y} = 0.$$

□

$\langle P\rangle$ is known as Haar's dual of $\langle D\rangle$, and if T is compact and $\boldsymbol{a}_t$ is continuous as a function of t, then they are in fact alternatives each other; $\langle D\rangle$ is feasible if and only if $\langle P\rangle$ is infeasible (See, e.g., Lemma 1 of [17]). However, in this paper, we do not assume any topology on T; it is possible that both $\langle P\rangle$ and $\langle D\rangle$ be simultaneously infeasible, as seen in the next example.

Example 2. *Let $T = (0, \pi]$ and $\boldsymbol{a}_t = (\cos t, \sin t)^T$. Then both $\langle D\rangle$ and $\langle P\rangle$ are infeasible.*

We consider a bounded version of the feasible region of $\langle D\rangle$:

$$\mathcal{F}_0 = \left\{\, \boldsymbol{y} \in \mathbb{R}^m \;\middle|\; \boldsymbol{a}_t^T \boldsymbol{y} > \boldsymbol{0}\ (t \in T), \|\boldsymbol{y}\| \le 1 \,\right\}.$$

Obviously, $\mathcal{F}_0$ is nonempty if and only if $\langle D\rangle$ is feasible. In this paper, we always work under the following assumption.

Assumption 1. *The set $\mathcal{F}_0$ is full-dimensional, i.e., $\dim \mathcal{F}_0 = m$, or equivalently, there exist linearly independent vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \in \mathcal{F}_0$.*

The reason we work under Assumption 1 is that we will consider the maximum volume of parallelogram spanned by vectors in $\mathcal{F}_0$. Intuitively, if this volume is large, the problem is well-conditioned and a solution of $\langle D\rangle$ can easily be computed through a suitable basic procedure. If, however, the volume is small then it is hard to compute a solution in $\mathcal{F}_0$. In the extreme case where the volume is zero, we have to search a subspace in $\mathbb{R}^m$ containing $\mathcal{F}_0$ in its relative interior, which is harder.

To check the strength of the assumption, we consider the extended feasible region:

$$\mathcal{F}_1 = \left\{ \boldsymbol{y} \in \mathbb{R}^m \;\middle|\; \boldsymbol{a}_t^T \boldsymbol{y} \geq \boldsymbol{0} \ (t \in T), \|\boldsymbol{y}\| \leq 1 \right\}.$$

We have the following lemma.

Lemma 3. *If $\mathcal{F}_1$ is full-dimensional, then $\langle D \rangle$ has feasible solutions.*

Proof. Let $\boldsymbol{y}$ be an interior point of $\mathcal{F}_1$. By definition, there exists $\epsilon > 0$ such that $\boldsymbol{y} + B(\epsilon) \subseteq \mathcal{F}_1$ where $B(\epsilon)$ is the open ball whose radius is ϵ.

Suppose that there is no feasible solution for $\langle D \rangle$. This implies that there exists $t \in T$ such that $\boldsymbol{a}_t^T \boldsymbol{y} = 0$. For such t, let $\boldsymbol{u} = -\epsilon \boldsymbol{a}_t / (2\|\boldsymbol{a}_t\|)$, which is contained in $B(\epsilon)$. Then we have $\boldsymbol{a}_t^T(\boldsymbol{y} + \boldsymbol{u}) = -\epsilon\|\boldsymbol{a}_t\|/2 < 0$, which contradicts the fact that $\boldsymbol{y} + \boldsymbol{u} \in \mathcal{F}_1$. Therefore, $\boldsymbol{y}$ is a feasible solution to $\langle D \rangle$. $\square$

Lemma 3 has the following consequence. In [5], Chubanov mentions that his oracle-based algorithm solves the following problem.

$$\text{find } \boldsymbol{y} \text{ s.t. } A^T \boldsymbol{y} \geq 0, \boldsymbol{y} \neq 0, \tag{1}$$

where $A = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$ is some $m \times n$ matrix. He also assumes that the feasible region $\{ \boldsymbol{y} \,|\, A^T \boldsymbol{y} \geq 0 \}$ is either full-dimensional or has no nonzero solution. From Lemma 3, we conclude that all the nontrivial problems considered in [5] have a solution $\boldsymbol{y}$ that satisfy the inequalities strictly. That is why we consider here the problem $\langle D \rangle$, instead of dealing with a generalized form of (1).

Moving on, in the next example we show that there is a case where $\langle D \rangle$ has feasible solutions but $\mathcal{F}_1$ is not full-dimensional.

Example 4. *Let $T = (0, \pi)$ and $\boldsymbol{a}_t = (\cos t, \sin t)^T$. Then the feasible region of $\langle D \rangle$ is $\{ \lambda(0,1)^T \,|\, \lambda > 0 \}$, which is one-dimensional, and so is $\mathcal{F}_1$.*

Given an invertible matrix $M \in \mathbb{R}^{m \times m}$, we define the following problem scaled by M:

$$\langle D(M) \rangle \ \text{find } \tilde{\boldsymbol{y}} \text{ s.t. } \boldsymbol{a}_t^T M \tilde{\boldsymbol{y}} > 0 \ (t \in T).$$

Notice that $\langle D \rangle = \langle D(I) \rangle$. Also, the relationship:

$$\boldsymbol{y} \text{ is feasible for } \langle D \rangle \Leftrightarrow \tilde{\boldsymbol{y}} = M^{-1}\boldsymbol{y} \text{ is feasible for } \langle D(M) \rangle$$

is obvious.

Now we observe some properties on maximum volume of parallelogram spanned by vectors in a bounded set. These relations are used in the following sections to prove the polynomial convergence of the proposed algorithm.

For a bounded set $\mathcal{F} \subseteq \mathbb{R}^m$, we define:

$$\begin{aligned} \mathcal{Y}(\mathcal{F}) &= \left\{ Y \in \mathbb{R}^{m \times m} \,|\, Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m), \ \boldsymbol{y}_i \in \mathcal{F} \ (i = 1, \ldots, m) \right\} \\ d^*(\mathcal{F}) &= \sup \{ |\det Y| \,|\, Y \in \mathcal{Y}(\mathcal{F}) \}. \end{aligned}$$

For an invertible matrix $M \in \mathbb{R}^{m \times m}$ and a scaling factor $\delta > 0$, we define:

$$\mathcal{F}(M, \delta) = \left\{ \tilde{\boldsymbol{y}} \;\middle|\; \boldsymbol{a}_t^T M \tilde{\boldsymbol{y}} \geq \boldsymbol{0} \ (t \in T), \|\tilde{\boldsymbol{y}}\| \leq \delta \right\}.$$

Some observations follow.

Lemma 5. *1. If $\mathcal{F} \supseteq \mathcal{F}'$, then $d^*(\mathcal{F}) \geq d^*(\mathcal{F}')$.*

2. $d^*(M^{-1}\mathcal{F}) = |\det M^{-1}| d^*(\mathcal{F})$.

3. $d^*(\mathcal{F}(M, \delta)) = \delta^m d^*(\mathcal{F}(M, 1))$.

Proof. The first statement is obvious.

To prove the second statement, choose an arbitrary $\tilde{Y} \in \mathcal{Y}(M^{-1}\mathcal{F})$, i.e.,

$$\tilde{Y} = \left(M^{-1}\boldsymbol{y}_1 \dots M^{-1}\boldsymbol{y}_m\right) \text{ where } \boldsymbol{y}_i \in \mathcal{F}\ (i = 1, \dots, m).$$

Then we can write $\tilde{Y} = M^{-1}Y$ where $Y = (\boldsymbol{y}_1 \dots \boldsymbol{y}_m)$. Therefore, we have

$$\begin{aligned} d^*(M^{-1}\mathcal{F}) &= \sup\left\{ |\det \tilde{Y}| \,\middle|\, \tilde{Y} \in M^{-1}\mathcal{F} \right\} \\ &= \sup\left\{ |\det M^{-1}||\det Y| \,|\, Y \in \mathcal{F} \right\} \\ &= |\det M^{-1}| \sup\left\{ |\det Y| \,|\, Y \in \mathcal{F} \right\} \\ &= |\det M^{-1}| d^*(\mathcal{F}), \end{aligned}$$

which is the second statement.

For the final statement, first observe

$$\delta\tilde{\boldsymbol{y}} \in \mathcal{F}(M, \delta) \Leftrightarrow \tilde{\boldsymbol{y}} \in \mathcal{F}(M, 1),$$

and thus

$$\delta\tilde{Y} \in \mathcal{Y}(\mathcal{F}(M, \delta)) \Leftrightarrow \tilde{Y} \in \mathcal{Y}(\mathcal{F}(M, 1)).$$

From this, it follows that

$$\begin{aligned} d^*(\mathcal{F}(M, \delta)) &= \sup\{ |\det(\delta\tilde{Y})| \,|\, \tilde{Y} \in \mathcal{Y}(\mathcal{F}(M, 1)) \} \\ &= \sup\{ \delta^m |\det \tilde{Y}| \,|\, \tilde{Y} \in \mathcal{Y}(\mathcal{F}(M, 1)) \} \\ &= \delta^m d^*(\mathcal{F}(M, 1)). \end{aligned}$$

□

3 Basic Procedure

The basic procedure receives the scaling matrix M and a positive number μ, and returns either a solution of $\langle D \rangle$ or $\langle P \rangle$, or positive weights $\{ x_t \,|\, t \in T_+ \}$ that satisfy a certain property depending on μ.

Inside the basic procedure, we use the notation

$$\tilde{\boldsymbol{a}}_t = \frac{M^T \boldsymbol{a}_t}{\|M^T \boldsymbol{a}_t\|}$$

for $t \in T$. The complexity to compute $\tilde{\boldsymbol{a}}_t$ from $\boldsymbol{a}_t$ and M is $O(m^2)$ in the real computation model. Note that we shall never compute $\{ \tilde{\boldsymbol{a}}_t \,|\, t \in T \}$. Since $|T| = \infty$, this is computationally intractable. Instead, we compute $\tilde{\boldsymbol{a}}_t$ only when it is needed. More specifically, we keep positive weights $\{ x_t \,|\, t \in T_+ \}$ where $T_+ \subseteq T$ with $|T_+| \le m + 1$. Such a set of positive weights can be implemented by using appropriate data structure such as list. We presume that $x_t = 0$ for $t \notin T_+$.

In each iteration, the basic procedure calls the oracle with $M\tilde{\boldsymbol{z}}$ where $\tilde{\boldsymbol{z}} = \sum_{t \in T_+} x_t \tilde{\boldsymbol{a}}_t$. If the oracle detects feasibility of $M\tilde{\boldsymbol{z}}$, returns it as a solution of $\langle D \rangle$. If the oracle returns a violating index $\hat{t}$, the basic procedure computes the minimum distance point from the origin between $\tilde{\boldsymbol{z}}$ and $\tilde{\boldsymbol{a}}_{\hat{t}}$, which will replace $\tilde{\boldsymbol{z}}$. The index $\hat{t}$ is added to T_+, and if $|T_+| > m+1$, a procedure called *the index eliminating procedure* is invoked to replace the positive weights by a new set $\{ x_t \,|\, t \in T_+ \}$ satisfying $|T_+| \le m + 1$, $\sum_{t \in T_+} x_t \boldsymbol{a}_t = \tilde{\boldsymbol{z}}$, and $\sum_{t \in T_+} x_t = 1$.

Now we describe the basic procedure step by step. In the following description, we assume that the oracle always returns the second case, because otherwise we immediately obtain a solution of $\langle D \rangle$.

Basic Procedure

Input: a positive number $\mu > 0$ and an invertible matrix $M \in \mathbb{R}^{m\times m}$

Output:

1. $\tilde{\boldsymbol{y}}$ such that $\forall t \in T, \boldsymbol{a}_t^T M\tilde{\boldsymbol{y}} > 0$, or
2. $T_+ \subseteq T$ where $|T_+| \leq m+1$, and positive weights $\{\, x_t \,|\, t \in T_+ \,\}$ such that $\sum_{t\in T_+} x_t\tilde{\boldsymbol{a}}_t = \boldsymbol{0}$, or
3. $T_+ \subseteq T$ where $|T_+| \leq m+1$, and positive weights $\{\, x_t \,|\, t \in T_+ \,\}$ such that $\| \sum_{t\in T_+} x_t\tilde{\boldsymbol{a}}_t\| \leq \mu/(m+1)$.

Steps:

// **Initialization**: We suppose that $x_t = 0$ for every $t \in T$.

Step 1. Let $\tilde{\boldsymbol{y}}$ be an arbitrary nonzero vector.

Step 2. Let $\bar{t} \in T$ be the index returned by Oracle($M\tilde{\boldsymbol{y}}$) and compute $\tilde{\boldsymbol{a}}_{\bar{t}}$.

Step 3. Let $T_+ = \{\bar{t}\}$, $x_{\bar{t}} = 1$, and $\tilde{\boldsymbol{z}} = \tilde{\boldsymbol{a}}_{\bar{t}}$.

// **Loop**

Step 4. Let $\hat{t} \in T$ be the index returned by Oracle($M\tilde{\boldsymbol{z}}$) and compute $\tilde{\boldsymbol{a}}_{\hat{t}}$.

Step 5. Let $T'_+ = T_+ \cup \{\hat{t}\}$.

Step 6. For $t \in T'_+$, set

$$x'_t = \begin{cases} \alpha x_t & \text{if } t \neq \hat{t} \\ \alpha x_{\hat{t}} + 1 - \alpha & \text{if } t = \hat{t} \end{cases} \quad \text{where } \alpha = \frac{\tilde{\boldsymbol{a}}_{\hat{t}}^T(\tilde{\boldsymbol{a}}_{\hat{t}} - \tilde{\boldsymbol{z}})}{\|\tilde{\boldsymbol{a}}_{\hat{t}} - \tilde{\boldsymbol{z}}\|^2}. \tag{2}$$

Step 7. Let $\tilde{\boldsymbol{z}} = \sum_{t\in T'_+} x'_t\tilde{\boldsymbol{a}}_t$.

Step 8. Call *index elimination procedure* to compute T_+ and $\{\, x_t \,|\, t \in T_+ \,\}$ such that $|T_+| \leq m$, $x_t \geq 0 (t \in T_+)$, $\sum_{t\in T_+} x_t = 1$, and $\sum_{t\in T_+} x_t\tilde{\boldsymbol{a}}_t = \tilde{\boldsymbol{z}}$.

Step 9. If $\tilde{\boldsymbol{z}} = \boldsymbol{0}$, then return $\{\, x_t \,|\, t \in T_+ \,\}$ with the second status.

Step 10. If $\|\tilde{\boldsymbol{z}}\| \leq \mu/(m+1)$, then return the third status with $\{\, x_t \,|\, t \in T_+ \,\}$.

Step 11. Go to Step 4.

The index elimination procedure works as follows.

1. During the first m iterations, it does nothing. (Just set $T_+ = T'_+$ and $x_t = x'_t$ for every $t \in T_+$.)
2. At the $(m+1)$-th iteration, it computes[1]

$$G = \begin{pmatrix} \tilde{\boldsymbol{a}}_{t_1} & \dots & \tilde{\boldsymbol{a}}_{t_{m+1}} \\ 1 & \dots & 1 \end{pmatrix}^{-1}, \tag{3}$$

where $T_+ = T'_+ = \{t_1, \dots, t_{m+1}\}$, and store it.

[1]If this matrix is not invertible, then just skip this procedure until we have $m+1$ independent vectors.

3. If $|T'_+| = m+2$ with $\hat{t}$ being the last index added, then compute:

$$\boldsymbol{\gamma} = G \begin{pmatrix} \boldsymbol{a}_{\hat{t}} \\ 0 \end{pmatrix}.$$

   Let
$$t^* = \operatorname{argmax}\{\, \beta \mid x'_t - \beta\gamma_t \geq 0 \,\} = \operatorname{argmax}\{\, -x'_t/\gamma_t \mid \gamma_t < 0 \,\}$$
   and $\beta^* = -x'_{t^*}/\gamma_{t^*}$. If $\beta^* \geq x'_{\hat{t}}$, then set

$$x_t = x'_t + x'_{\hat{t}}\gamma_t \ (t \in T_+).$$

   Else, set

$$x_t = x'_t - \frac{x'_{t^*}}{\gamma_{t^*}}\gamma_t \ (t \in T'_+\backslash\{t^*, \hat{t}\}),\ x_{\hat{t}} = x'_t + \frac{x'_{t^*}}{\gamma_{t^*}},$$

   and update

$$G \leftarrow G - \frac{G\boldsymbol{e}_{t^*}\bar{\boldsymbol{a}}^T G}{1 + \boldsymbol{e}_{t^*}^T G\bar{\boldsymbol{a}}},\ T_+ \leftarrow T'_+\backslash\{t^*\},$$

   where $\bar{\boldsymbol{a}} = \begin{pmatrix} \boldsymbol{a}_{\hat{t}} - \boldsymbol{a}_{t^*} \\ 0 \end{pmatrix}$ and $\boldsymbol{e}_{t^*}$ is the vector having 1 at the position corresponding to t^* and zero otherwise.

Using the index elimination procedure, we can bound the size of T_+ by $m+1$.

Lemma 6. *The index elimination procedure returns T_+ and $\{\, x_t \,|\, t \in T_+ \,\}$ such that*

$$\begin{aligned} x_t &\geq 0 \ (t \in T_+) &(4)\\ \sum_{t\in T_+} x_t\tilde{\boldsymbol{a}}_t &= \tilde{\boldsymbol{z}} &(5)\\ \sum_{t\in T_+} x_t &= 1 &(6)\\ |T_+| &= m+1. &(7)\end{aligned}$$

Furthermore, (3) holds with a suitable arrangement of vectors $\{\, \tilde{\boldsymbol{a}}_t \,|\, t \in T_+ \,\}$ when the index elimination procedure ends.

Proof. In the procedure, there are two cases, (i) $\beta^* \geq x'_{\hat{t}}$, and (ii) $\beta^* < x'_{\hat{t}}$, and we divide the proof accordingly.
Case (i). When $\beta^* \geq x'_{\hat{t}}$, since T_+ does not change, (3) and (7) hold automatically. Since

$$\tilde{\boldsymbol{z}} = \sum_{t\in T'_+} x'_t\tilde{\boldsymbol{a}}_t = \sum_{t\in T_+\backslash\{\hat{t}\}} \left(x'_t + x'_{\hat{t}}\gamma_t\right)\tilde{\boldsymbol{a}}_t = \sum_{t\in T_+\backslash\{\hat{t}\}} x_t\tilde{\boldsymbol{a}}_t,$$

(5) also holds. The equality $\sum_{t\in T_+\backslash\{\hat{t}\}} \gamma_t = 0$ implies (6).
Case (ii). When $\beta^* < x'_{\hat{t}}$, since we replace t^* with $\hat{t}$, it is obvious that (7) holds.
The equation (3) implies

$$\sum_{t\in T'_+\backslash\{\hat{t}\}} \gamma_t\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_{\hat{t}},$$

from which it follows that

$$\boldsymbol{a}_{t^*} = -\sum_{t\in T_+\backslash\{\hat{t}\}} \gamma_t/\gamma_{t^*}\tilde{\boldsymbol{a}}_t + 1/\gamma_{t^*}\tilde{\boldsymbol{a}}_{\hat{t}}.$$

Now we have

$$\begin{aligned}\sum_{t\in T_+} x_t\tilde{\boldsymbol{a}}_t &= \sum_{t\in T_+\setminus\{\hat{t}\}}\left(x'_t-\frac{x'_{t^*}}{\gamma_{t^*}}\gamma_t\right)\tilde{\boldsymbol{a}}_t+\left(x'_{\hat{t}}+\frac{x'_{t^*}}{\gamma_{t^*}}\right)\tilde{\boldsymbol{a}}_{\hat{t}}\\ &= \sum_{t\in T'_+} x'_t\tilde{\boldsymbol{a}}_t=\tilde{\boldsymbol{z}}.\end{aligned}$$

Similarly, (6) easily follows.

Due to the definition of t^*, $x_t \geq 0$ for every $t \in T_+\setminus\{\hat{t}\}$, and for $\hat{t}$, $x_{\hat{t}} > 0$ since $\beta^* < x'_{\hat{t}}$.

Finally, the Sherman-Morrison-Woodbury formula can be applied to the relation

$$\begin{pmatrix}\tilde{\boldsymbol{a}}_{t_1} & \ldots & \tilde{\boldsymbol{a}}_{\hat{t}} & \ldots & \tilde{\boldsymbol{a}}_{t_{m+1}}\\ 1 & \ldots & 1 & \ldots & 1\end{pmatrix}=\begin{pmatrix}\tilde{\boldsymbol{a}}_{t_1} & \ldots & \tilde{\boldsymbol{a}}_{t^*} & \ldots & \tilde{\boldsymbol{a}}_{t_{m+1}}\\ 1 & \ldots & 1 & \ldots & 1\end{pmatrix}+\bar{\boldsymbol{a}}(\boldsymbol{e}_{t^*})^T,$$

which produces

$$\begin{pmatrix}\tilde{\boldsymbol{a}}_{t_1} & \ldots & \tilde{\boldsymbol{a}}_{\hat{t}} & \ldots & \tilde{\boldsymbol{a}}_{t_{m+1}}\\ 1 & \ldots & 1 & \ldots & 1\end{pmatrix}^{-1}=\frac{G\boldsymbol{e}_{t^*}\bar{\boldsymbol{a}}^TG}{1+\boldsymbol{e}_{t^*}^TG\bar{\boldsymbol{a}}}.$$

This implies that the new G is the inverse of the left-hand side of the above, which proves (3). □

The choice of α in Step 6 ensures the following property of the procedure.

Lemma 7. *Suppose that $\sum_{t\in T_+} x_t = 1$ at the beginning of Step 6. Then it hold that*

$$\sum_{t\in T'_+} x'_t = 1 \tag{8}$$

and

$$\frac{1}{\|\sum_{t\in T'_+}\tilde{\boldsymbol{a}}_t x'_t\|^2}\geq\frac{1}{\|\sum_{t\in T_+}\tilde{\boldsymbol{a}}_t x_t\|^2}+1 \tag{9}$$

at the end of Step 6.

Lemma 7 can be proved by the standard arguments. For completeness, we give one in Appendix A.

Theorem 8. *The basic procedure stops in at most $(m+1)^2/\mu^2$ iterations. If the complexity of Oracle is C_o, then the total complexity of Basic procedure is $O((m^2+C_o)m^2/\mu^2)$.*

Proof. The algorithm stops at Step 10 when

$$\frac{1}{\|\sum_{t\in T_+}\tilde{\boldsymbol{a}}_t\boldsymbol{x}'_t\|^2}\geq\frac{(m+1)^2}{\mu^2}. \tag{10}$$

Since, starting from a positive value, the left-hand side is increased at least by 1 at each iterations, (10) is met at most $(m+1)^2/\mu^2$ iterations.

Now we check the complexity of one iteration of the basic procedure.

Step 4 is $O(C_o+m^2)$ where C_o is the complexity of the oracle, since we have to compute $M\boldsymbol{a}_{\hat{t}}$ for $\tilde{\boldsymbol{a}}_{\hat{t}}$. Step 6 is $O(m)$. Step 7 needs $O(m)$, since the new $\tilde{\boldsymbol{z}}$ can be computed by $\alpha\tilde{\boldsymbol{z}}+(1-\alpha)\tilde{\boldsymbol{a}}_{\hat{t}}$.

In Step 8, we call the index elimination procedure. At the $(m+1)$-th calls of the index elimination procedure, we need to compute the inverse of an $(m+1)\times(m+1)$ matrix, which costs $O(m^3)$, but this is only once. Each call after that includes only matrix-vector computations, and needs $O(m^2)$.

Summing up them all, the total complexity is $O((m^2+C_o)m^2/\mu^2)$. □

Let $\boldsymbol{x}'$ be the vector returned by the basic procedure at Step 10. Since $\sum_{t \in T_+} x'_t = 1$, $x'_t \geq 0$ $(t \in T_+)$ and $|T_+| \leq m+1$, we have

$$\frac{1}{m+1} \leq \max_{t \in T_+} x'_t \leq 1.$$

Lemma 9. *Let* $\tilde{t} = \operatorname{argmax}\{\, x'_t \mid t \in T_+ \,\}$. *Then for arbitrary* $\boldsymbol{y}$ *such that* $\tilde{\boldsymbol{a}}_t^T \boldsymbol{y} > 0$ $(t \in T)$, *it holds that*

$$0 < \tilde{\boldsymbol{a}}_{\tilde{t}}^T \boldsymbol{y} \leq \mu \|\boldsymbol{y}\|.$$

Proof. We have

$$0 \leq \frac{1}{m+1} \tilde{\boldsymbol{a}}_{\tilde{t}}^T \boldsymbol{y} \leq x'_{\tilde{t}} \tilde{\boldsymbol{a}}_{\tilde{t}}^T \boldsymbol{y} \leq \sum_{t \in T_+} x'_t \tilde{\boldsymbol{a}}_t^T \boldsymbol{y} \leq \| \sum_{t \in T_+}^{n} x'_t \tilde{\boldsymbol{a}}_t \| \|\boldsymbol{y}\| \leq \frac{\mu}{m+1} \|\boldsymbol{y}\|.$$

□

4 The main algorithm

The main algorithm receives an instance of $\langle D \rangle$ and a positive number ϵ. The main algorithm calls the basic procedure with a suitable scaling matrix, and if it returns a solution of $\langle D \rangle$ or $\langle P \rangle$, then it finishes with the solution. Otherwise, it uses positive weights that is returned by the basic procedure to make a scaling matrix, and call the basic procedure again. The number of calls of the basic procedure from the main algorithm is bounded by

$$s^*_\epsilon = \left(\frac{1}{2} \log_2 e \right) \log_2 \epsilon^{-1},$$

as we will see in the following.

Below we describe the main algorithm.

Main Algorithm

Input: an instance of $\langle D \rangle$ and a positive number ϵ

Output:

1. $\boldsymbol{y}$ such that $\forall t \in T, \boldsymbol{a}_t^T \boldsymbol{y} > 0$, or
2. $T_+ \subseteq T$ where $|T_+| \leq m+1$, and positive weights $\{\, x_t \mid t \in T_+ \,\}$ such that $\sum_{t \in T_+} x_t \boldsymbol{a}_t = \boldsymbol{0}$, or
3. declare that $d^*(\mathcal{F}(I,1)) \leq \epsilon$.

Steps:

```
Step 1. Let $s = 1$ and $M_s = I$.

Step 2. If $s \geq s^*_\epsilon$, then stop. $d^*(\mathcal{F}(I,1)) \leq \epsilon$.

Step 3. Call Basic Procedure with $M_s$ and $\mu = \sqrt{3m}^{-1}$.
— If a solution of $\langle D \rangle$ or $\langle P \rangle$ is returned, then return it.
— Otherwise, let $\{\, x_t \mid t \in T_+ \,\}$ be the returned positive weights.

Step 4. Let $\tilde{t} = \operatorname{argmax}\{\, x_t \mid t \in T_+ \,\}$.

Step 5. Let $\tilde{\boldsymbol{a}}_{\tilde{t}} = M_s^T \boldsymbol{a}_{\tilde{t}} / \|M_s^T \boldsymbol{a}_{\tilde{t}}\|$ and
        $$D_{\tilde{t}} = I - \frac{1}{2} \tilde{\boldsymbol{a}}_{\tilde{t}} \tilde{\boldsymbol{a}}_{\tilde{t}}^T. \qquad (11)$$

Step 6. Let $M_{s+1} = M_s D_{\tilde{t}}$.
```

Step 7. Let $s = s + 1$ and go to Step 2.

Now we analyze the complexity of the main algorithm. The proof is basically the same as the argument presented before Theorem 2.1 in Chubanov [5].

Lemma 10. *Suppose that the basic procedure called with M returned $\{ x_t \mid t \in T_+ \}$. Let $\tilde{t} = \operatorname{argmax}\{ x'_t \mid t \in T_+ \}$ and define $D_{\tilde{t}}$ by (11). For every $\tilde{\boldsymbol{y}} \in \mathcal{F}(M, 1)$, it holds that*

$$\|D_{\tilde{t}}^{-1}\tilde{\boldsymbol{y}}\| \leq \|\tilde{\boldsymbol{y}}\|\sqrt{1 + \frac{1}{m}}.$$

Proof. Given $\tilde{\boldsymbol{y}} \in \mathcal{F}(M, 1)$, we first decompose:

$$\tilde{\boldsymbol{y}} = \lambda\tilde{\boldsymbol{a}}_{\tilde{t}} + \boldsymbol{u}$$

uniquely where $\boldsymbol{u} \in \ker \tilde{\boldsymbol{a}}_{\tilde{t}}^T$ and $\lambda \in \mathbb{R}$. Due to Lemma 9 with $\mu = \sqrt{3m}^{-1}$, we have

$$|\lambda|\|\tilde{\boldsymbol{a}}_{\tilde{t}}\| = |\tilde{\boldsymbol{a}}_{\tilde{t}}^T\tilde{\boldsymbol{y}}| \leq \frac{\|\tilde{\boldsymbol{y}}\|}{\sqrt{3m}}.$$

Since

$$D_{\tilde{t}}^{-1} = I + \tilde{\boldsymbol{a}}_{\tilde{t}}\tilde{\boldsymbol{a}}_{\tilde{t}}^T,$$

we have

$$\begin{aligned}
\|D_{\tilde{t}}^{-1}\tilde{\boldsymbol{y}}\|^2 &= \left\|\left(I + \boldsymbol{a}_{\tilde{t}}\boldsymbol{a}_{\tilde{t}}^T\right)\boldsymbol{y}\right\|^2 \\
&= \|2\lambda\tilde{\boldsymbol{a}}_{\tilde{t}} + \boldsymbol{u}\|^2 = 4\lambda^2\|\tilde{\boldsymbol{a}}_{\tilde{t}}\|^2 + \|\boldsymbol{u}\|^2 \\
&= 3\lambda^2\|\tilde{\boldsymbol{a}}_{\tilde{t}}\|^2 + \|\tilde{\boldsymbol{y}}\|^2 \\
&\leq \|\tilde{\boldsymbol{y}}\|^2\left(\frac{1}{m} + 1\right).
\end{aligned}$$

□

Lemma 11. *Suppose that the basic procedure called with M returned $\{ x_t \mid t \in T_+ \}$. Let $\tilde{t} = \operatorname{argmax}\{ x'_t \mid t \in T_+ \}$ and define $D_{\tilde{t}}$ by (11). Then it holds that*

$$d^*(\mathcal{F}(M, 1)) \leq \frac{\sqrt{e}}{2}d^*(\mathcal{F}(MD_{\tilde{t}}, 1)).$$

Proof. For arbitrary $\boldsymbol{u} \in \mathcal{F}(M, 1)$, it follows from Lemma 10 that

$$\|D_{\tilde{t}}^{-1}\boldsymbol{u}\| \leq \sqrt{1 + \frac{1}{m}}\|\boldsymbol{u}\| \leq \sqrt{1 + \frac{1}{m}}.$$

This and the fact that

$$\boldsymbol{a}_t^T MD_{\tilde{t}}D_{\tilde{t}}^{-1}\boldsymbol{u} > 0 \ (t \in T)$$

lead $D_{\tilde{t}}^{-1}\boldsymbol{u} \in \mathcal{F}(MD_{\tilde{t}}, \sqrt{1 + m^{-1}})$, which implies $\mathcal{F}(MD_{\tilde{t}}, \sqrt{1 + m^{-1}}) \supseteq D_{\tilde{t}}^{-1}\mathcal{F}(M, 1)$. Now the first statement of Lemma 5 implies

$$d^*(\mathcal{F}(MD_{\tilde{t}}, \sqrt{1 + m^{-1}})) \geq d^*(D_{\tilde{t}}^{-1}\mathcal{F}(M, 1)), \tag{12}$$

and the third statement of Lemma 5 leads

$$d^*(\mathcal{F}(MD_{\tilde{t}}, \sqrt{1 + m^{-1}})) = \left(1 + \frac{1}{m}\right)^{m/2} d^*(\mathcal{F}(MD_{\tilde{t}}, 1)) < \sqrt{e}d^*(\mathcal{F}(MD_{\tilde{t}}, 1)).$$

On the other hand, using the second statement of Lemma 5 and the fact $\det D_{\tilde{t}}^{-1} = 2$, we can rewrite the right-hand side of (12) as:

$$d^*(D_{\tilde{t}}^{-1}\mathcal{F}(M,1)) = |\det D_{\tilde{t}}^{-1}| d^*(\mathcal{F}(M,1)) = 2d^*(\mathcal{F}(M,1)).$$

Substituting these relations into (12), we obtain

$$\sqrt{e} d^*(\mathcal{F}(MD_{\tilde{t}},1)) \geq 2d^*(\mathcal{F}(M,1)),$$

which is the desired relation. □

Theorem 12. *If the main algorithm stops in s^*_ϵ calls of the basic procedure without finding any feasible solution of $\langle D\rangle$ or $\langle P\rangle$, then $d^*(\mathcal{F}(I,1)) \leq \epsilon$.*

Proof. We assume that the s-th call of the basic procedure where $1 \leq s \leq s^*_\epsilon$ returns $\{\, x^s_t \mid t \in T^s_+ \,\}$, and let $\tilde{t}_s = \operatorname{argmax}\{\, x^s_t \mid t \in T^s_+ \,\}$. Define $D_{\tilde{t}_s}$ by (11). Then Lemma 11 implies that

$$\begin{array}{rcl} d^*(\mathcal{F}(I,1)) & \leq & \dfrac{\sqrt{e}}{2} d^*(\mathcal{F}(D_{\tilde{t}_1},1)) \\ & \vdots & \\ & \leq & \left(\dfrac{\sqrt{e}}{2}\right)^{s^*_\epsilon} d^*(\mathcal{F}(D_{\tilde{t}_1}\cdots D_{\tilde{t}_{s^*_\epsilon}},1)) \\ & \leq & \left(\dfrac{\sqrt{e}}{2}\right)^{s^*_\epsilon} \end{array}$$

where the last inequality is due to the fact that $d^*(\mathcal{F}(D_{\tilde{t}_1}\cdots D_{\tilde{t}_{s^*_\epsilon}},1)) \leq 1$, because the length of each edge of the parallelogram is bounded by 1.

Now it is easy to see that, by the definition of t^*, it holds that

$$\left(\frac{\sqrt{e}}{2}\right)^{s^*_\epsilon} \leq \epsilon.$$

□

Since we call the basic procedure with $\mu = \sqrt{3m}^{-1}$, we have the following complexity result for the main algorithm.

Corollary 13. *The main algorithm returns a feasible solution of $\langle D\rangle$ or $\langle P\rangle$, or declare that $d^*(\mathcal{F}(I,1)) \leq \epsilon$ in $O((m^2 + C_o)m^3 \log \epsilon^{-1})$ arithmetic operations if C_o is the complexity of the oracle.*

Finally, we discuss the space complexity needed by the main algorithm. If $m > \log_2 \epsilon^{-1}$, it is a clever choice to store $\{\tilde{\boldsymbol{a}}_{t_1}, \ldots, \tilde{\boldsymbol{a}}_{t_s}\}$ instead of M_s, because we can easily compute

$$M_s \boldsymbol{z} = D_{t_1} \cdots D_{t_s} \boldsymbol{z}$$

by using $\{\tilde{\boldsymbol{a}}_{t_1}, \ldots, \tilde{\boldsymbol{a}}_{t_s}\}$ in $O(ms)$ where $s \leq \log_2 \epsilon^{-1} < m$. Therefore, in this case, we can reduce the space complexity to $O(m \log_2 \epsilon^{-1})$ without sacrificing the complexity of arithmetic operations.

5 Applications to SDP and SOCP

In this section, we use our projection and rescaling algorithm for solving SDP and SOCP.

5.1 SDP

Let us consider the SDP feasibility problem:

$$\langle LMI \rangle \text{ find } \boldsymbol{y} \text{ s.t. } \sum_{i=1}^{m} y_i A_i \succ O,$$

where $A_i (i = 1, \ldots, m)$ are symmetric matrices and $A \succ O$ means that A is positive definite. This problem is particular case of what is called a *linear matrix inequality (LMI)*, which appears in the theory of H_∞ control [2].

The problem $\langle LMI \rangle$ can be cast into an LSIP problem as follows:

$$\langle LMI/LSIP \rangle \text{ find } \boldsymbol{y} \text{ s.t. } \sum_{i=1}^{m} y_i A_i \bullet (\boldsymbol{v}\boldsymbol{v}^T) > 0 \ (\boldsymbol{v} \in \bar{B}),$$

where $\bar{B} = \{ \boldsymbol{v} \mid \|\boldsymbol{v}\| = 1 \}$ is the surface of the unit ball whose center is at the origin. Using the notation $(\boldsymbol{a_v})_i = A_i \bullet (\boldsymbol{v}\boldsymbol{v}^T) = \boldsymbol{v}^T A_i \boldsymbol{v}$ for $\boldsymbol{v} \in \bar{B}$, $\langle LMI/LSIP \rangle$ can be expressed as

$$\text{find } \boldsymbol{y} \text{ s.t. } \boldsymbol{a}_{\boldsymbol{v}}^T \boldsymbol{y} > 0 \ (\boldsymbol{v} \in \bar{B}).$$

Given $\boldsymbol{v}$, computing $\boldsymbol{a}_v$ is $O(mn^2)$ if each A_i is dense.

Now we apply the algorithm proposed in this paper to $\langle LMI/LSIP \rangle$. The oracle for $\langle LMI/LSIP \rangle$ should have the following input and output.

SDPOracle:
Input: $n \times n$ symmetric matrix X
Output: $\boldsymbol{v} \in \bar{B}$ such that $\boldsymbol{v}^T X \boldsymbol{v} \leq 0$, or declare X is positive definite.

The oracle can be implemented by, say, a modification of Cholesky factorization. Let us consider to apply Cholesky factorization to X. If we can factorize to the end, then we obtain $X = LL^T$ with full-ranked L and we know that X is positive definite. Otherwise, we cannot continue at some point of Cholesky factorization by getting nonpositive pivot at a diagonal, and we realize that X is not positive definite. Then, a certificate can be constructed by using the row where the nonpositive pivot appeared. See also [7]. Overall, we can implement the oracle for $\langle LMI/LSIP \rangle$ so that it runs in $O(n^3)$ where n is the size of the matrix X.

Note that as soon as the oracle returns $\boldsymbol{v}$, we have to compute $\boldsymbol{a_v}$, which is $O(mn^2)$. After all, the complexity of the oracle including the computation of $\boldsymbol{a_v}$ is $O((m + n)n^2)$.

We state the complexity of the proposed algorithm applied for SDP.

Theorem 14. *Suppose that we reformulate $\langle LMI \rangle$ into $\langle LMI/LSIP \rangle$ and solve it by the main algorithm using the oracle above. Then the algorithm with SDPOracle computes a feasible solution of $\langle LMI/LSIP \rangle$ or declare that $\mathcal{F}(I, 1) \leq \epsilon$ in*

$$O\left((m^2 + mn^2 + n^3)m^3 \log \epsilon^{-1}\right).$$

Proof. By Corollary 13, since the size of $\boldsymbol{y}$ is m and $C_o = O((m+n)n^2)$, we immediately obtain the result. □

In Table 2, we summarize the complexity of solving SDP by three projection and rescaling algorithms: Peña and Soheili [20], Lourenço, Kitahara, Muramatsu and Tsuchiya (LKMT) [18], and the proposed one.

Peña & Soheili [20]	LKMT [18]	Proposed
$O(mn^6 \log \delta^{-1})$	$O((m^3 + m^2n^2 + n^4m)n \log \tilde{\epsilon}^{-1})$	$O((m^2 + mn^2 + n^3)m^3 \log \epsilon^{-1})$

Table 2: Complexity Comparison in case of SDP

The first entry in Table 2 corresponds to the projection and rescaling algorithm by Peña and Soheili [20], when the von Neumann scheme is used as a basic procedure. For the von Neumann scheme, see Section 5 in

[20] for more details. Basically, the basic procedures used in [18] and the proposed algorithm are categorized to this scheme. The second entry in Table 2 corresponds to the bound for the algorithm developed in [18], see the remarks after Theorem 17 in [18]. Here, we specialize the complexity bound obtained in [18] to the case of a single positive semidefinite cone.

Our algorithm looks less dependent on n, compared to the others. However, there are a few caveats when trying to compare the complexity results described in Table 2. In all three algorithms different measures are used in the complexity. In Peña and Soheili [20], δ is the condition measure of the linear subspace and the semidefinite cone, and in [18], $\tilde{\epsilon}$ is an upper bound of minimum eigenvalue of feasible solution contained in a certain half space. In the proposed method, ϵ is the maximum volume of parallelogram spanned by vectors in the bounded feasible region $\mathcal{F}_0$. Although we do not know the precise relationships between those measures, it is hard to imagine that there are significant order differences between them.

The ability to exploit sparsity in X is an advantage of our algorithm to the others. Since our algorithm pushes the numerical part concerning X into an oracle, if we know in advance that X has a sparse structure so that its positive semidefiniteness can be computed efficiently, we can immediately take advantage of it. In contrast, to the best of the authors' knowledge, how to exploit sparsity in X has not known for the algorithms in [20] and [18], since they use projections in the space of X.

5.2 SOCP

We denote the k dimensional second-order cone by $\mathcal{K}_k$, i.e.,

$$\mathcal{K}_k = \left\{ \begin{pmatrix} x_0 \\ \tilde{\boldsymbol{x}} \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{k-1} \mid x_0 \geq \|\tilde{\boldsymbol{x}}\| \right\}.$$

If we denote

$$\mathcal{L}_k = \{ \boldsymbol{x} \in \mathbb{R}^k \mid x_0 = 1 \},$$

then we can easily see the following equivalence:

$$\boldsymbol{x} \in \mathrm{Int}(\mathcal{K}_k) \Leftrightarrow \forall \boldsymbol{a} \in \mathcal{K}_k \cap \mathcal{L}_k,\ \boldsymbol{a}^T\boldsymbol{x} > 0.$$

Furthermore, if $\boldsymbol{x} = (x_0, \tilde{\boldsymbol{x}}) \notin \mathrm{Int}(\mathcal{K}_k)$ and $\tilde{\boldsymbol{x}} \neq \boldsymbol{0}$, then for $\boldsymbol{v} = (1, -\tilde{\boldsymbol{x}}/\|\tilde{\boldsymbol{x}}\|) \in \mathcal{K}_k \cap \mathcal{L}_k$, we have

$$\boldsymbol{v}^T\boldsymbol{x} = x_0 - \|\tilde{\boldsymbol{x}}\| \leq 0. \tag{13}$$

In other words, we can check whether $\boldsymbol{x}$ is in the interior of $\mathcal{K}_n$ or not, and if not, can find $\boldsymbol{v} \in \mathcal{K}_n \cap \mathcal{L}_n$ such that $\boldsymbol{v}^T\boldsymbol{x} \leq 0$ in $O(n)$.

We consider the following feasibility problem of homogeneous SOCP:

$$\langle SOCP \rangle \text{ find } \boldsymbol{y} \text{ s.t. } A^T\boldsymbol{y} \in \mathcal{K}$$

where $A \in \mathbb{R}^{m \times n}$, $\mathcal{K} = \mathcal{K}_{n_1} \times \cdots \times \mathcal{K}_{n_p}$, and $n = \sum_{i=1}^p n_i$. According to the structure of $\mathcal{K}$, we set $\mathcal{L} = \mathcal{L}_{n_1} \times \cdots \times \mathcal{L}_{n_p}$.

The aim of this subsection is to solve $\langle SOCP \rangle$ by using the proposed algorithm for LSIP. To this end, we reformulate $\langle SOCP \rangle$ into

$$\langle SOCP/LSIP \rangle \text{ find } \boldsymbol{y} \text{ s.t. } \sum_{i=1}^m y_i \boldsymbol{a}_i^T \boldsymbol{v} > 0 \ (\boldsymbol{v} \in \mathcal{K} \cap \mathcal{L}),$$

and what we request for inputs and outputs of the oracle are as follows.
SOCPOracle:
Input: $\boldsymbol{x} \in \mathbb{R}^n$
Output: $\boldsymbol{v} \in \mathcal{K} \cap \mathcal{L}$ such that $\boldsymbol{x}^T\boldsymbol{v} \leq 0$, or declare $\boldsymbol{x} \in \mathrm{Int}(\mathcal{K})$.

Due to the statement just after (13), it is clear that this oracle will run in $O(n)$, where $n = \sum_{i=1}^p n_i$. Now we obtain the complexity of computing a solution of $\langle SOCP/LSIP \rangle$.

Theorem 15. *Suppose that we reformulate $\langle SOCP \rangle$ into $\langle SOCP/LSIP \rangle$ and solve it by the main algorithm using the oracle above. Then the algorithm computes a feasible solution of $\langle SOCP/LSIP \rangle$ or declare that $\mathcal{F}(I, 1) \leq \epsilon$ in*

$$O\left((m^2 + n)m^3 \log \epsilon^{-1}\right).$$

Proof. Plug in the complexity of the SOCPOracle into C_o in Corollary 13, and we immediately obtain the result. □

6 Concluding Remarks

We have extended Chubanov's recent work [5] on a projection and rescaling algorithm to the framework of LSIP. Chubanov's idea was applied almost word by word with significant simplification. Then we applied the proposed algorithm to SDP and SOCP, and showed a polynomial complexity of the algorithm.

We now discuss briefly the relation between our approach and the ellipsoid method . Both the proposed oracle-based projection and rescaling algorithm and the ellipsoid method [14, 19] work in the variable space using an separation oracle; they can be viewed as general scheme for linear and nonlinear problems. One may also consider applying the ellipsoid algorithm to LSIPs with the aid of an oracle. Indeed, this type of extension of the ellipsoid method was developed for SDP in the seminal work by Grötschel, Lovász and Schrijver on combinatorial optimization [10, 11]. According to the textbook by Bubeck [3], the complexity to solve an SDP by using the ellipsoid method using SDPOracle is $O(\max(m, n)n^6 \log \hat{\epsilon}^{-1})$, where $\hat{\epsilon} > 0$ is another measure for optimality. Note that this complexity is to solve an optimization problem, not for the homogeneous feasibility problem, but from the complexity point of view, they are the same.

It is well-known that the ellipsoid method is not practical at all, because the algorithm proceeds just as the theory predicts, and cannot take any advantage of the easiness of real-world problems.

Since our algorithm uses only projection and rescaling, its implementation is by far easier than that of the ellipsoid method. However, we do not know whether the algorithm is efficient enough to solve real-world problems. To check the practicality of the proposed algorithm through extensive numerical experiments is an important next step.

Acknowledgements

M. Muramatsu is supported in part by Grant-in-aid for Scientific Research(C) 17K00031. T. Tsuchiya, B. Lourenço, and T. Kitahara are supported in part by Grant-in-aid for Scientific Research (B) 18H03206. T. Okuno is supported in part by Grant-in-aid for Young Scientists (B) 15K15943. T. Kitahara is supported in part by Grant-in-aid for Young Scientists (B) 15K15941.

Appendix A: Proof of Lemma 7

The equality (8) can easily be seen by

$$\sum_{t \in T_+} x'_t = \sum_{t \in T_+, t \neq \hat{t}} x'_t + x'_{\hat{t}} = \alpha \sum_{t \in T_+, t \neq \hat{t}} x_t + \alpha x_{\hat{t}} + 1 - \alpha = \alpha \sum_{t \in T_+} x_t + 1 - \alpha = 1.$$

For (9), we will prove

$$\frac{1}{\| \sum_{t \in T_+} \tilde{\boldsymbol{a}}_t x'_t \|^2} \geq \frac{1}{\| \sum_{t \in T_+} \tilde{\boldsymbol{a}}_t x_t \|^2} + \frac{1}{\| \tilde{\boldsymbol{a}}_{\hat{t}} \|^2}. \tag{14}$$

Then (9) readily follows since $\|\tilde{\boldsymbol{a}}_{\hat{t}}\| = 1$. The inequality (14) is obvious from the following lemma.

Lemma 16. *Let $\boldsymbol{a}, \boldsymbol{z} \in \mathbb{R}^m$ such that $\boldsymbol{a}^T\boldsymbol{z} \leq 0$, and consider to find the minimum length point between them;*

$$\begin{cases} \text{minimize} & f(\alpha) = \|\alpha\boldsymbol{z} + (1-\alpha)\boldsymbol{a}\|^2 \\ \text{subject to} & 0 \leq \alpha \leq 1. \end{cases}$$

The solution α^* of the above problem is

$$\alpha^* = \frac{\boldsymbol{a}^T(\boldsymbol{a}-\boldsymbol{z})}{\|\boldsymbol{a}-\boldsymbol{z}\|},$$

and it holds that

$$f(\alpha^*) \leq \frac{\|\boldsymbol{a}\|^2\|\boldsymbol{z}\|^2}{\|\boldsymbol{a}\|^2 + \|\boldsymbol{z}\|^2}.$$

Proof. Notice that α^* is the solution of $df/d\alpha = 0$. It is easy to see that $0 \leq \alpha^* \leq 1$ if and only if $\boldsymbol{a}^T\boldsymbol{z} \leq 0$. Finally, we have:

$$\begin{aligned} f(\alpha^*) &= \|\boldsymbol{a}\|^2 - \frac{(\boldsymbol{a}^T(\boldsymbol{a}-\boldsymbol{z}))^2}{\|\boldsymbol{a}-\boldsymbol{z}\|^2} = \|\boldsymbol{a}\|^2 - \frac{\|\boldsymbol{a}\|^4 - 2\boldsymbol{a}^T\boldsymbol{z}\|\boldsymbol{a}\|^2 + (\boldsymbol{a}^T\boldsymbol{z})^2}{\|\boldsymbol{a}\|^2 - 2\boldsymbol{a}^T\boldsymbol{z} + \|\boldsymbol{z}\|^2} \\ &= \frac{\|\boldsymbol{a}\|^2\|\boldsymbol{z}\|^2 - (\boldsymbol{a}^T\boldsymbol{z})^2}{\|\boldsymbol{a}\|^2 + \|\boldsymbol{z}\|^2 - 2\boldsymbol{a}^T\boldsymbol{z}} \leq \frac{\|\boldsymbol{a}\|^2\|\boldsymbol{z}\|^2}{\|\boldsymbol{a}\|^2 + \|\boldsymbol{z}\|^2}. \end{aligned}$$

□

References

[1] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: *np*-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc. (N.S.)*, 21(1):1–46, 07 1989.

[2] S. Boyd, L. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Studies in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1994. URL: https://books.google.co.jp/books?id=H2Nxxi5_fo0C.

[3] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015. URL: http://dx.doi.org/10.1561/2200000050, doi:10.1561/2200000050.

[4] S. Chubanov. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, 153(2):687–713, 2015.

[5] S. Chubanov. A polynomial algorithm for linear feasibility problems given by separation oracles. *Optimization Online*, Jan. 2017. URL: http://www.optimization-online.org/DB_HTML/2017/01/5838.html.

[6] D. Dadush, L. A. Végh, and G. Zambelli. Rescaling Algorithms for Linear Conic Feasibility. *ArXiv e-prints*, Nov. 2016. arXiv:1611.06427.

[7] J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards B. Mathematics and Mathematical Physics*, 71b(4):241–245, 1967.

[8] S. Fujishige. A note on submodular function minimization by Chubanov's lp algorithm. *Optimization Online*, Sept. 2017. URL: http://www.optimization-online.org/DB_HTML/2017/09/6217.html.

[9] M. Goberna and M. López. Linear semi-infinite programming theory: An updated survey. *European Journal of Operational Research*, 143(2):390 – 405, 2002.

[10] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, Jun 1981.

[11] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[12] D. H. Gutman. Enhanced Basic Procedures for the Projection and Rescaling Algorithm. *ArXiv e-prints*, July 2018. arXiv:1807.05982.

[13] R. Hettich and K. Kortanek. Semi-infinite programming: Theory, methods, and applications. *SIAM Review*, 35(3):380–429, 1993.

[14] L. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72, 1980.

[15] T. Kitahara and T. Tsuchiya. An extension of Chubanov's polynomial-time linear programming algorithm to second-order cone programming. *Optimization Methods and Software*, 33:1–25, 1 2018.

[16] D. Li, K. Roos, and T. Terlaky. A polynomial column-wise rescaling von Neumann algorithm. *Optimization Online*, June 2015. URL: http://www.optimization-online.org/DB_HTML/2015/06/4979.html.

[17] M. López and G. Still. Semi-infinite programming. *European Journal of Operational Research*, 180(2):491 – 518, 2007.

[18] B. F. Lourenço, T. Kitahara, M. Muramatsu, and T. Tsuchiya. An extension of Chubanov's algorithm to symmetric cones. *Mathematical Programming*, Nov 2017.

[19] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. A Wiley-Interscience publication. Wiley, 1983.

[20] J. Peña and N. Soheili. Solving conic systems via projection and rescaling. *Mathematical Programming*, 166(1):87–111, Nov 2017.

[21] J. Peña and N. Soheili. Computational performance of a projection and rescaling algorithm. *Optimization Online*, June 2018. URL: http://www.optimization-online.org/DB_HTML/2018/03/6532.html.

[22] M. V. Ramana. An exact duality theory for semidefinite programming and its complexity implications. *Mathematical Programming*, 77, 1995.

[23] K. Roos. An improved version of Chubanov's method for solving a homogeneous feasibility problem. *Optimization Methods and Software*, 33(1):26–44, 2018.

[24] W. Zhang and K. Roos. Using nemirovski's mirror-prox method as basic procedure in chubanov's method for solving homogeneous feasibility problems. *Optimization Online*, Apr. 2018. URL: http://www.optimization-online.org/DB_HTML/2018/04/6559.html.