Oracle-Based Primal-Dual Algorithms for Packing and Covering Semidefinite Programs

Khaled Elbassioni
Khalifa University of Science and Technology, Masdar City Campus,
P.O. Box 54224, Abu Dhabi, UAE
khaled.elbassioni@ku.ac.ae

Kazuhisa Makino
Research Institute for Mathematical Sciences (RIMS), Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract
Packing and covering semidefinite programs (SDPs) appear in natural relaxations of many combinatorial optimization problems as well as a number of other applications. Recently, several techniques were proposed, that utilize the particular structure of this class of problems, to obtain more efficient algorithms than those offered by general SDP solvers. For certain applications, such as those described in this paper, it may be required to deal with SDP’s with exponentially or infinitely many constraints, which are accessible only via an oracle. In this paper, we give an efficient primal-dual algorithm to solve the problem in this case, which is an extension of a logarithmic-potential based algorithm of Grigoriadis, Khachiyan, Porkolab and Villavicencio (SIAM Journal of Optimization 41 (2001)) for packing/covering linear programs.

2012 ACM Subject Classification
Mathematics of computing → Semidefinite programming; Theory of computation → Numeric approximation algorithms

Keywords and phrases
Semidefinite programs, packing and covering, logarithmic potential, primal-dual algorithms, approximate solutions

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.43

Related Version A full version of the paper is available at https://arxiv.org/abs/1809.09698.

1 Introduction
1.1 Packing and Covering SDPs

We denote by $\mathbb{S}^n$ the set of all $n \times n$ real symmetric matrices and by $\mathbb{S}_+^n \subseteq \mathbb{S}^n$ the set of all $n \times n$ positive semidefinite matrices. Consider the following pairs of packing-covering semidefinite programs (SDPs):

$$
\begin{align*}
  z_I^* &= \max \ C \cdot X \\
  \text{s.t.} \quad A_i \cdot X &\leq b_i, \forall i \in [m] \\
  X &\in \mathbb{R}^{n \times n}, \quad X \succeq 0
\end{align*}
$$

and

$$
\begin{align*}
  z^* &= \min \ b^T y \\
  \text{s.t.} \quad \sum_{i=1}^{m} y_i A_i &\succeq C \\
  y &\in \mathbb{R}^m, \quad y \geq 0
\end{align*}
$$

$$
\begin{align*}
  z_{II}^* &= \min \ C \cdot X \\
  \text{s.t.} \quad A_i \cdot X &\geq b_i, \forall i \in [m] \\
  X &\in \mathbb{R}^{n \times n}, \quad X \succeq 0
\end{align*}
$$

and

$$
\begin{align*}
  z_{II}^* &= \max \ b^T y \\
  \text{s.t.} \quad \sum_{i=1}^{m} y_i A_i &\preceq C \\
  y &\in \mathbb{R}^m, \quad y \geq 0
\end{align*}
$$
where $C, A_1, \ldots, A_m \in S_n^+$ are (non-zero) positive semidefinite matrices, and $b = (b_1, \ldots, b_m)^T \in \mathbb{R}^n_+$ is a nonnegative vector. In the above, $C \cdot X := \text{Tr}(CX) = \sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij}$, and “$\succeq$” is the Löwner order on matrices: $A \succeq B$ if and only if $A - B$ is positive semidefinite. This type of SDPs arise in many applications, see, e.g. [19, 18] and the references therein.

We will make the following assumption throughout the paper:

(A) $b_i > 0$ and hence $b_i = 1$ for all $i \in [m]$.

It is known that, under assumption (A), strong duality holds for problems (P-I)-(C-I) (resp., (P-II)-(C-II)).

Let $\epsilon \in [0, 1]$ be a given constant. We say that $(X, y)$ is an $\epsilon$-optimal primal-dual solution for (P-I)-(C-I) if $(X, y)$ is a primal-dual feasible pair such that

$$C \cdot X \geq (1-\epsilon)b^Ty \geq (1-\epsilon)z^*_{II}. \tag{1}$$

Similarly, we say that $(X, y)$ is an $\epsilon$-optimal primal-dual solution for (P-II)-(C-II) if $(X, y)$ is a primal-dual feasible pair such that

$$C \cdot X \leq (1+\epsilon)b^Ty \leq (1+\epsilon)z^*_{II}. \tag{2}$$

Since in this paper we allow the number of constraints $m$ in (P-I) (resp., (C-II)) to be exponentially (or even infinitely) large, we will assume the availability of the following oracle:

Max$(Y)$ (resp., Min$(Y)$): Given $Y \in S_n^+$, find $i \in \arg\max_{i \in [m]} A_i \cdot Y$ (resp., $i \in \arg\min_{i \in [m]} A_i \cdot Y$).

Note that an approximation oracle computing the maximum (resp., minimum) above within a factor of $(1-\epsilon)$ (resp., $(1+\epsilon)$) is also sufficient for our purposes.

Our objective in this paper is to develop oracle-based algorithms that find $\epsilon$-optimal solutions for (P-I)-(C-I) and (P-II)-(C-II). An interesting property of our algorithms which distinguishes them from most previously known algorithms is that they produce solutions which are sparse, in the following sense: A primal-dual solution $(X, y)$ to (C-I) (resp., (P-II)) is said to be $\eta$-sparse, if the size of $\text{supp}(y) := \{i \in [m] : y_i > 0\}$ is at most $\eta$. Two applications for SDP’s with infinite/exponential number of constraints are given in Section 3.

1.2 Main Result and Related Work

Problems (P-I)-(C-I) and (P-II)-(C-II) can be solved using general SDP solvers, such as interior-point methods: for instance, the barrier method (see, e.g., [29]) can compute a solution, within an additive error of $\epsilon$ from the optimal, in time $O(\sqrt{nm}(n^3 + mn^2 + m^2) \log \frac{1}{\epsilon})$ (see also [1, 34]). However, due to the special nature of (P-I)-(C-I) and (P-II)-(C-II), better algorithms can be obtained. Most of the improvements are obtained by using first order methods [4, 6, 7, 2, 13, 19, 20, 21, 22, 28, 30, 31], or second order methods [17, 18]. In general, we can classify these algorithms according to whether they:

(I) are width-independent: the running time of the algorithm depends polynomially on the bit length of the input; for example, in the of case of (P-I)-(C-I), the running time is $\text{poly}(n, m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$, where $\mathcal{L}$ is the maximum bit length needed to represent any number in the input; on the other hand, the running time of a width-dependent algorithm will depend polynomially on a “width parameter” $\rho$, which is polynomial in $\mathcal{L}$ and $\tau$;

(II) are parallel: the algorithm takes $\text{polylog}(n, m, \mathcal{L}, \log \tau) \cdot \text{poly}(\frac{1}{\epsilon})$ time, on a $\text{poly}(n, m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$ number of processors;
(III) output sparse solutions: the algorithm outputs an $\eta$-sparse solution to (C-I) (resp., (P-II)), for $\eta = \text{poly}(n, \log m, \mathcal{L}, \log \tau, \frac{1}{\epsilon})$ (resp., $\eta = \text{poly}(n, \log m, \mathcal{L}, \frac{1}{\epsilon})$), where $\tau$ is a parameter that bounds the trace of any optimal solution $X$ (see Section 2 for details);

(IV) are oracle-based: the only access of the algorithm to the matrices $A_1, \ldots, A_m$ is via the maximization/minimization oracle, and hence the running time is independent of $m$.

Table 1 gives a summary of the most relevant results together with their classifications, according to the four criteria described above. We note that almost all these algorithms for packing/covering SDPs are generalizations of similar algorithms for packing/covering linear programs (LPs), and most of them are essentially based on an exponential potential function in the form of scalar exponentiation, e.g., [4, 22], or matrix exponentiation [6, 7, 2, 21, 19]. For instance, several of these results use the scalar or matrix versions of the multiplicative weights updates (MWU) method (see, e.g., [5]), which are extensions of similar methods for packing/covering LPs [14, 15, 35, 32].

In [16], a different type of algorithm was given for covering LPs (indeed, more generally, for a class of concave covering inequalities) based on a logarithmic potential function. In this paper, we show that this approach can be extended to provide oracle-based algorithms for both versions of packing and covering SDPs:

\begin{theorem}
For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-I)-(C-I), outputs an $O(n\mathcal{L}\log(n\tau) + \frac{n}{\epsilon^2})$-sparse $O(\epsilon)$-optimal primal-dual pair in time $\tilde{O}(\frac{n^{\omega+1}\mathcal{L}\log \tau}{\epsilon^2} + \frac{nLT\mathcal{L}\log \tau}{\epsilon^2})$, where $T$ is the time taken by a single call to the oracle Max(-)$\omega$ and $\omega$ is the exponent of matrix multiplication.
\end{theorem}

\begin{theorem}
For any $\epsilon > 0$, there is a randomized algorithm that, for any given instance of (P-II)-(C-II), outputs an $O(n\mathcal{L}\log n + \frac{n}{\epsilon^2})$-sparse $O(\epsilon)$-optimal primal-dual pair in time $\tilde{O}(\frac{n^{\omega+1}\mathcal{L}\log n}{\epsilon^2} + \frac{nLT\mathcal{L}}{\epsilon^2})$, where $T$ is the time taken by a single call to the oracle Min(-).
\end{theorem}

As we can see from the table, among all the algorithms listed, the logarithmic-potential algorithm, presented in this paper, is the only one that produces sparse solutions, in the sense described above. Moreover, the only known other oracle-based algorithm (matrix MWU [7]) is not width-independent. It can also be shown (see [12]) that a modified version of the matrix exponential MWU algorithm [6] can yield sparse solutions for (P-II)-(C-II). However, the overall running time of this matrix MWU algorithm is larger by a factor of (roughly) $\Omega(n^{3-\omega})$ than that of the logarithmic-potential algorithm. Moreover, we were not able to extend the matrix MWU algorithm to solve (P-I)-(C-I) (in particular, it seems tricky to bound the number of iterations).

A work that is also related to ours is the sparsification of graph Laplacians [8] and positive semidefinite sums [33]. Given matrices $A_1, \ldots, A_m \in \mathbb{S}^n_+$ and $\epsilon > 0$, it was shown in [33] that one can find, in $O\left(\frac{\omega}{\epsilon^2} (n^\omega + T)\right)$ time, a vector $y \in \mathbb{R}^m_+$ with support size $O\left(\frac{n}{\epsilon^2}\right)$, such that $B \preceq \sum_i y_i A_i \preceq (1 + \epsilon)B$, where $B := \sum_i A_i$ and $T$ is the time taken by a single call to the minimization oracle $\text{Min}(Y)$ (for a not necessarily positive semidefinite matrix $Y$). An immediate corollary is that, given an $\epsilon$-optimal solution $y$ for (C-I) (resp., (P-I)), one can find in $O\left(\frac{\omega}{\epsilon^2} (n^\omega + T)\right)$ time an $O(\epsilon)$-optimal solution $y'$ with support size $O\left(\frac{n}{\epsilon^2}\right)$. Interestingly, the algorithm in [33] (which is an extension for the rank-one version in [8]) uses the barrier potential function $\Phi'(x, F) := \text{Tr}((H - xI)^{-1})$ (resp., $\Phi'(x, H) := \text{Tr}((xI - H)^{-1})$), while in

\footnotesize
\begin{itemize}
  \item[1] We provide rough estimates of the bounds, as some of them are not explicitly stated in the corresponding paper in terms of the parameters we consider here.
  \item[2] $\tilde{O}(\cdot)$ hides polylogarithmic factors in $n$ and $\frac{1}{\epsilon}$.
\end{itemize}
When $C = I = I_n$, the identity matrix in $\mathbb{R}^{n \times n}$ and $b = 1$, the vector of all ones in $\mathbb{R}^m$, we say that the packing-covering SDPs are in normalized form:

\[
z^*_I = \max \quad I \mathbf{1} \cdot X \quad \text{(N-P-I)}
\]
\[
\text{s.t.} \quad A_i \cdot X \leq 1, \forall i \in [m], \\
X \in \mathbb{R}^{n \times n}, \; X \succeq 0
\]

\[
z^*_I = \min \quad \mathbf{1}^T y \quad \text{(N-C-I)}
\]
\[
\text{s.t.} \quad \sum_{i=1}^m y_i A_i \succeq I, \\
y \in \mathbb{R}^m, \; y \succeq 0.
\]
where, for a positive semidefinite matrix \( B \), we may assume in this normalized form that

\[
\min \quad I \bullet X \\
\text{s.t.} \quad A_i \bullet X \geq 1, \quad i \in [m] \\
X \in \mathbb{R}^{n \times n}, \quad X \succeq 0
\]

\[
\max \quad 1^T y \\
\text{s.t.} \quad \sum_{i=1}^m y_i A_i \preceq I \\
y \in \mathbb{R}^m, \quad y \succeq 0.
\]

It can be shown\(^4\) that, at the loss of a factor of \((1 + \epsilon)\) in the objective, any pair of packing-covering SDPs of the form (P-I)-(C-I) can be brought in \(O(n^3)\), increasing the oracle time only by \(O(n^\omega)\), where \( \omega \) is the exponent of matrix multiplication, to the normalized form (P-NP)-I(N-C-I), under the following assumption:

(B-I) There exist \( r \) matrices, say \( A_1, \ldots, A_r \), such that \( \bar{A} := \sum_{i=1}^r A_i > 0 \). In particular, \( \text{Tr}(X) \leq \tau := \frac{\text{Tr}(\bar{A})}{\lambda_{\min}(\bar{A})} \) for any optimal solution \( X \) for (P-I).

Similarly, one can show (see \([12, 20]\)) that, at the loss of a factor of \((1 + \epsilon)\) in the objective, any pair of packing-covering SDPs of the form (P-II)-(C-II) can be brought in \(O(n^3)\) time, increasing the oracle time only by \(O(n^\omega)\), to the normalized form (P-NP-II)-(N-C-II). Moreover, we may assume in this normalized form that

(B-II) \( \lambda_{\min}(A_i) = O(\frac{\tau}{\omega} \cdot \min_j \lambda_{\max}(A_{ij})) \) for all \( i \in [m] \),

where, for a positive semidefinite matrix \( B \in \mathbb{S}^{n \times n}_+ \), we denote by \( \{ \lambda_j(B) : j = 1, \ldots, n \} \) the eigenvalues of \( B \), and by \( \lambda_{\min}(B) \) and \( \lambda_{\max}(B) \) the minimum and maximum eigenvalues of \( B \), respectively. With an additional \( O(mn^2) \) time, we may also assume that:

(B-II') \( \frac{\lambda_{\max}(A_i)}{\lambda_{\min}(A_i)} = O(\frac{n^2}{\omega}) \) for all \( i \in [m] \).

Thus, from now on we focus on the normalized problems.

## 3 Applications

### 3.1 Robust Packing and Covering SDPs

Consider a packing-covering pair of the form (P-I)-(C-I) or (P-II)-(C-II). In the framework of robust optimization (see, e.g. \([9, 10]\)), we assume that each constraint matrix \( A_i \) is not known exactly; instead, it is given by a convex uncertainty set \( A_i \subseteq \mathbb{S}^n_+ \). It is required to find a (near)-optimal solution for the packing-covering pair under the worst-case choice \( A_i \in A_i \) of the constraints in each uncertainty set. A typical example of a convex uncertainty set is given by an affine perturbation around a nominal matrix \( A_i^0 \in \mathbb{S}^n_+ \):

\[
A_i = \left\{ A_i := A_i^0 + \sum_{r=1}^k \delta_r A_r^\delta : \delta = (\delta_1, \ldots, \delta_k) \in \mathcal{D} \right\},
\]

where \( A_1^\delta, \ldots, A_k^\delta \in \mathbb{S}^n_+ \), and \( \mathcal{D} \subseteq \mathbb{R}^k_+ \) can take, for example, one of the following forms:

- Ellipsoidal uncertainty: \( \mathcal{D} = E(\delta_0, D) := \{ \delta \in \mathbb{R}^k_+ : (\delta - \delta_0)^T D^{-1} (\delta - \delta_0) \leq 1 \} \), for given positive definite matrix \( D \in \mathbb{S}^k_+ \) and vector \( \delta_0 \in \mathbb{R}^k_+ \) such that \( E(\delta_0, D) \subseteq \mathbb{R}^k_+ \);

- Polyhedral uncertainty: \( \mathcal{D} := \{ \delta \in \mathbb{R}^k_+ : D\delta \leq w \} \), for given matrix \( D \in \mathbb{R}^{k \times k} \) and vector \( w \in \mathbb{R}^k \).

\(^4\) In fact, unlike previously known reductions, the reduction we give in [12] is simpler to compute, as it is based on the \( LDL\)-decompositions rather than the eigenvalue decompositions of the input matrices.
Without loss of generality, we consider the robust version of \((N-P-I)-(N-C-I)\), where \(A_i\), for \(i \in [m]\), belongs to a convex uncertainty set \(A_i\). Then the robust optimization problem and its dual can be written as follows:

\[
\begin{align*}
\min_{X} & \quad \mathbf{1} \cdot X \\
\text{s.t.} & \quad A_i \cdot X \leq 1, \quad \forall A_i \in A, \quad \forall i \in [m] \\
& \quad X \in \mathbb{R}^{n \times n}, \quad X \succeq 0
\end{align*}
\]

\(\star\)

\[
\begin{align*}
\min_{\mathbf{y}} & \quad \sum_{i=1}^{m} \int_{A_i} y_i A_i dA_i \\
\text{s.t.} & \quad \sum_{i=1}^{m} \int_{A_i} y_i A_i dA_i \geq \mathbf{1} \\
& \quad y' is a discrete measure on \(A_i\), \quad \forall i \in [m].
\end{align*}
\]

As before, we assume \((B-I)\), where \(A_1, \ldots, A_r \in \bigcup_{i \in [m]} A_i\). We call a pair of solutions \((X, \mathbf{y})\) to be \(\epsilon\)-optimal for \((R-P-I)-(R-C-I)\), if

\[
\begin{align*}
z^\star_P & \geq \mathbf{1} \cdot X \geq (1 - \epsilon) \sum_{i=1}^{m} \int_{A_i} y_i A_i dA_i \geq (1 - \epsilon) z^\star_D.
\end{align*}
\]

Note that the number of constraints in \((R-P-I)\) is infinite and hence any algorithm that solves the problem would have to be oracle-based. The Ellipsoid method is one such algorithm; a more efficient procedure is given by the following corollary of Theorem 1.

\[\textbf{Theorem 3.}\] For any \(\epsilon > 0\), there is a randomized algorithm that outputs an \(O(\epsilon)\)-optimal primal-dual pair for \((R-P-I)-(R-C-I)\) in time \(\tilde{O}(\frac{m^{2+\epsilon^2}}{\epsilon^{2+\epsilon^2}} + \frac{nT \log \psi}{\epsilon^2})\), where \(\psi := \frac{r \cdot \max_{i \in [m], A_i \in A_i} \lambda_{\max}(A_i)}{\lambda_{\min}(A)}\) and \(T\) is the time to compute, for a given \(Y \in \mathbb{S}^n_+\), a pair \((i, A_i)\) such that

\[\begin{align*}
(i, A_i) \in \arg\max_{i \in [m], A_i \in A_i} A_i \cdot Y.
\end{align*}\]

Note that (4) amounts to solving a linear optimization problem over a convex set. Moreover, for simple uncertainty sets, such as boxes or ellipsoids, such computation can be done very efficiently.

### 3.2 Carr-Vempala-Type Decomposition

Consider a maximization (resp., minimization) problem over a discrete set \(S \subseteq \mathbb{Z}^n\) and a corresponding SDP-relaxation over \(Q \subseteq \mathbb{S}^n_+\):

\[
\begin{align*}
z^\star_{CO} &= \left\{ \begin{array}{ll}
\max \min & \quad C \cdot qq^T \\
q \in S & \quad (DOP)
\end{array} \right. \\
z^\star_{SDP} &= \left\{ \begin{array}{ll}
\max \min & \quad C \cdot Q \\
Q \in Q, & \quad (SDP-RLX)
\end{array} \right.
\end{align*}
\]

where \(C \in \mathbb{S}^n_+\).

\[\textbf{Definition 4.}\] For \(\alpha \in (0, 1]\) (resp., \(\alpha \geq 1\)), an \(\alpha\)-integrality gap verifier \(A\) for \((SDP-RLX)\) is a polytime algorithm that, given any \(C \in \mathbb{S}^n_+\) and any \(Q \in Q\) returns a \(q \in S\) such that \(C \cdot qq^T \geq \alpha C \cdot Q\) (resp., \(C \cdot qq^T \leq \alpha C \cdot Q\)).

For instance, if \(S = \{-1, 1\}^n\) and \(Q = \{X \in \mathbb{S}^n_+ : X_{ii} = 1 \quad \forall i \in [n]\}\), then a \(\frac{2}{\pi}\)-integrality gap verifier for the maximization version of \((SDP-RLX)\) is known [26].
Carr and Vempala [11] gave a decomposition theorem that allows one to use an \( \alpha \)-integrality gap verifier for a given LP-relaxation of a combinatorial maximization (resp., minimization) problem, to decompose a given fractional solution to the LP into a convex combination of integer solutions that is dominated by (resp., dominates) \( \alpha \) times the fractional solution. Here we derive a similar result for SDP relaxations:

**Theorem 5.** Consider a combinatorial maximization (resp., minimization) problem (DOP) and its SDP relaxation (SDP-RLX), admitting an \( \alpha \)-integrality gap verifier \( A \). Assume the set \( S \) is full-dimensional and let \( \epsilon > 0 \) be a given constant. Then there is a polytime algorithm that, for any given \( Q \in Q \), finds a set \( X \subseteq S \) of size

\[
|X| = O\left( \frac{n^3 \epsilon^2}{\log(nW)} \right) \quad \text{(resp., of size } |X| = O\left( n \log n + n \epsilon^2 \right) \text{), where } W := \max_{q \in S, i \in [n]} |q_i|, \text{ and a set of convex multipliers } \{ \lambda_q \in \mathbb{R}_+: q \in X \} \text{ such that }
\]

\[
(1 - O(\epsilon))\alpha Q \preceq \sum_{q \in X} \lambda_q q q^T \quad \text{(resp., } (1 + O(\epsilon))\alpha Q \succeq \sum_{q \in X} \lambda_q q q^T \text{).} 
\]

The proof of Theorem 5 is obtained by considering the following pairs of packing and covering SDPs (of types I and II, respectively):

\[
z^*_I = \min \sum_{q \in S} \lambda_q \quad \text{(CVX-I)}
\]

s.t. \[
\sum_{q \in S} \lambda_q q q^T \geq \alpha Q \quad \text{(6)}
\]

\[
\sum_{q \in S} \lambda_q \geq 1 \quad \text{(7)}
\]

\[
\lambda \in \mathbb{R}^S, \; \lambda \geq 0
\]

\[
z^*_II = \max \sum_{q \in S} \lambda_q \quad \text{(CVX-II)}
\]

s.t. \[
\sum_{q \in S} \lambda_q q q^T \leq \alpha Q \quad \text{(9)}
\]

\[
\sum_{q \in S} \lambda_q \leq 1 \quad \text{(10)}
\]

\[
\lambda \in \mathbb{R}^S, \; \lambda \geq 0
\]

\[
z^*_I = \max \alpha Q \cdot Y + u \quad \text{(CVX-dual-I)}
\]

s.t. \[
qq^T \cdot Y + u \leq 1, \forall q \in S \quad \text{(8)}
\]

\[
Y \in \mathbb{S}^n_+, \; u \geq 0.
\]

\[
z^*_II = \min \alpha Q \cdot Y + u \quad \text{(CVX-dual-II)}
\]

s.t. \[
qq^T \cdot Y + u \geq 1, \forall q \in S \quad \text{(11)}
\]

\[
Y \in \mathbb{S}^n_+, \; u \geq 0.
\]

It can be shown, using the fact that the SDP relaxation admits an \( \alpha \)-integrality gap verifier, that \( z^*_I = z^*_II = 1 \), and that the two primal-dual pairs can be solved in polynomial time using the Ellipsoid method. A more efficient (but approximate version) can be obtained using the algorithms of Theorems 1 and 2. Note that, once we have a set \( X \) as in Theorem 5, its support can be reduced to \( O\left( n^2 \epsilon \right) \) using the sparsification techniques of \([8, 33]\).

### 4 A Logarithmic Potential Algorithm for (P-I)-(C-I)

In this section we give an algorithm for finding a sparse \( O(\epsilon) \)-optimal primal-dual solution for (N-P-I)-(N-C-I). Since the algorithm updates only one component of the dual solution in each iteration, it follows that the number of positive components of the dual solution when the algorithm terminates is exactly equal to the number of iterations; from this sparsity follows.
4.1 High-level Idea of the Algorithm

The idea of the algorithm is quite intuitive. It can be easily seen that problem (N-C-I) is equivalent to finding a convex combination of the $A_i$’s that maximizes the minimum eigenvalue, that is, $\max_{y \in \mathbb{R}^m_+} \sum_{i=1}^m y_i A_i$, and 1 is the $m$-dimensional vector of all ones. Since $\lambda_{\min}(F(y))$ is not a smooth function in $y$, it is more convenient to work with a smooth approximation of it, which is obtained by maximizing (over $x$) a logarithmic potential function $\Phi(x, F(y))$ that captures the constraints that each eigenvalue of $F(y)$ is at least $x$. The unique maximizer $x = \theta^* \text{ of } \Phi(x, F(y))$ defines a set of “weights” (these are the eigenvalues of the primal matrix $X$ computed in line 6 of the algorithm) such that the weighted average of the $\lambda_j(F(y))$’s is a very close approximation of $\lambda_{\min}(F(y))$. Thus, to maximize this average (which is exactly $X \cdot F(y)$), we obtain a direction (line 7) along which $y$ is modified with an appropriate step size (line 10). For numbers $x \in \mathbb{R}_+$ and $\delta \in (0, 1)$, a $\delta$-(lower) approximation $x_\delta$ of $x$ is a number such that $(1 - \delta)x \leq x_\delta < x$. For $i \in [m]$, $\mathbf{1}_i$ denotes the $i$th unit vector of dimension $m$.

The algorithm is shown as Algorithm 1. The main while-loop (step 4) is embedded within a sequence of scaling phases, in which each phase starts from the vector $y(t)$ computed in the previous phase and uses double the accuracy. The algorithm stops when the scaled accuracy $\varepsilon_s$ drops below the desired accuracy $\epsilon \in (0, 1/2)$. When referring to an arbitrary iteration of the algorithm, we assume it is iteration $t$ in phase $s$.

4.2 Analysis

4.2.1 High-level Idea of the Analysis

The analysis is based on a matrix generalization of the scalar arguments given in [16] (as is the case for all algorithms for SDP’s, which are driven from their LP counterparts; see, e.g., [1]). Besides the technical details, the algorithm also requires estimating the minimum eigenvalue of the dual matrix $F(y(t))$, which is done using Lanczos’ algorithm (see Section 4.2.4 for details).

The proof of $\epsilon$-optimality follows easily from the stopping condition in line 4 of the algorithm, the definition of the “approximation error” $\nu$ in line 8, and the fact that $X \cdot F(y)$ is a very close approximation of $\lambda_{\min}(F(y(t)))$. The main part of the proof is to bound the number of iterations in the inner while-loop (line 4). This is done by using a potential function argument: we define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$ and show in Claim 23 that, in each iteration, the choice of the step size in line 9 guarantees that $\Phi(t)$ is increased substantially; on the other hand, by Claim 24, the potential difference cannot be very large, and the two claims together imply that we cannot have many iterations.

4.2.2 Some Preliminaries

Up to Claim 26, we fix a particular iteration $s$ of the outer while-loop in the algorithm. For simplicity in the following, we will sometimes write $F := F(y(t)), \theta := \theta(t), \theta^* := \theta^*(t), X := X(t), \tilde{F} := A(t), \tau := \tau(t + 1), \nu := \nu(t + 1), F' := F(y(t + 1))$, and $\theta' := \theta(t + 1)$, when the meaning is clear from the context. For $H > 0$ and $x \in (0, \lambda_{\min}(H))$, define the logarithmic potential function $\Phi(x, H)$ in [16, 29]:

$$\Phi(x, H) = \ln x + \frac{\varepsilon_s}{n} \ln \det (H - xI). \quad (12)$$

Note that the term $\ln \det (H - xI)$ forces the value of $x$ to stay away from the “boundary” $\lambda_{\min}(H)$, while the term $\ln x$ pushes $x$ towards that boundary; hence, one would expect the maximizer of $\Phi(x, H)$ to be a good approximation of $\lambda_{\min}(H)$ (see Claim 8).
Algorithm 1: Logarithmic-potential Algorithm for (P-I)-(C-I).

1. \( s \leftarrow 0; \delta_0 \leftarrow \frac{1}{2}; t \leftarrow 0; \nu(0) \leftarrow 1; y(0) \leftarrow \frac{1}{p} \sum_{i=1}^{P} 1_i \)
2. while \( \delta_s > \epsilon \) do
3. \( \delta_s \leftarrow \frac{\epsilon_s}{2\nu} \)
4. \( \nu(t) > \epsilon_s \) do
5. \( \theta(t) \leftarrow \theta^*(t)_{\delta_s} \), where \( \theta^*(t) \) is the smallest positive number root of the equation \( \frac{\epsilon_s n}{\nu} \operatorname{Tr}(F(y(t)) - \theta(t)I)^{-1} = 1 \)
6. \( X(t) \leftarrow \frac{\epsilon_s \theta(t)}{n} (F(y(t)) - \theta(t)I)^{-1} \) /* Set the primal solution */
7. \( i(t) \leftarrow \arg\max_i A_i \cdot X(t) \) /* Call the maximization oracle */
8. \( \nu(t + 1) \leftarrow \frac{X(t) \cdot A_{i(t)} - X(t) \cdot F(y(t))}{X(t) \cdot A_{i(t)} + X(t) \cdot F(y(t))} \) /* Compute the error */
9. \( \tau(t + 1) \leftarrow \frac{4n(X(t) \cdot A_{i(t)} + X(t) \cdot F(y(t)))}{\epsilon_s \theta(t) \nu(t + 1)} \) /* Compute the step size */
10. \( y(t + 1) \leftarrow (1 - \tau(t + 1))y(t) + \tau(t + 1)1_{t(t)} \) /* Update the dual solution */
11. \( t \leftarrow t + 1 \)
12. end
13. \( \epsilon_{s+1} \leftarrow \frac{\epsilon_s}{\nu} \)
14. \( s \leftarrow s + 1 \)
15. end
16. \( \hat{X} \leftarrow \frac{(1 - \epsilon_{s+1})X(t)}{(1 + \epsilon_{s+1})\theta(t)}; \hat{y} \leftarrow \frac{y(t - 1)}{\theta(t - 1)} \) /* Scale primal and dual to retain feasibility */
17. return \((\hat{X}, \hat{y}, t)\)

Claim 6. If \( F(y(t)) > 0 \), then \( \theta^*(t) = \arg\max_{0 < x < \lambda_{\min}(F)} \Phi(x, F(y(t))) \) and \( X(t) > 0 \).

For \( x \in (0, \lambda_{\min}(F)) \), let \( g(x) := \frac{\epsilon_s n}{\nu} \operatorname{Tr}(F - xI)^{-1} \). The following claim shows that our choice of \( \delta_s \) guarantees that \( g(\theta) \) is a good approximation of \( g(\theta^*) = 1 \).

Claim 7. \( g(\theta(t)) \in (1 - \epsilon_s, 1) \).

The following two claims show that \( \theta(t) \approx X(t) \cdot F(y(t)) \) provides a good approximation for \( \lambda_{\min}(F(y(t))) \).

Claim 8. \( (1 - \epsilon_s)\lambda_{\min}(F(y(t))) < \theta(t) < \frac{\lambda_{\min}(F(y(t)))}{1 + \epsilon_s / n} \) and \( \frac{\lambda_{\min}(F(y(t)))}{1 + \epsilon_s / n} \leq \theta^*(t) \leq \frac{\lambda_{\min}(F(y(t)))}{1 + \epsilon_s / n} \).

Claim 9. \( \theta(t) < X(t) \cdot F(y(t)) < (1 + \epsilon_s)\theta(t) \).

Throughout the algorithm, we maintain the invariants that the (non-scaled) dual objective \( 1^T y(t) \) is exactly 1, that the step size \( \tau(t) \) and the approximation error \( \nu(t) \) are between 0 and 1, and that the dual matrix \( F(y(t)) = \sum_i y_i(t)A_i \) is positive definite. This is summarized in the following claims.

Claim 10. \( 1^T y(t) = 1 \).

Claim 11. For all iterations \( t \), except possibly the last, \( \nu(t + 1), \tau(t + 1) \in (0, 1) \).

Claim 12. \( F(y(t)) > 0 \).
4.2.3 Primal Dual Feasibility and Approximate Optimality

Let \( t_f + 1 \) be the value of \( t \) when the algorithm terminates and \( s_f + 1 \) be the value of \( s \) at termination. For simplicity, we write \( s = s_f \).

\[ \triangleright \text{Claim 13. (Primal feasibility).} \quad \tilde{X} \succ 0 \text{ and } \max_i A_i \cdot \tilde{X} \leq 1. \]

Proof. The first claim is immediate from Claim 6. To see the second claim, we use the definition of \( \nu(t_f) \) and the termination condition in line 4 (which is also satisfied even if \( X(t_f) \cdot A_i(t_f) = X(t_f) \cdot F(y(t_f)) = 0 \):)

\[
\frac{X(t_f) \cdot A_i(t_f) - X(t_f) \cdot F(y(t_f))}{X(t_1) \cdot A_i(t_f) + X(t_f) \cdot F(y(t_f))} \leq \varepsilon_s.
\]

\( \therefore (1 + \varepsilon_s)X(t_f) \cdot F(y(t_f)) \geq (1 - \varepsilon_s)X(t_f) \cdot A_i(t_f)
\]

\( = (1 - \varepsilon_s) \max_i X(t_f) \cdot A_i \quad \text{(by the definition of } i(t_f)) \)

\( \therefore (1 + \varepsilon_s)^2 \theta(t_f) \geq (1 - \varepsilon_s) \max X(t_f) \cdot A_i, \)

\( \therefore X(t_f) \cdot F(y(t_f)) \leq (1 + \varepsilon_s)\theta(t_f) \) by Claim 9)

The claim follows by the definition of \( \tilde{X} \) in step 16 of the algorithm.

\[ \triangleright \text{Claim 14. (Dual feasibility).} \quad \hat{y} \geq 0 \text{ and } F(\hat{y}) \succ I. \]

Proof. The fact that \( \hat{y} \geq 0 \) follows from the initialization of \( y(0) \) in step 1, Claim 11, and the update of \( y(t + 1) \) in step 10. For the other claim, we have

\[
\lambda_{\min}(F(\hat{y})) = \frac{1}{\theta(t_f)} \lambda_{\min}(F(y(t_f))) \geq 1 + \frac{\varepsilon_s}{n}, \quad \text{(by Claim 8)}
\]

\[ \triangleright \text{Claim 15. (Approximate optimality).} \quad I \cdot \tilde{X} \geq \left( \frac{1 - \varepsilon_s}{1 + \varepsilon_s} \right)^2 1^T \hat{y}. \]

Proof. By Claim 7, we have \( \text{Tr}(X(t_f)) \geq 1 - \varepsilon_s \), and by Claim 10, we have \( 1^T y(t_f) = 1 \). The claim follows by the definition of \( \tilde{X} \) and \( \hat{y} \) in step 16.

4.2.4 Running Time per Iteration

Given \( F := F(y(t)) \succ 0 \), we first compute an approximation \( \tilde{\lambda} \) of \( \lambda_{\min}(F) \) using Lanczos’ algorithm with a random start [25].

\[ \triangleright \text{Lemma 16 ([25]).} \quad \text{Let } M \in S^+_n \text{ be a positive semidefinite matrix with } N \text{ non-zeros and } \gamma \in (0, 1) \text{ be a given constant. Then there is a randomized algorithm that computes, with high (i.e., } 1 - o(1) \text{ probability a unit vector } v \in \mathbb{R}^n \text{ such that } v^T M v \geq (1 - \gamma) \lambda_{\max}(M). \text{ The algorithm takes } O\left( \frac{\sqrt{2} \cdot N}{\varepsilon_0} \right) \text{ iterations, each requiring } O(N) \text{ arithmetic operations.} \]

By Claim 8, we need \( \tilde{\lambda} \) to lie in the range \( \left[ \frac{\lambda_{\min}(F)}{1 + \varepsilon_0}, \lambda_{\min}(F) \right] \). To obtain \( \tilde{\lambda} \), we may apply the above lemma with \( M := F^{-1} \) and \( \gamma := \frac{\varepsilon_0}{\delta_0} \). Then in \( O\left( \frac{\sqrt{2}}{\varepsilon_0} \log n \right) \) iterations we get

\( \tilde{\lambda} := \frac{1 - \gamma}{\sqrt{2} \cdot \varepsilon_0} \) satisfying our requirement. However, we can save (roughly) a factor of \( \sqrt{n} \) in
the running time by using, instead, $M := F^{-n}$ and $\gamma := \frac{\epsilon}{\varepsilon}$. Let $v$ be the vector obtained from Lemma 16, and set $\tilde{\lambda} := \left(\frac{1 + \gamma}{\varepsilon F^{-n}}\right)^{1/n}$. Then, as $\lambda_{\max}(M) \geq v^T M v \geq (1 - \gamma) \lambda_{\max}(M)$, and $\lambda_{\min}(F) = \lambda_{\max}(F^{-n})^{-1/n}$, we get
\begin{equation}
\frac{\lambda_{\min}(F)}{1 + \varepsilon/n} \leq (1 - \gamma)^{1/n} \lambda_{\min}(F) \leq \tilde{\lambda} \leq \lambda_{\min}(F). (13)
\end{equation}

Note that we can compute $F^{-n}$ in $O(n^\omega \log n)$, where $w$ is the exponent of matrix multiplication. Thus, the overall running time for computing $\tilde{\lambda}$ is $O(n^\omega \log n + \frac{n^2 \log n}{\sqrt{\varepsilon}})$.

Given $\tilde{\lambda}$, we know by Claim 8 and (13) that $\theta^*(t) \in \left[\frac{\tilde{\lambda}}{1 + \varepsilon/n}, \tilde{\lambda}\right]$. Then we can apply binary search to find $\theta(t) := \theta^*(t)\delta_s$ as follows. Let $s_k = \frac{\tilde{\lambda}}{1 + \varepsilon/n}(1 + \delta_s)^k$, for $k = 0, 1, \ldots, K := [\frac{2\ln(1 + \varepsilon/n)}{2\delta_s}]$, and note that $s_k \geq \tilde{\lambda}$. Then we do binary search on the exponent $k \in \{0, 1, \ldots, K\}$; each step of the search evaluates $g(\theta_k) := \frac{\varepsilon s_k \theta_t}{n} \text{Tr}(F - \theta_k I)^{-1}$, and depending on whether this value is less than or at least 1, the value of $k$ is increased or decreased, respectively. The search stops when the search interval $[t, u]$ has $u - t + 1$, in which case we set $\theta(t) = \theta_t$; the number of steps until this happens is $O(\log K) = O(\log \frac{n}{\varepsilon^2})$. By the monotonicity of $g(x)$ (in the interval $[0, \lambda_{\min}(F)]$), and the property of binary search, we know that $\theta^* \in [\theta_t, \theta_u]$. Thus, by the stopping criterion,
\[ \theta(t) = \theta_t \leq \theta^*(t) \leq \theta_u \leq \theta_{t+1} = (1 + \delta_s)\theta_t, \]

implying that $(1 - \delta_s)\theta^*(t) \leq \theta(t) \leq \theta^*(t)$. Since evaluating $g(\theta_t)$ takes $O(n^\omega)$, the overall running time for the binary search procedure is $O(n^\omega \log \frac{n}{\varepsilon^2})$, and hence the total time needed for computing $\theta(t)$ is $O(n^\omega \log \frac{n}{\varepsilon^2} + \frac{n^2 \log n}{\sqrt{\varepsilon}})$.

All other steps of the algorithm inside the inner while-loop can be done in $O(T + n^2)$ time, where $T$ is the time taken by a single call to the oracle $\text{Max}(X(t))$ in step 7 of the algorithm. Thus, in view of Claim 26 on the number of iterations below, we obtain Theorem 1.

### 4.2.5 Number of Iterations

Define $B = B(t) := \frac{\varepsilon s_t}{\sqrt{\varepsilon}} \left(\tau X^{1/2} (F - F) X^{1/2} - (\theta^* - \theta) X\right)$. The following (technical) claims are needed for the proofs of Claims 23 and 24 below (which are, in turn, the main claims needed for the analysis of the potential function). They can be skipped at a first reading and recalled when needed.

- **Claim 17.** $(F - \theta^* I)^{-1} = (\frac{\varepsilon s_t}{\sqrt{\varepsilon}} I - (\theta^* - \theta) X)^{-1} X$.
- **Claim 18.** $F^* - \theta^* I = (F - \theta I)^{1/2}(I + B)(F - \theta I)^{1/2}$.
- **Claim 19.** $\max_j |\lambda_j(B)| \leq \frac{1}{2}$.
- **Claim 20.** $\theta^*(t) < \lambda_{\min}(F(y(t + 1)))$.
- **Claim 21.** If $\nu > \varepsilon_s$, then $\text{Tr}(B) \geq \frac{n^2}{\nu^2}$.
- **Claim 22.** If $\nu > \varepsilon_s$, then $\text{Tr}(B^2) < \frac{n^2}{\nu^2}$.

Define the potential function $\Phi(t) := \Phi(\theta^*(t), F(y(t)))$.

The following claim states that the potential difference between two consecutive iterations of the algorithm is sufficiently large.
Claim 23. For \( t, t + 1 \) in phase \( s \), \( \Phi(t + 1) - \Phi(t) \geq \frac{\varepsilon_s \lambda(t+1)^2}{40n} \).

Proof. Note that Claim 20 implies that \( \theta^* \) is feasible to the problem \( \max \{ \Phi(\xi, F') : 0 \leq \xi \leq \lambda_{\min}(F') \} \). Thus,

\[
\Phi(t + 1) = \Phi(\theta^*(t + 1), F') \geq \ln(\theta^* + \frac{\varepsilon_s}{n} \ln \det(F' - \theta^* I))
\]

\[
\therefore (t + 1) - \Phi(t) \geq \frac{\varepsilon_s}{n} \left( \ln \det(F' - \theta^* I) - \ln \det(F - \theta^* I) \right)
\]

\[
= \frac{\varepsilon_s}{n} \ln \det(I + B)
\]

\[
= \frac{\varepsilon_s}{n} \sum_{j=1}^{n} \ln(1 + \lambda_j(B))
\]

\[
\geq \frac{\varepsilon_s}{n} \sum_{j=1}^{n} \left( \lambda_j(B) - \lambda_j(B)^2 \right)
\]

(by Claim 19 and \( \ln(1 + z) \geq z - z^2, \forall z \geq -0.5 \))

\[
= \frac{\varepsilon_s}{n} \left( \text{Tr}(B) - \text{Tr}(B^2) \right)
\]

\[
\geq \frac{\varepsilon_s}{8n} \nu^2 - \frac{\varepsilon_s}{10n} \nu^2
\]

(by Claims 21 and 22)

\[
= \frac{\varepsilon_s}{40n} \nu^2.
\]

\( \blacksquare \)

On the other hand, the following claim states that the overall potential difference between any iterations cannot be too large.

Claim 24. For any \( t, t' \) in phase \( s \),

\[
\Phi(t') - \Phi(t) \leq \left( 1 + \varepsilon_s \right) \ln \frac{X(t) \bullet A_{i(t)}}{(1 - \varepsilon_s)X(t) \bullet F(y(t))}.
\]

Proof. Write \( F = F(y(t)), \theta^* := \theta^*(t), \theta := \theta(t), X := X(t), F' = F(y(t')), \theta'^* := \theta^*(t') \).

Then

\[
\Phi(t') - \Phi(t) = \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \det \left[ (F - \theta^* I)^{-1}(F' - \theta^* I) \right]
\]

\[
= \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \left[ \left( \frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta)X \right)^{-1} X(F' - \theta^* I) \right]
\]

(by Claim 17)

\[
= \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \left[ \ln \det \left( \frac{\varepsilon_s \theta}{n} I - (\theta^* - \theta)X \right)^{-1} + \ln \det \left[ X(F' - \theta^* I) \right] \right]
\]

\[
\leq \ln \frac{\theta'^*}{\theta^*} + \frac{\varepsilon_s}{n} \ln \left( \frac{\varepsilon_s \theta}{n} - \frac{\delta_s \theta}{1 - \delta_s} \right)^{-n} + \ln \det \left[ X(F' - \theta^* I) \right]
\]

(by \( \text{Tr}(X) \leq 1 \) by Claim 7 and \( 1 - \delta_s \theta^* \leq \theta \))
The claim follows.

Recall by assumption (B-I) that K. Elbassioni and K. Makino 43:13

Claim 25.

Proof.

Now we combine claims 23, 24 and 25 to obtain a bound on the number of iterations.
Claim 26. The algorithm terminates in at most $O(n \log \psi + \frac{n}{\epsilon})$ iterations.

Remark 27. If we do not insist on a sparse dual solution, then we can use the initialization $y(0) \leftarrow \frac{1}{m}1$ in step 1 in Algorithm 1, and replace $\psi$ in Claim 25, and hence in the running time in Claim 26, by $m$.

References


26 Y. Nesterov. Quality of semidefinite relaxation for nonconvex quadratic optimization. CORE Discussion Papers 1997019, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), March 1997.


