1 NUMERICAL EVALUATION OF SDPA (SEMIDEFINITE PROGRAMMING ALGORITHM)

Katsuki Fujisawa[1], Mituhiro Fukuda[2], Masakazu Kojima[3] and Kazuhide Nakata[4]

[1]Kyoto University,
Kyoto, Japan
fujisawa@is-mj.archi.kyoto-u.ac.jp

[2]Tokyo Institute of Technology,
Tokyo, Japan.
mituhiro@is.titech.ac.jp

[3]Tokyo Institute of Technology,
Tokyo, Japan.
kojima@is.titech.ac.jp

[4]Japan Sun Microsystems K.K.,
Tokyo, Japan
kazuhide.nakata@sun.co.jp

Abstract. SDPA (SemiDefinite Programming Algorithm) is a C++ implementation of a Mehrotra-type primal-dual predictor-corrector interior-point method for solving the standard form semidefinite program and its dual. We report numerical results of large scale problems to evaluate its performance, and investigate how major time-consuming parts of SDPA vary with the problem size, the number of constraints and the sparsity of data matrices.

1.1 INTRODUCTION.

The main purpose of this paper is to evaluate the performance of SDPA (SemiDefinite Programming Algorithm) [6] for semidefinite programs. Besides SDPA, there are some computer programs SDPpack [3], SDP-SOL [28], CSDP [5], SDPHA [21] and SDPT3 [23] for semidefinite programs which are available through the Internet. Among others, we mainly compare SDPA with SDPT3 through numerical experiments on several types of test problems. Either of them is an implementation of a Mehrotra-type [15] primal-dual predictor-corrector interior-point method. The choice of SDPT3 as a competitor of SDPA was based on the preliminary numerical experiments given in Section 4. Although three types of search directions, the HRVW/KSH/M direction [11, 14, 17], the NT direction [19, 20, 22] and the AHO direction [1, 2] are available in both of SDPA and SDPT3, we employed the HRVW/KSH/M direction in our numerical experiments because its computation is the cheapest among the three directions (particularly, for sparse data matrices) when we employ the method proposed by Fujisawa et al. [7]. Monteiro et al. [18] recently showed that in theory, the NT direction requires less computation for dense matrices. However, their method needs large amount of memory and does not efficiently exploit the sparse data structures. Actually, according to their numerical results, the computation of the HRVW/KSH/M direction is favorable compared to the computation of the NT and AHO directions.

The main differences between SDPA and SDPT3 are:

(a) The programming languages used for SDPA and SDPT3 are different; the former is written in C++ while the latter is written in MATLAB.

(b) SDPA incorporates dense and sparse matrix data structures and an efficient method proposed by [7] for computing search directions when the problem is large scale and sparse.

When the problem is dense, SDPA is a few times faster than SDPT3 mainly because of the reason (a). The feature (b) of SDPA is crucial in solving large scale sparse problems. It saves not only much memory, but also much computation time. Without such a sparsity consideration, it would be difficult to solve large scale sparse semidefinite programs arising from semidefinite relaxation of combinatorial optimization problems. We will also observe through the numerical results that SDPA is as stable and efficient (measured in the number of iterations) as SDPT3 for such small and medium scale semidefinite programs that SDPT3 can solve within a reasonable time.

Let $R^{n\times n}$ and $\mathcal{S}^n \subset R^{n\times n}$ denote the set of all $n \times n$ real matrices and the set of all $n \times n$ real symmetric matrices, respectively. We use the notation $\boldsymbol{U} \bullet \boldsymbol{V}$ for the inner product of $\boldsymbol{U},\ \boldsymbol{V} \in R^{n\times n}$, *i.e.,* $\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{i=1}^n \sum_{j=1}^n U_{ij}V_{ij}$, where U_{ij} and V_{ij} denote the (i,j)th element of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. We write $\boldsymbol{X} \succeq \boldsymbol{O}$ and $\boldsymbol{X} \succ \boldsymbol{O}$ when $\boldsymbol{X} \in \mathcal{S}^n$ is positive semidefinite and positive definite, respectively.

Let $\boldsymbol{A}_i \in \mathcal{S}^n$ $(0 \le i \le m)$ and $b_i \in R$ $(1 \le i \le m)$. SDPA solves the standard form semidefinite program and its dual:

$$\left.\begin{array}{llll} \mathcal{P}: & \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} & \text{subject to} \quad \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i \ (1 \le i \le m),\ \boldsymbol{X} \succeq \boldsymbol{O}. \\ \mathcal{D}: & \text{maximize} & \displaystyle\sum_{i=1}^m b_i y_i & \text{subject to} \quad \displaystyle\sum_{i=1}^m \boldsymbol{A}_i y_i + \boldsymbol{Z} = \boldsymbol{A}_0,\ \boldsymbol{Z} \succeq \boldsymbol{O}. \end{array}\right\} \tag{1.1}$$

For simplicity, we say that $(\boldsymbol{X},\boldsymbol{y},\boldsymbol{Z})$ is a feasible solution (an interior-feasible solution, or an optimal solution, respectively) of the SDP (1.1) if $\boldsymbol{X}$ is a feasible solution (an interior-feasible solution, *i.e.,* a feasible solution satisfying $\boldsymbol{X} \succ \boldsymbol{O}$ or a minimizing solution, respectively) of $\mathcal{P}$ and $(\boldsymbol{y},\boldsymbol{Z})$ is a feasible solution (an interior-feasible solution, *i.e.,* a feasible solution satisfying $\boldsymbol{Z} \succ \boldsymbol{O}$ or a maximizing solution, respectively) of $\mathcal{D}$.

In Section 2, we present some issues on the implementation of SDPA which are relevant for our numerical experiments. Section 3 is devoted to seven kinds of test problems, and Section 4 to preliminary numerical experiments for deciding the target software and the search direction we employ. In Section 5, we present numerical results and their analyses on those test problems solved by SDPA and SDPT3. We report the total CPU time and the number of iterations required to attain a given accuracy ϵ^*. We also investigate each individual CPU time for major time-consuming parts in SDPA such as computation of an $m\times m$ dense symmetric matrix $\boldsymbol{B}$ induced from the Newton equation for search directions, LDL^T factorization of $\boldsymbol{B}$, and approximation of the minimum eigenvalues of some matrices used for computing step lengths. We will see that the most time consuming part deeply depends on the size n of the variable matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$, the number m of equalities of $\mathcal{P}$ and the sparsity of data matrices $\boldsymbol{A}_i$ $(0 \le i \le m)$.

1.2 SOME ISSUES OF THE IMPLEMENTATION OF SDPA.

We describe the HRVW/KSH/M search direction in Section 2.1, and an efficient method for approximating the minimum eigenvalue of a symmetric matrix in Section 2.2. Next, we explain the algorithmic framework of SDPA in Section 2.3 and, finally, the technical details about infeasibility and unboundedness in Section 2.4.

1.2.1 Search Direction.

The HRVW/KSH/M direction at the current iterate $(\boldsymbol{X},\boldsymbol{y},\boldsymbol{Z})$ is the solution $(d\boldsymbol{X},d\boldsymbol{y},d\boldsymbol{Z})$ of the system of equations

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \le i \le m),\ d\boldsymbol{X} \in \mathcal{S}^n, \tag{1.2}$$

$$\sum_{i=1}^m \boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D},\ d\boldsymbol{Z} \in \mathcal{S}^n, \tag{1.3}$$

$$\widehat{d\boldsymbol{X}}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z} = \boldsymbol{K},\ \widehat{d\boldsymbol{X}} \in R^{n\times n},\ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^T)/2. \tag{1.4}$$

Here

$$\begin{aligned} p_i &= b_i - \boldsymbol{A}_i \bullet \boldsymbol{X} \ (1 \leq i \leq m), \\ \boldsymbol{D} &= \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i y_i - \boldsymbol{Z}, \end{aligned}$$

and $\boldsymbol{K}$ denotes an $n \times n$ constant matrix which is specified later in Section 2.3. Note that $\widehat{d\boldsymbol{X}} \in R^{n \times n}$ serves as an auxiliary variable matrix. Under the linear independence assumption on the set $\{\boldsymbol{A}_i : 1 \leq i \leq m\}$ of constraint matrices, we know [14] that for any $\boldsymbol{X} \succ \boldsymbol{O}$, $\boldsymbol{Z} \succ \boldsymbol{O}$ and $\boldsymbol{K} \in R^{n \times n}$, the system of equations (1.2), (1.3) and (1.4) has a unique solution $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$.

We can reduce the system of equations (1.2), (1.3) and (1.4) to

$$\boldsymbol{B} d\boldsymbol{y} = \boldsymbol{g}, \tag{1.5}$$

$$\left.\begin{aligned} &d\boldsymbol{Z} = \boldsymbol{D} - \sum_{i=1}^{m} \boldsymbol{A}_i dy_i, \\ &\widehat{d\boldsymbol{X}} = (\boldsymbol{K} - \boldsymbol{X} d\boldsymbol{Z})\, \boldsymbol{Z}^{-1}, \ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^T)/2, \end{aligned}\right\} \tag{1.6}$$

where

$$\left.\begin{aligned} &B_{ij} = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j \ (1 \leq i \leq m, \ 1 \leq j \leq m) \\ &g_i = p_i - (\boldsymbol{K} - \boldsymbol{X}\boldsymbol{D})\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i \ (1 \leq i \leq m). \end{aligned}\right\} \tag{1.7}$$

The matrices $\boldsymbol{X}$, $\boldsymbol{Z}^{-1}$ and $\boldsymbol{B}$ are symmetric and dense in general even when all $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ are sparse. Hence solving the system of equations (1.5) in $d\boldsymbol{y}$ by using a direct method such as the Cholesky factorization and the LDL^T factorization of $\boldsymbol{B}$ requires $O(m^3)$ arithmetic operations. On the other hand, if we treat all $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ as dense matrices and if we use the above formulae (1.7) for the coefficient matrix $\boldsymbol{B}$ in a straightforward way, the computation of $\boldsymbol{B}$ requires $O(mn^3 + m^2n^2)$ arithmetic operations. Therefore computing the coefficient matrix $\boldsymbol{B}$ is more crucial than solving $\boldsymbol{B}d\boldsymbol{y} = \boldsymbol{g}$ in the entire computation of the HRVW/KSH/M direction.

In their paper [7], Fujisawa, Kojima and Nakata proposed three distinct formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ for computing $\boldsymbol{B}$, and their efficient combination $\mathcal{F}_*$. They demonstrated through numerical experiments that the combined formula $\mathcal{F}_*$ worked very efficiently when some of $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ are sparse. We incorporated their formula $\mathcal{F}_*$ into SDPA. See the paper [7] for more details.

1.2.2 Approximation of the Minimum Eigenvalue of a Symmetric Matrix.

We need to compute the minimum eigenvalue of a symmetric matrix when we compute the primal and dual step lengths in SDPA. Let $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ denote a current iterate satisfying $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Z} \succ \boldsymbol{O}$, and $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ be a search direction. Then the primal step length α_p is computed as follows:

$$\bar{\alpha}_p = \max\{\alpha \in [0,1] : \boldsymbol{X} + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}\} \ \text{ and } \alpha_p = \gamma \bar{\alpha}_p,$$

where $\gamma \in (0, 1)$ denotes a prescribed parameter. Applying the Cholesky factorization to the positive definite matrix $\boldsymbol{X}$, we have an $n \times n$ lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$. Then we can rewrite $\bar{\alpha}_p$ as

$$\begin{aligned} \bar{\alpha}_p &= \max\{\alpha \in [0,1] : \boldsymbol{I} + \alpha \boldsymbol{L}^{-1} d\boldsymbol{X} \boldsymbol{L}^{-T} \succeq \boldsymbol{O}\}, \\ &= \begin{cases} \min\{-1/\xi_{\min}, \ 1\} & \text{if } \xi_{\min} < 0, \\ 1 & \text{otherwise,} \end{cases} \end{aligned}$$

where $\xi_{\min}$ denotes the minimum eigenvalue of $\boldsymbol{L}^{-1} d\boldsymbol{X} \boldsymbol{L}^{-T}$, which coincides with the minimum eigenvalue of $\boldsymbol{X}^{-1} d\boldsymbol{X}$. The dual step length α_d is computed similarly;

$$\begin{aligned} \bar{\alpha}_d &= \begin{cases} \min\{-1/\eta_{\min}, \ 1\} & \text{if } \eta_{\min} < 0, \\ 1 & \text{otherwise,} \end{cases}, \\ \alpha_d &= \gamma \bar{\alpha}_d, \end{aligned}$$

where $\eta_{\min}$ denotes the minimum eigenvalue of $\boldsymbol{M}^{-1} d\boldsymbol{Z} \boldsymbol{M}^{-T}$, and $\boldsymbol{Z} = \boldsymbol{M}\boldsymbol{M}^T$ denotes the Cholesky factorization of $\boldsymbol{Z}$.

In the remainder of this section, we present an efficient method for approximating the minimum eigenvalue of an $n \times n$ symmetric matrix $\boldsymbol{U} \in \mathcal{S}^n$. Our method is known as the bisection method for finding eigenvalues of a symmetric tridiagonal matrix. Applying the Householder tridiagonalization to $\boldsymbol{U}$, we first transform $\boldsymbol{U}$ to a symmetric tridiagonal matrix $\boldsymbol{T}$ having the same eigenvalues as $\boldsymbol{U}$ (see, for example, [9]). Let $\boldsymbol{T}_r$ denote the leading $r \times r$ principal submatrix of $\boldsymbol{T}$;

$$\boldsymbol{T}_r = \begin{pmatrix} a_1 & b_2 & & & \dots & 0 \\ b_2 & a_2 & \ddots & & & \vdots \\ & \ddots & \ddots & \ddots & & \\ \vdots & & \ddots & a_{r-1} & b_r \\ 0 & \dots & & b_r & a_r \end{pmatrix}.$$

By definition, $\boldsymbol{T} = \boldsymbol{T}_n$. Assume that some b_r $(2 \le r \le n)$ vanishes. In this case, if we rewrite $\boldsymbol{T}$ as

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{T}_{r-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix},$$

then $\lambda_{\min}(\boldsymbol{T}) = \min\{\lambda_{\min}(\boldsymbol{T}_{r-1}),\ \lambda_{\min}(\boldsymbol{S})\}$. We note that either of $\boldsymbol{T}_{r-1}$ and $\boldsymbol{S}$ is a tridiagonal symmetric matrix with a size smaller than the size n of $\boldsymbol{T}$. Therefore we will assume below that no b_r $(2 \le r \le n)$ vanishes.

Now we define the polynomials $p_0(\lambda) = 1$ and $p_r(\lambda) = \det(\boldsymbol{T}_r - \lambda \boldsymbol{I})$ $(1 \le r \le n)$. Then the minimum eigenvalue $\lambda_{\min}(\boldsymbol{T})$ of $\boldsymbol{T}$, which we want to compute, corresponds to the minimum root of the polynomial $p_n(\lambda)$. We can compute $p_r(\lambda) = \det(\boldsymbol{T}_r - \lambda \boldsymbol{I})$ by the following recursive formulae:

$$\begin{aligned} p_0(\lambda) &= 1,\ p_1(\lambda) = a_1 - \lambda \\ p_r(\lambda) &= (a_r - \lambda)p_{r-1}(\lambda) - b_r^2 p_{r-2}(\lambda) \quad (2 \le r \le n). \end{aligned}$$

Theorem 1.2.1. Sturm Sequence Property [9, Theorem 8.4-1] *Assume that* $b_r \neq 0$ $(2 \le r \le n)$.

(i) If $p_r(\lambda) = 0$ *for some* $r \in \{1, 2, \dots, n-1\}$*, then* $p_{r+1}(\lambda)p_{r-1}(\lambda) < 0$

(ii) Let $\lambda \in R$*. Let* $\sigma(\lambda)$ *denote the number of sign changes in the sequence*

$$\{p_0(\lambda), p_1(\lambda), \dots, p_n(\lambda)\}.$$

Then $\sigma(\lambda)$ *equals the number of* $\boldsymbol{T}$*'s eigenvalues that are less than* λ*. Here we assume that* $p_r(\lambda)$ *has the opposite sign from* $p_{r-1}(\lambda)$ *if* $p_{r-1}(\lambda) \neq 0$ *and* $p_r(\lambda)p_{r-1}(\lambda) \le 0$ *and that* $p_r(\lambda)$ *has the same sign as* $p_{r-1}(\lambda)$ *if* $p_{r-1}(\lambda) = 0$ *or* $p_r(\lambda)p_{r-1}(\lambda) > 0$.

From the Gerschgorin circle theorem (see, for example, [9, Theorem 7.2-1]), we also know that $\lambda_{min}(\boldsymbol{T}) \in [\underline{\lambda}, \overline{\lambda}]$ where

$$\underline{\lambda} = \min_{1 \le i \le n} a_i - |b_i| - |b_{i+1}| \ \text{ and } \overline{\lambda} = \max_{1 \le i \le n} a_i + |b_i| + |b_{i+1}|. \tag{1.8}$$

Here we assume that $b_1 = b_{n+1} = 0$.

Let $\underline{\lambda} < \lambda < \overline{\lambda}$. Recall that $p_0(\lambda) = 1 > 0$. Hence, by the theorem above, if

$$p_k(\lambda) > 0 \ (0 \le k < r) \ \text{ and } p_r(\lambda) \le 0 \tag{1.9}$$

for some $r \in \{1, 2, \dots, n\}$, then we know that $\lambda_{\min}(\boldsymbol{T}) \in [\underline{\lambda}, \lambda]$, and otherwise $\lambda_{\min}(\boldsymbol{T}) \in [\lambda, \overline{\lambda}]$.

Summarizing the discussions above, we now present the bisection method to approximate the minimum eigenvalue $\lambda_{\min}(\boldsymbol{T})$.

Bisection method for finding an approximate value of $\lambda_{min}(\boldsymbol{T})$:

Step 0: Define $\underline{\lambda}$ and $\overline{\lambda}$ by (1.8). Let $\epsilon > 0$.

Step 1: If $|\overline{\lambda} - \underline{\lambda}| < \epsilon(|\underline{\lambda}| + |\overline{\lambda}|)$, then stop the iteration and output $\underline{\lambda}$.

Step 2: Let $\lambda = (\underline{\lambda} + \overline{\lambda})/2$, $r = 1$ and $p_1(\lambda) = a_1 - \lambda$.

Step 3: If $p_r(\lambda) \leq 0$, then $\overline{\lambda} = \lambda$ and go to Step 1.

Step 4: $r = r + 1$.

Step 5: If $r > n$, then $\underline{\lambda} = \lambda$ and go to Step 1.

Step 6: Let $p_r(\lambda) = (a_r - \lambda)p_{r-1}(\lambda) - b_r^2 p_{r-2}(\lambda)$ and go to Step 3.

Remark 1.2.2. In practice, generating the sequence $\{p_r(\lambda) : r = 0, 1, 2, \ldots, n\}$ often yields numerical instability, *i.e.*, an overflow of the numerical value $p_r(\lambda)$ for larger r. SDPA employs the sequence $\{q_r(\lambda) : r = 1, 2, \ldots, n\}$ defined below instead of the sequence $\{p_r(\lambda) : r = 0, 1, 2, \ldots, n\}$. For every $r = 1, 2, \ldots, n$ and $\lambda \in R$, let

$$q_r(\lambda) \quad = \quad \begin{cases} p_r(\lambda)/p_{r-1}(\lambda) & \text{if } p_{r-1}(\lambda) \neq 0, \\ +\infty & \text{otherwise} \end{cases}$$

For every $\lambda \in R$, we can compute $q_r(\lambda)$ $(1 \leq r \leq n)$ recursively as

$$q_r(\lambda) = a_r - \lambda - b_r^2 / q_{r-1}(\lambda)$$

as long as $q_{r-1}(\lambda) \neq 0$. By definition, we see that (1.9) holds if and only if

$$q_k(\lambda) > 0 \ (0 \leq k < r) \ \text{ and } q_r(\lambda) \leq 0 \tag{1.10}$$

hold. Therefore we can consistently utilize the sequence $\{q_r(\lambda) : r = 1, 2, \ldots, n\}$, which is more stable numerically, instead of the sequence $\{p_r(\lambda) : r = 0, 1, 2, \ldots, n\}$ if we replace Step 6 by

Step 6': Let $q_r(\lambda) = (a_r - \lambda) - b_r^2 / q_{r-1}(\lambda)$ and go to Step 3.

Remark 1.2.3. There are different methods for computing the minimum eigenvalue of a real symmetric matrix, for example, the bisection method, the power method and so on. The computation of the minimum eigenvalue employing the bisection method takes $\mathcal{O}(n^3)$ arithmetic operations because the Householder tridiagonalization is required. Undoubtedly, the bisection method is not the bottleneck in the eigenvalue computation at all, if we only need the minimum eigenvalue. According to our numerical experiments, the Householder tridiagonalization of $\boldsymbol{L}^{-1} d\boldsymbol{X} \boldsymbol{L}^{-T} \in \mathcal{S}^{n \times n}$ $(n = 300)$ in Section 2.2 takes about 129.95 seconds, but the bisection method only takes about 0.72 seconds. We observed, however, the following tendency. The power method for finding the minimum eigenvalue is an approximative iterative method. The required number of iterations until the power method converges strongly depends on the given accuracy ϵ^* and the problem to be solved. If more than $\Omega(n)$ iterations is required, the power method costs more computation time than the bisection method because the power method executes $\mathcal{O}(n^2)$ arithmetic operations per iteration. In fact, for the same example above $(n = 300)$, our numerical experiments showed that the power method takes about 300.23 seconds to find the same solution with less accuracy. Therefore, judging from the efficiency and stability of the algorithm, the bisection method is one of the recommended methods for computing the minimum eigenvalue.

1.2.3 The Algorithmic Framework of SDPA.

Step 0 : Set an initial point $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ with $\boldsymbol{X}^0 \succ \boldsymbol{O}, \boldsymbol{Z}^0 \succ \boldsymbol{O}$, and a positive integer k^* for the number of maximum iterations. Decide on the search direction to use. Set the parameters: $0.0 < \epsilon^*$, $0.80 \leq \gamma \leq 0.98$, $0.01 \leq \beta^* \leq 0.10$ and $\beta^* \leq \bar{\beta} \leq 0.20$. (The default values of these parameters are: $k^* = 50$, $\epsilon^* = 10^{-5}$, $\gamma = 0.95$, $\beta^* = 0.05$ and $\bar{\beta} = 0.1$). Let $k = 0$.

Step 1 : If the current iterate $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) = (\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is feasible and the relative gap

$$\frac{|P - D|}{\max\{1.0, \ (|P| + |D|)/2\}}$$

gets smaller than ϵ^*, then stop the iteration. Here P and D denote the primal and the dual objective values, respectively. In this case, $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ gives an approximate optimal solution of the SDP (1.1).

If $k \geq k^*$ then stop the iteration. If $\mathcal{P}$ or $\mathcal{D}$ of the SDP (1.1) is likely to be infeasible or unbounded, then stop the iteration. More details of this part will be described in Section 2.4.

Step 2 : (Predictor Step) Let

$$\beta_p = \begin{cases} 0 & \text{if the current iterate is feasible,} \\ \bar{\beta} & \text{otherwise.} \end{cases}$$

Solve the system of equations (1.2), (1.3) and (1.4) with $\boldsymbol{K} = \beta_p(\boldsymbol{X} \bullet \boldsymbol{Z}/n)\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}$ to compute the predictor direction $(d\boldsymbol{X}_p, d\boldsymbol{y}_p, d\boldsymbol{Z}_p)$.

Step 3 : (Corrector Step) Let

$$\beta = \frac{(\boldsymbol{X} + \bar{\alpha}_p d\boldsymbol{X}_p) \bullet (\boldsymbol{Z} + \bar{\alpha}_d d\boldsymbol{Z}_p)}{(\boldsymbol{X} \bullet \boldsymbol{Z})},$$

where

$$\bar{\alpha}_p = \max\{\alpha \in [0,1] : \boldsymbol{X} + \alpha d\boldsymbol{X}_p \succeq \boldsymbol{O}\},$$
$$\bar{\alpha}_d = \max\{\alpha \in [0,1] : \boldsymbol{Z} + \alpha d\boldsymbol{Z}_p \succeq \boldsymbol{O}\}.$$

See Section 2.2 for more details. Choose the parameter β_c as follows:

$$\beta_c = \begin{cases} \max\{\beta^*, \beta^2\} & \text{if the current iterate is feasible and } \beta \leq 1.0, \\ \max\{\bar{\beta}, \beta^2\} & \text{if the current iterate is infeasible and } \beta \leq 1.0, \\ 1.0 & \text{otherwise.} \end{cases}$$

Compute the combined predictor-corrector direction $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ by solving the system of equations (1.2), (1.3) and (1.4) with $\boldsymbol{K} = \beta_c(\boldsymbol{X} \bullet \boldsymbol{Z}/n)\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z} - d\boldsymbol{X}_p d\boldsymbol{Z}_p$.

Step 4 : Set the next iterate $(\boldsymbol{X}^{k+1}, \boldsymbol{y}^{k+1}, \boldsymbol{Z}^{k+1})$ such that

$$\begin{aligned} \boldsymbol{X}^{k+1} &= \boldsymbol{X}^k + \gamma\bar{\alpha}_p d\boldsymbol{X} \quad \text{and} \\ (\boldsymbol{y}^{k+1}, \boldsymbol{Z}^{k+1}) &= (\boldsymbol{y}^k, \boldsymbol{Z}^k) + \gamma\bar{\alpha}_d(d\boldsymbol{y}, d\boldsymbol{Z}), \end{aligned}$$

where

$$\bar{\alpha}_p = \max\{\alpha \in [0,1] : \boldsymbol{X} + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}\},$$
$$\bar{\alpha}_d = \max\{\alpha \in [0,1] : \boldsymbol{Z} + \alpha d\boldsymbol{Z} \succeq \boldsymbol{O}\}.$$

See Section 2.2 for more details.

Step 5 : $k \longleftarrow k + 1$ and go to Step 1.

1.2.4 Getting Information on Infeasibility and Unboundedness.

In additions to the parameters ϵ^*, γ, β^* and $\bar{\beta}$, we introduce three parameters $\ell^* \in R$, $u^* \in R$ and $\omega^* > 1$ at Step 0 of the algorithmic framework of SDPA; the default values of these parameters are: $\ell^* = -10^5$, $u^* = 10^5$ and $\omega^* = 2.0$. We also set $\eta_p = \eta_d = +\infty$ at Step 0.

First we describe how SDPA guesses the unboundedness of the primal problem $\mathcal{P}$ or the dual problem $\mathcal{D}$ of the SDP (1.1).

(a) If the current iterate $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is primal-feasible and $\boldsymbol{A}_0 \bullet \boldsymbol{X}^k \leq \ell^*$, then SDPA regards that $\mathcal{P}$ is likely to be unbounded; hence the dual problem $\mathcal{D}$ is likely to be infeasible.

(b) If the current iterate $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is dual-feasible and $\sum_{i=1}^{m} b_i y_i^k \geq u^*$, then SDPA regards that $\mathcal{D}$ is likely to be unbounded; hence the primal problem $\mathcal{P}$ is likely to be infeasible.

Now we show how SDPA guesses the infeasibility of the SDP (1.1). The discussion below is based on the paper [14]. We can prove that each iterate $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ satisfies

$$\left.\begin{array}{l} \boldsymbol{X}^k \succ \boldsymbol{O},\ \boldsymbol{Z}^k \succ \boldsymbol{O}, \\ p_i^k \equiv b_i - \boldsymbol{A}_i \bullet \boldsymbol{X}^k = \theta_p^k p_i^0 \ (1 \leq i \leq m), \\ \boldsymbol{D}^k \equiv \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i y_i^k - \boldsymbol{Z}^k = \theta_d^k \boldsymbol{D}^0, \end{array}\right\} \tag{1.11}$$

where

$$
\begin{aligned}
&1 = \theta_p^0 \geq \theta_p^k \geq \theta_p^{k+1} \geq 0 \ (k = 1, 2, \dots),\\
&1 = \theta_d^0 \geq \theta_d^k \geq \theta_d^{k+1} \geq 0 \ (k = 1, 2, \dots),\\
&p_i^0 = b_i - \boldsymbol{A}_i \bullet \boldsymbol{X}^0 \ (1 \leq i \leq m) \text{ (the initial primal residuals)},\\
&\boldsymbol{D}^0 = \boldsymbol{A}_0 - \sum_{i=1}^m \boldsymbol{A}_i y_i^0 - \boldsymbol{Z}^0 \text{ (the initial dual residual matrix)}.
\end{aligned}
$$

For $k = 0$, (1.11) follows directly from the definitions of the initial primal residual vector $\boldsymbol{p}^0 = (p_1^0, p_2^0, \dots, p_m^0)^T \in R^m$ and the initial dual residual matrix $\boldsymbol{D}^0 \in \mathcal{S}^n$. For $k > 0$, (1.11) means that the primal residual vector $\boldsymbol{p}^k = (p_1^k, p_2^k, \dots, p_m^k)^T \in R^m$ lies on the line segment $\{\theta \boldsymbol{p}^0 : \theta \in [0, 1]\}$, and the dual residual matrix $\boldsymbol{D}^k \in \mathcal{S}^n$ on the line segment $\{\theta \boldsymbol{D}^0 : \theta \in [0, 1]\}$. We also see that the current iterate $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is primal-feasible if and only if $\theta_p^k = 0$, and dual-feasible if and only if $\theta_d^k = 0$. At Step 1, we update η_p and η_d as follows:

$$
\begin{aligned}
\eta_p &= \min\{\eta_p,\ \boldsymbol{X}^k \bullet \boldsymbol{Z}^0\} \quad \text{if } (\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k) \text{ is primal-feasible},\\
\eta_d &= \min\{\eta_d,\ \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k\} \quad \text{if } (\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k) \text{ is dual-feasible}.
\end{aligned}
$$

Then we obtain the following:

(c) If $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is dual-feasible and

$$
1 < \rho_p \equiv \frac{\theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{(\theta_p^k + \omega^*(1 - \theta_p^k))\eta_d + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k}, \tag{1.12}
$$

then there is no feasible solution $\boldsymbol{X}$ of the primal problem $\mathcal{P}$ such that

$$
\omega^* \boldsymbol{X}^0 \succeq \boldsymbol{X} \succeq \boldsymbol{O}.
$$

(d) If $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is primal-feasible and

$$
1 < \rho_d \equiv \frac{\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0}{(\theta_d^k + \omega^*(1 - \theta_d^k))\eta_p + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k},
$$

then there is no feasible solution $(\boldsymbol{y}, \boldsymbol{Z})$ of the dual problem $\mathcal{D}$ such that

$$
\omega^* \boldsymbol{Z}^0 \succeq \boldsymbol{Z} \succeq \boldsymbol{O}.
$$

(e) If $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ is primal-infeasible, dual-infeasible, and

$$
1 < \rho_{pd} \equiv \frac{\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0 + \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\left(\theta_p^k \theta_d^k + \omega^*(\theta_p^k(1 - \theta_d^k) + (1 - \theta_p^k)\theta_d^k)\right) \boldsymbol{X}^0 \bullet \boldsymbol{Z}^0 + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k}, \tag{1.13}
$$

then there is no optimal solution $(\boldsymbol{X}^*, \boldsymbol{y}^*, \boldsymbol{Z}^*)$ of the SDP (1.1) such that

$$
\left.
\begin{aligned}
&\omega^* \boldsymbol{X}^0 \succeq \boldsymbol{X}^* \succeq \boldsymbol{O},\ \omega^* \boldsymbol{Z}^0 \succeq \boldsymbol{Z}^* \succeq \boldsymbol{O},\\
&\boldsymbol{X}^* \bullet \boldsymbol{Z}^* = \boldsymbol{A}_0 \bullet \boldsymbol{X}^* - \sum_{i=1}^m b_i y_i^* = 0 \text{ (no duality gap)}.
\end{aligned}
\right\} \tag{1.14}
$$

The remainder of this section is devoted to the proofs of assertions (c), (d) and (e) above. First we will show that the inequality

$$
\begin{aligned}
&\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0 + \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k\\
&\quad\leq\ \theta_d^k \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^0 + \theta_d^k (1 - \theta_p^k) \bar{\boldsymbol{X}} \bullet \boldsymbol{Z}^0\\
&\qquad + (1 - \theta_d^k)\theta_p^k \boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + (1 - \theta_d^k)(1 - \theta_p^k) \bar{\boldsymbol{X}} \bullet \bar{\boldsymbol{Z}} + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k
\end{aligned} \tag{1.15}
$$

holds for any feasible solution $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{Z}})$ of the SDP (1.1). Let $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{Z}})$ be a feasible solution of the SDP (1.1). Let

$$\boldsymbol{X}' = \theta_p^k \boldsymbol{X}^0 + (1-\theta_p^k)\bar{\boldsymbol{X}} \quad \text{and } (\boldsymbol{y}', \boldsymbol{Z}') = \theta_d^k(\boldsymbol{y}^0, \boldsymbol{Z}^0) + (1-\theta_d^k)(\bar{\boldsymbol{y}}, \bar{\boldsymbol{Z}}).$$

Then $(\boldsymbol{X}^k, \boldsymbol{y}^k, \boldsymbol{Z}^k)$ and $(\boldsymbol{X}', \boldsymbol{y}', \boldsymbol{Z}')$ satisfy

$$\begin{aligned} b_i - \boldsymbol{A}_i \bullet \boldsymbol{X}^k = b_i - \boldsymbol{A}_i \bullet \boldsymbol{X}' = \theta_p^k p_i^0 \ (i = 1, 2, \ldots, m), \\ \boldsymbol{A}_0 - \sum_{i=1}^m \boldsymbol{A}_i y_i^k - \boldsymbol{Z}^k = \boldsymbol{A}_0 - \sum_{i=1}^m \boldsymbol{A}_i y_i' - \boldsymbol{Z}' = \theta_d^k \boldsymbol{D}^0. \end{aligned}$$

Hence

$$\begin{aligned} 0 &= (\boldsymbol{X}' - \boldsymbol{X}^k) \bullet (\boldsymbol{Z}' - \boldsymbol{Z}^k) \\ &= \left(\theta_p^k \boldsymbol{X}^0 + (1-\theta_p^k)\bar{\boldsymbol{X}} - \boldsymbol{X}^k\right) \bullet \left(\theta_d^k \boldsymbol{Z}^0 + (1-\theta_d^k)\bar{\boldsymbol{Z}} - \boldsymbol{Z}^k\right) \\ &= \left(\theta_p^k \boldsymbol{X}^0 + (1-\theta_p^k)\bar{\boldsymbol{X}}\right) \bullet \left(\theta_d^k \boldsymbol{Z}^0 + (1-\theta_d^k)\bar{\boldsymbol{Z}}\right) \\ &\quad -\boldsymbol{X}^k \bullet \left(\theta_d^k \boldsymbol{Z}^0 + (1-\theta_d^k)\bar{\boldsymbol{Z}}\right) - \left(\theta_p^k \boldsymbol{X}^0 + (1-\theta_p^k)\bar{\boldsymbol{X}}\right) \bullet \boldsymbol{Z}^k + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k \\ &\le \theta_d^k \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^0 + \theta_d^k (1-\theta_p^k)\bar{\boldsymbol{X}} \bullet \boldsymbol{Z}^0 \\ &\quad +(1-\theta_d^k)\theta_p^k \boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + (1-\theta_d^k)(1-\theta_p^k)\bar{\boldsymbol{X}} \bullet \bar{\boldsymbol{Z}} \\ &\quad -\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0 - \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k. \end{aligned}$$

Thus (1.15) follows.

Now we are ready to derive assertion (c). By assumption, we know that $\theta_p^k > 0, \theta_d^k = 0$ and $\eta_d = \boldsymbol{X}^0 \bullet \boldsymbol{Z}^r < \infty$ for some dual feasible $(\boldsymbol{X}^r, \boldsymbol{y}^r, \boldsymbol{Z}^r)$ $(0 \le r \le k)$. Assume on the contrary that there is a feasible solution $\boldsymbol{X}$ of the primal problem $\mathcal{P}$ such that $\omega^* \boldsymbol{X}^0 \succeq \boldsymbol{X} \succeq \boldsymbol{O}$. Then, letting $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{Z}}) = (\boldsymbol{X}, \boldsymbol{y}^r, \boldsymbol{Z}^r)$, we have a feasible solution of the SDP (1.1). It follows from (1.15) that

$$\begin{aligned} 1 &\ge \frac{\theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\theta_p^k \boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + (1-\theta_p^k)\bar{\boldsymbol{X}} \bullet \bar{\boldsymbol{Z}} + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k} \\ &\ge \frac{\theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\theta_p^k \boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + \omega^*(1-\theta_p^k)\boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k} \\ &= \frac{\theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\left(\theta_p^k + \omega^*(1-\theta_p^k)\right)\eta_d + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k} \\ &= \rho_p \end{aligned}$$

This contradicts to assumption (1.12). Thus we have shown assertion (c).

We can prove assertion (d) similarly.

Finally we prove assertion (e). In this case, we have that $\theta_d^k > 0$ and $\theta_p^k > 0$. Assume on the contrary that there is an optimal solution $(\boldsymbol{X}^*, \boldsymbol{y}^*, \boldsymbol{Z}^*)$ satisfying the conditions in (1.14). Let $(\bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{Z}}) = (\boldsymbol{X}^*, \boldsymbol{y}^*, \boldsymbol{Z}^*)$. Then $\bar{\boldsymbol{X}} \bullet \bar{\boldsymbol{Z}} = 0$ and inequality (1.15) holds. Hence

$$\begin{aligned} 1 &\ge \frac{\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0 + \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\theta_d^k \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^0 + \theta_d^k(1-\theta_p^k)\bar{\boldsymbol{X}} \bullet \boldsymbol{Z}^0 + (1-\theta_d^k)\theta_p^k \boldsymbol{X}^0 \bullet \bar{\boldsymbol{Z}} + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k} \\ &\ge \frac{\theta_d^k \boldsymbol{X}^k \bullet \boldsymbol{Z}^0 + \theta_p^k \boldsymbol{X}^0 \bullet \boldsymbol{Z}^k}{\left(\theta_d^k \theta_p^k + \omega^* \theta_d^k (1-\theta_p^k) + \omega^*(1-\theta_d^k)\theta_p^k\right) \boldsymbol{X}^0 \bullet \boldsymbol{Z}^0 + \boldsymbol{X}^k \bullet \boldsymbol{Z}^k} \\ &= \rho_{pd}. \end{aligned}$$

This contradicts to assumption (1.13), and we have shown assertion (e).

1.3 TEST PROBLEMS.

We present 7 types of semidefinite programs in this section, and we show numerical results on those problems in the next two sections.

1.3.1 Randomly Generated Dense Semidefinite Program.

The first example [23] is the standard form SDP (1.1) with dense data matrices $\boldsymbol{A}_i \in \mathcal{S}^n$ $(1 \leq i \leq m)$. Using the standard normal distribution $\mathcal{N}(0,1)$, we generate each element of $\boldsymbol{A}_i$ $(1 \leq i \leq m)$. Then we choose $\boldsymbol{A}_0 \in \mathcal{S}^n$ and $\boldsymbol{b} \in R^m$ so that the SDP (1.1) has an interior feasible solution.

1.3.2 Norm Minimization Problem.

Let $\boldsymbol{F}_i \in R^{q\times r}$ $(0 \leq i \leq p)$. The norm minimization problem [23] is defined as:

$$\begin{array}{ll} \text{minimize} & \left\| \boldsymbol{F}_0 + \sum_{i=1}^{p} \boldsymbol{F}_i y_i \right\| \\ \text{subject to} & y_i \in R \ (1 \leq i \leq p). \end{array}$$

Here $\|\boldsymbol{C}\|$ denotes the 2-norm of $\boldsymbol{C}$, *i.e.*, $\|\boldsymbol{C}\| = \max_{\|\boldsymbol{u}\|=1} \|\boldsymbol{C}\boldsymbol{u}\|$ = the square root of the maximum eigenvalue of $\boldsymbol{C}^T\boldsymbol{C}$. We can reduce this problem to an SDP:

$$\begin{array}{ll} \text{maximize} & -y_{p+1} \\ \text{subject to} & \sum_{i=1}^{p} \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_i^T \\ \boldsymbol{F}_i & \boldsymbol{O} \end{pmatrix} y_i + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} y_{p+1} + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix} \succeq \boldsymbol{O}. \end{array}$$

Thus if we take

$$\begin{array}{rcl} m & = & p+1, \ n = r+q, \ \boldsymbol{A}_0 = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix}, \\ \boldsymbol{A}_i & = & \begin{pmatrix} \boldsymbol{O} & -\boldsymbol{F}_i^T \\ -\boldsymbol{F}_i & \boldsymbol{O} \end{pmatrix}, \ b_i = 0 \ (1 \leq i \leq p), \\ \boldsymbol{A}_{p+1} & = & \begin{pmatrix} -\boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & -\boldsymbol{I} \end{pmatrix}, \ b_{p+1} = -1, \end{array}$$

then we can reformulate the problem as the dual of the standard form SDP (1.1).

1.3.3 Chebyshev Approximation Problem for a Matrix.

This problem is a special case of the norm minimization problem above. Given a real square matrix $\boldsymbol{F} \in R^{p\times p}$, the problem is formulated as:

$$\text{minimize} \quad \left\| \boldsymbol{F}^r + \sum_{j=1}^{r} x_j \boldsymbol{F}^{j-1} \right\| \quad \text{subject to } \boldsymbol{x} \in R^r,$$

where we assume that $\boldsymbol{F}^0 = \boldsymbol{I}$. It is known that the set $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$ does not constitute a well distributed matrix basis in $R^{p\times p}$. We used an orthonormalized basis $\{\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_{r+1}\}$ instead of the set $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$ to obtain a better convergence and numerical stability. This basis is derived by a modified Gram-Schmidt orthonormalization procedure with respect to the matrix inner product from the set $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$. This was suggested in [23, Section 5] (see also [24]), and we used the MATLAB function “chebymat.m” contained in the SDPT3 package to generate such Chebyshev approximation problems for our numerical experiments.

1.3.4 Semidefinite Program Arising from Control and System Theory.

Let $\boldsymbol{P} \in R^{\ell\times\ell}$, $\boldsymbol{Q} \in R^{\ell\times k}$ and $\boldsymbol{R} \in R^{k\times\ell}$. We consider a semidefinite program of the form

$$\begin{array}{ll} \text{maximize} & \lambda \\ \text{subject to} & \begin{pmatrix} -\boldsymbol{P}^T\boldsymbol{S} - \boldsymbol{S}\boldsymbol{P} - \boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R} & -\boldsymbol{S}\boldsymbol{Q} \\ -\boldsymbol{Q}^T\boldsymbol{S} & \boldsymbol{D} \end{pmatrix} \succeq \lambda\boldsymbol{I}, \ \boldsymbol{S} \succeq \lambda\boldsymbol{I}, \end{array}$$

where the minimization is taken over the $k\times k$ diagonal matrix $\boldsymbol{D} = \mathrm{diag}(d_1, d_2, \ldots, d_k)$, the $\ell\times\ell$ symmetric matrix $\boldsymbol{S} \in \mathcal{S}^\ell$ and the real number λ. This problem arises from an investigation into the existence of an invariant ellipsoid for *the linear system with uncertain, time-varying, unity-bounded, diagonal feedback*

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{Px}(t) + \boldsymbol{Qu}(t), \boldsymbol{y}(t) = \boldsymbol{Rx}(t), |u_i(t)| \le y_i(t)\ (1 \le i \le k).$$

See [25] for more details.

We can reformulate the semidefinite program above as the dual problem of the standard form SDP (1.1) with $m = \ell(\ell+1)/2 + k + 1$ and $n = 2\ell + k$. We used randomly generated data matrices $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$ for our numerical experiments.

1.3.5 *Semidefinite Programming Relaxation of Maximum Cut Problem.*

Let $G = (V, E)$ be a complete undirected graph with a vertex set $V = \{1, 2, \ldots, n\}$ and an edge set $E = \{(i, j) : i, j \in V,\ i < j\}$. We assign a weight $C_{ij} = C_{ji}$ to each edge $(i, j) \in E$. The maximum cut problem is to find a partition (L, R) of V that maximizes the *cut* $c(L, R) = \sum_{i\in L, j\in R} C_{ij}$. Introducing a variable vector $\boldsymbol{u} \in R^n$, we can formulate the problem as a nonconvex quadratic program:

$$\text{maximize} \quad \frac{1}{2}\sum_{i<j} C_{ij}(1 - u_i u_j) \text{ subject to } \quad u_i^2 = 1\ (1 \le i \le n).$$

Here each feasible solution $\boldsymbol{u} \in R^n$ of this problem is corresponding to a cut (L, R) with $L = \{i \in V : u_i = -1\}$ and $R = \{i \in V : u_i = 1\}$. If we define $\boldsymbol{C}$ to be the $n \times n$ symmetric matrix with elements $C_{ji} = C_{ij}$ $((i, j) \in E)$ and $C_{ii} = 0$ $(1 \le i \le n)$, and the $n \times n$ symmetric matrix $\boldsymbol{A}_0 \in \mathcal{S}^n$ by $\boldsymbol{A}_0 = \mathrm{diag}(\boldsymbol{Ce}) - \boldsymbol{C}$, where $\boldsymbol{e} \in R^n$ denotes the vector of ones and $\mathrm{diag}(\boldsymbol{Ce})$ the diagonal matrix of the vector $\boldsymbol{Ce} \in R^n$, we can rewrite the quadratic program above as

$$\text{minimize} \quad -\boldsymbol{x}^T\boldsymbol{A}_0\boldsymbol{x} \text{ subject to } \quad x_i^2 = 1/4\ (1 \le i \le n).$$

If $\boldsymbol{x} \in R^n$ is a feasible solution of the latter quadratic program, then the $n \times n$ symmetric and positive semidefinite matrix $\boldsymbol{X}$ whose (i, j)th element X_{ij} is given by $X_{ij} = x_i x_j$ satisfies $\boldsymbol{A}_0 \bullet \boldsymbol{X} = \boldsymbol{x}^T\boldsymbol{A}_0\boldsymbol{x}$ and $X_{ii} = 1/4$ $(1 \le i \le n)$. This leads to the following semidefinite programming relaxation of the maximum cut problem:

$$\begin{array}{ll} \text{minimize} & -\boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4\ (1 \le i \le n),\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}$$

Here $\boldsymbol{E}_{ii}$ denotes the $n\times n$ symmetric matrix with (i, i)th element 1 and all others 0. See [11] and references therein for more details.

1.3.6 *Semidefinite Programming Relaxation of Graph Equipartition Problem.*

Let $G = (V, E)$ be a complete undirected graph with a vertex set $V = \{1, 2, \ldots, n\}$, an edge set $E = \{(i, j) : i, j \in V,\ i < j\}$ and weights $C_{ij} = C_{ji}$ $((i, j) \in E)$. We assume that n is an even number. The graph equipartition problem is to find a uniform partition (L, R) of V, *i.e.*, a partition (L, R) of V with the same cardinality $|L| = |R| = n/2$, that minimizes the cut $c(L, R) = \sum_{i\in L, j\in R} C_{ij}$. This problem is formulated as a nonconvex quadratic program:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{i<j} C_{ij}(1 - u_i u_j) \\ \text{subject to} & \left(\sum_{i=1}^n u_i\right)^2 = 0,\ u_i^2 = 1\ (1 \le i \le n). \end{array}$$

As in the maximum cut problem, we can derive a semidefinite programming relaxation of the graph equipartition problem:

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4\ (1 \le i \le n),\ \boldsymbol{E} \bullet \boldsymbol{X} = 0,\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}\right\} \quad (1.16)$$

Here $\boldsymbol{A}_0$ and $\boldsymbol{E}_{ii}$ $(1 \le i \le n)$ are the same matrices as in the previous section, and $\boldsymbol{E}$ denotes the $n \times n$ matrix with all elements 1. See [11, 12] and references therein for more details.

1.3.7 Semidefinite Programming Relaxation of Maximum Clique Problem.

Let $G = (V, E)$ be an undirected graph with a vertex set $V = \{1, 2, \ldots, n\}$ and an edge set $E \subset \{(i, j) : i, j \in V,\ i < j\}$. A subset C of V is said to be a clique of G if $(i, j) \in E$ for every pair of distinct $i,\ j \in C$ such that $i < j$. The maximum clique problem is to find a clique of maximum cardinality in G. It is well-known that the following SDP provides an upper bound (the Lovász number of a graph G) for the cardinality of the maximum clique of G.

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{E} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ij} \bullet \boldsymbol{X} = 0 \ ((i,j) \notin E), \\ & \boldsymbol{I} \bullet \boldsymbol{X} = 1,\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}\right\} \tag{1.17}$$

Here $\boldsymbol{E}$ denotes the $n \times n$ matrix of 1's and $\boldsymbol{E}_{ij}$ the $n \times n$ matrix with the (i, j)th and (j, i)th element 1/2 and all other elements 0. See [11] and references therein for more details.

Note that this semidefinite program has $m = n(n-1)/2 - |E| + 1$ equality constraints, where $|E|$ denotes the cardinality of the edge set E. Hence the number m of equality constraints can be of order n^2 and much larger than the size n of the variable matrix $\boldsymbol{X}$. On the other hand, all the constraint matrices $\boldsymbol{E}_{ij}$ $((i, j) \notin E)$ and $\boldsymbol{I}$ are sparse; $\boldsymbol{E}_{ij}$ $((i, j) \notin E)$ have only two nonzero elements, and $\boldsymbol{I}$ has n nonzero elements.

1.4 PRELIMINARY NUMERICAL EXPERIMENTS.

In this section, we compare four computer programs that solve semidefinite programs as well as the performance of the three search directions. The analysis of this section will serve as a base of justification for the choice of the software, SDPA and SDPT3, and the search direction, the HRVW/KSH/M direction, that will be employed in Section 5. We report numerical results of the 7 types of test problems presented in the previous section.

Throughout our numerical experiments, we adopted the relative gap

$$\frac{|P - D|}{\max\{1.0,\ (|P| + |D|)/2\}}$$

for the stopping criteria; when the relative gap got smaller than a prescribed accuracy ϵ^*, we stopped the iteration of SDPA, SDPT3 and CSDP. Here P and D denote the primal and the dual objective values, respectively. All numerical experiments in this section were executed on DEC Alpha Station (CPU Alpha 21164-400MHz with 512MB memory) under DIGITAL UNIX V4.0. First, we decide which search direction we employ in our numerical experiments.

Table 1.1 Performance comparison (SDPA:time) among HRVW/KSH/M, NT and AHO search directions.

Problem type	size		time(sec.)		
	n	m	KSH	NT	AHO
Random SDP	10	10	0.1	0.1	0.2
Random SDP	20	20	0.5	0.6	1.0
Random SDP	50	50	10.4	11.2	29.4
Norm minimization	100	100	149.7	154.6	471.0
Chebyshev approx.	100	100	159.3	161.6	533.0
Control and system theory	60	231	40.2	43.1	270.6
Maximum cut	100	100	10.3	16.5	242.1
Graph equipartition	100	101	11.7	20.7	375.9
Maximum clique	100	1024	172.1	173.0	3511.4

Tables 1.1, 1.2 and 1.3 show the performance comparisons among the HRVW/KSH/M, the NT and the AHO directions when we use SDPA. The required number of iterations when we employ the HRVW/KSH/M direction is almost the same as when we employ the NT direction. Furthermore, the HRVW/KSH/M direction is faster than the NT direction on all problems. We can also observe the same tendency for the

Table 1.2 Performance comparison (SDPA:iterations) among HRVW/KSH/M, NT and AHO search directions.

Problem type	size		iterations		
	n	m	KSH	NT	AHO
Random SDP	10	10	15	15	14
Random SDP	20	20	14	14	14
Random SDP	50	50	14	14	14
Norm minimization	100	100	17	17	16
Chebyshev approx.	100	100	18	18	18
Control and system theory	60	231	35	34	50
Maximum cut	100	100	15	15	14
Graph equipartition	100	101	15	17	21
Maximum clique	100	1024	17	17	17

Table 1.3 Performance comparison (SDPA:relative gap) among HRVW/KSH/M, NT and AHO search directions.

Problem type	size		relative gap		
	n	m	KSH	NT	AHO
Random SDP	10	10	5.05e-09	1.23e-09	9.43e-09
Random SDP	20	20	2.68e-09	2.04e-09	2.12e-09
Random SDP	50	50	2.40e-09	1.56e-09	1.34e-09
Norm minimization	100	100	1.06e-09	3.86e-09	7.91e-09
Chebyshev approx.	100	100	1.59e-09	4.57e-09	1.20e-09
Control and system theory	60	231	1.91e-07	1.21e-07	1.85e-06
Maximum cut	100	100	1.55e-09	4.26e-09	9.42e-09
Graph equipartition	100	101	5.33e-07	2.59e-06	4.19e-07
Maximum clique	100	1024	2.50e-09	5.87e-09	2.33e-09

Table 1.4 Performance comparison (time) among SDPA, SDPT3, CSDP and SDPSOL.

Problem type	size		time(sec.)			
	n	m	SDPA	SDPT3	CSDP	SDPSOL
Random SDP	10	10	0.1	0.9	0.2	0.6
Random SDP	20	20	0.5	2.9	3.9	27.1
Random SDP	50	50	10.4	32.6	159.0	7147.5
Norm minimization	100	100	149.7	350.5	3649.4	-
Chebyshev approx.	100	100	159.3	384.2	4058.8	-
Control and system theory	60	231	40.2	325.2	*	-
Maximum cut	100	100	10.3	65.8	12.7	-
Graph equipartition	100	101	11.7	89.2	*	-
Maximum clique	100	1024	172.1	3934.9	178.1	-

Table 1.5 Performance comparison (iterations) among SDPA, SDPT3, CSDP and SDPSOL.

Problem type	size		iterations			
	n	m	SDPA	SDPT3	CSDP	SDPSOL
Random SDP	10	10	15	11	15	16
Random SDP	20	20	14	11	14	17
Random SDP	50	50	14	10	15	20
Norm minimization	100	100	17	13	16	-
Chebyshev approx.	100	100	18	15	18	-
Control and system theory	60	231	35	21	*	-
Maximum cut	100	100	15	12	15	-
Graph equipartition	100	101	15	15	*	-
Maximum clique	100	1024	17	15	16	-

Table 1.6 Performance comparison (relative gap) among SDPA, SDPT3, CSDP and SDPSOL.

Problem type	size		relative gap			
	n	m	SDPA	SDPT3	CSDP	SDPSOL
Random SDP	10	10	5.05e-09	2.71e-08	4.06e-09	2.47e-07
Random SDP	20	20	2.68e-09	2.34e-09	7.66e-09	8.39e-07
Random SDP	50	50	2.40e-09	5.35e-06	4.16e-09	3.26e-06
Norm minimization	100	100	1.06e-09	3.83e-08	6.02e-09	-
Chebyshev approx.	100	100	1.59e-09	2.97e-10	1.37e-09	-
Control and system theory	60	231	1.91e-07	1.18e-05	*	-
Maximum cut	100	100	1.55e-09	3.08e-06	5.45e-09	-
Graph equipartition	100	101	5.33e-07	1.65e-03	*	-
Maximum clique	100	1024	2.50e-09	6.26e-09	4.47e-09	-

HRVW/KSH/M direction using SDPT3 [22, 23] excepting the ETP Problem. Monteiro and Zanjácomo [18] showed similar numerical results. Therefore, we mainly focus our attention on the HRVW/KSH/M direction in this paper.

As we have already seen in Section 1, there exists some MATLAB implementations besides SDPT3. Also, there are some other full implementations, for example, SDPSOL and CSDP, which are written in C language. Table 1.4, 1.5 and 1.6 show the performance comparisons among SDPA, SDPT3, CSDP and SDPSOL. It must be noted that SDPSOL is the only software that does not have the HRVW/KSH/M direction available, and we could not adopt the same initial point as we have not had access to the source code. However these details do not have significant importance in the comparison we made. SDPSOL can solve only small scale problems because it requires much memory compared to other software. SDPA and CSDP are faster when they solve the sparse problems, but we encountered some numerical instability (see the problems with $*$ in Table 1.4, 1.5 and 1.6) using CSDP. Therefore, we employ SDPT3 in this paper as it has numerical stability although it can not solve sparse problems rapidly.

1.5 NUMERICAL RESULTS.

In this section, we perform the main numerical experiments employing the HRVW/KSH/M search direction for SDPA and SDPT3. These choices were made judging from the preliminary numerical experiments of the previous section. We report numerical results of the 7 types of test problems presented in Section 3. These problems are roughly classified into 3 groups:

dense problems	— randomly generated semidefinite programs (Section 3.1),
	norm minimization problems (Section 3.2),
	Chebyshev approximation problems (Section 3.3),
mildly dense problems	— semidefinite programs from control and system theory
	(Section 3.4)
sparse problems	— maximum cut problems (Section 3.5),
	graph equipartition problems (Section 3.6),
	maximum clique problems (Section 3.7).

We address three main issues.

The first issue is an evaluation of the total performance of SDPA to know how large problems SDPA can solve within a reasonable time. We report the number of iterations and the computation time required for an approximate solution with a relative duality gap less than a given accuracy ϵ^* between 10^{-5} and 10^{-8}.

The second issue is a comparison between SDPA [6] and SDPT3 [23] through numerical experiments. Both software packages are based on the Mehrotra-type primal-dual predictor-corrector interior-point method although some parameters to control predictor search directions and step lengths are slightly different. To make the comparison fair, we utilized common initial points, the HRVW/KSH/M search direction (although the AHO and the NT search directions are available both in SDPA and SDPT3), and a common stopping criterion for both SDPA and SDPT3. Their major differences are

(a) The programming languages are different; SDPA is written in C++ while SDPT3 is written in MATLAB.

(b) SDPA incorporates a sparse matrix data structure and the efficient methods referred in Section 2 for computing search directions.

SDPA is a few times faster than the SDPT3 when they are applied to dense problems. This difference is mainly due to (a), and does not reflect the computational efficiency of the algorithms used in those software packages. In dense cases, however, the sparse data matrix structure makes the computation less efficient than the standard dense matrix structure. When SDPA is applied to mildly dense problems and sparse problems, SDPA is much faster than the SDPT3 in computation time. This is due to the reason (b).

The third issue is an analysis of SDPA. We focus our attention on some major time-consuming parts of SDPA:

(I) computation of the $m \times m$ dense positive definite matrix $\boldsymbol{B}$ (see Section 2.1).

(II) the LDL^T factorization of $\boldsymbol{B}$ for solving the system (1.5) of linear equations.

(III) computation of dX and dZ (see (1.6)).

(IV) computation of the minimum eigenvalues of $X^{-1}dX$, $Z^{-1}dZ$ and XZ (see Section 2.2).

Other parts of SDPA which are not included in (I), (II), (III) and (IV) are the Cholesky factorization of X and Z, the computation of Z^{-1}, the computation of the primal and dual step lengths, etc..

We remark that (I), (II), (III) and (IV) would require $O(mn^3 + m^2n^2)$, $O(m^3)$, $O(n^3 + mn^2)$ and $O(n^3)$ arithmetic operations, respectively, if we performed dense computation neglecting possible sparsity in data matrices. The most time consuming part among (I), (II), (III) and (IV) varies with the test problems to be solved. Part (I) requires about $80 - 95$ percent of the computation time for dense and mildly dense problems, while part (I) makes up less than 10 percent and some other parts become more important for sparse problems.

All numerical experiments in this section were executed on DEC Alpha Server 8400 (CPU Alpha 21164-437MHz with 8GB memory) under DIGITAL UNIX V3.2G.

1.5.1 Randomly Generated Semidefinite Program.

We took an initial point (X^0, y^0, Z^0) of the form $(100I, \mathbf{0}, 100I)$ for all randomly generated semidefinite programs. Table 1.7 and 1.8 summarize numerical results. Recall that all A_i's are dense matrices. We note that the number of iterations does not seem to depend significantly on n and m. † means that the problem is primal or dual infeasible. In these cases, SDPA detected that the problem has no feasible solution in a prescribed region (see Section 2.4) within a small number of iterations compared with SDPT3. SDPA is about $2 - 4$ times as fast as SDPT3 when $m/n = 1$, and even faster than SDPT3 when the ratio m/n gets larger. We also observe that SDPA worked as stable and efficient (measured in terms of the number of iterations) as SDPT3.

Table 1.7 Numerical results on randomly generated semidefinite programs.

		time(sec.)		iterations		relative gap	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	10	0.1	0.4	9	9	4.10e-09	7.05e-09
20	20	0.4	1.7	10	9	7.57e-10	6.14e-09
† 20	20	0.2	2.7	2	10	2.00e+00	3.11e+06
25	25	0.8	2.4	9	9	9.02e-09	7.15e-10
30	30	1.5	4.2	10	9	3.98e-09	2.00e-09
† 30	30	0.2	5.8	4	9	1.88e+00	3.28e+06
50	50	10.9	35.6	10	9	6.61e-09	5.83e-09
† 50	50	0.6	62.5	2	20	1.94e+00	6.80e+06
50	200	65.9	285.1	11	10	2.81e-09	7.16e-09
100	100	129.3	367.8	10	10	8.90e-09	3.39e-09
100	200	433.3	2069.6	13	12	1.89e-09	6.19e-09
200	50	552.9	896.2	11	10	7.60e-10	9.10e-09
200	100	1130.1	2220.9	11	11	2.52e-09	4.96e-10

From Table 1.8, we know that part (I) consisting of the computation of the $m \times m$ matrix B occupied over 90 percent of the total computation time.

1.5.2 Norm Minimization and Chebyshev Problems.

For each norm minimization problem, we took a feasible initial point (X^0, y^0, Z^0) such that

$$\begin{aligned} y_i^0 &= 0 \ (1 \le i \le p = m-1), \\ y_m^0 &= 1.1\|A_0\|, \end{aligned}$$

Table 1.8 Computation time of major parts of SDPA applied to randomly generated semidefinite programs.

	$n=100$, $m=100$		$n=100$, $m=200$	
part	time(sec.)	ratio(%)	time(sec.)	ratio(%)
(I)	116.80	90.32	413.07	95.34
(II)	0.05	0.04	0.47	0.11
(III)	2.60	2.01	5.88	1.36
(IV)	2.70	2.09	3.43	0.79
all other parts	7.17	5.54	10.43	2.41

$$\begin{aligned} \boldsymbol{Z}^0 &= \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} y_m^0 + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix} \succ \boldsymbol{O}, \\ \boldsymbol{X}^0 &= \frac{1}{n} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix}. \end{aligned}$$

Recall that a Chebyshev problem is a special case of norm minimization problems. We converted each Chebyshev problem into a norm minimization problem, and then applied the same initial point as above to the resultant norm minimization problem. Table 1.9, 1.10 and 1.11 summarize numerical results. Note that about 50% of the elements of data matrices $\boldsymbol{A}_i$ $(1 \le i \le p)$ are nonzero. But the percentage of part (I) with respect to the total computation time is slightly less than in the case of randomly generated semidefinite programs of Section 5.1; it is still more than 85%.

Table 1.9 Numerical results on norm minimization problems.

		time(sec.)		iterations		relative gap	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
20	10	0.1	0.7	9	9	1.11e-09	1.45e-09
40	20	1.1	5.4	9	9	3.90e-09	4.99e-09
50	25	3.5	8.3	9	9	8.24e-09	2.65e-09
60	30	5.4	15.9	9	9	3.26e-09	2.38e-09
100	50	38.3	165.7	10	10	6.30e-09	3.99e-10
200	100	721.2	1868.4	11	10	9.09e-10	9.65e-10

Table 1.10 Numerical results on Chebyshev approximation problems.

		time(sec.)		iterations		relative gap	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
20	10	0.1	0.8	9	9	9.46e-10	6.67e-09
40	20	1.1	6.0	9	9	3.49e-09	4.45e-09
50	25	3.4	11.2	9	9	1.82e-09	8.20e-09
60	30	5.9	23.0	10	10	8.76e-10	1.14e-10
100	50	41.3	163.6	11	10	2.18e-09	3.72e-09
200	100	713.3	2613.3	11	10	6.86e-10	2.77e-09

Table 1.11 Computation time of major parts of SDPA applied to NMP (norm minimization problems) and CAP (Chebyshev approximation problems).

	NMP, $n = 200$, $m = 100$		CAP, $n = 200$, $m = 100$	
part	time(sec.)	ratio(%)	time(sec.)	ratio(%)
(I)	627.80	87.05	620.15	86.94
(II)	0.02	0.00	0.08	0.00
(III)	15.12	2.10	15.08	2.11
(IV)	24.82	3.44	24.88	3.49
all other parts	53.44	7.41	53.1	7.44

1.5.3 **Control and System Theory Problem.**

We restrict ourselves to particular cases where $k = l$ takes values varying from 5 through 55 as shown in Table 6. We took $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0) = (10^i\boldsymbol{I}, \boldsymbol{0}, 10^i\boldsymbol{I})$ for an infeasible initial point, where we chose $i = 4$ or 5 depending on each test problem. Comparing with other types of test problems, we notice that the number of iterations is larger. This phenomenon occurred because the primal feasible region is narrow, so that both algorithms need much time to reach the feasible region. Particularly, SDPT3 stopped with the message “lack of progress in corrector” and “lack of progress in predictor” in the cases of $k = \ell = 20$ and 25, respectively, before it would attain a relative gap less than 10^{-5} (see the numbers with $\star$ in Table 1.12). In most of the cases, however, we observed that once SDPA and SDPT3 moved into the feasible region, they attained an approximate optimal solution with a given accuracy ϵ^* in a few steps.

Table 1.12 Numerical results on control and system theory problems

			time(sec.)		iterations		relative gap	
$k = l$	n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
5	15	21	0.1	2.4	21	18	1.36e-07	4.21e-07
10	30	66	1.3	25.7	22	20	2.35e-07	8.10e-07
15	45	136	8.0	106.5	26	23	6.43e-07	3.97e-06
20	60	231	25.4	355.3	28	21	8.38e-07	1.18e-05*
25	75	351	114.9	1059.2	31	22	3.74e-08	2.02e-05*
30	90	496	221.0	-	32	-	6.67e-06	-
35	105	666	457.0	-	27	-	2.35e-06	-
40	120	861	630.1	-	26	-	8.42e-06	-
45	135	1081	1594.8	-	28	-	7.33e-06	-
50	150	1326	3217.5	-	37	-	6.21e-06	-
55	165	1596	6903.0	-	35	-	9.91e-06	-

Table 1.13 gives the CPU time required in major parts of SDPA applied to the two largest cases shown in Table 12. Compared with the previous three problems (see Tables 1.8 and 1.11), part (II) consisting of the LDL^T factorization of the $m \times m$ matrix $\boldsymbol{B}$ now constitutes a much larger percentage of the computation time. This is because m is about $9 - 10$ times larger than n in these two cases.

1.5.4 Maximum Cut Problem and Graph Equipartition Problem.

We took a feasible initial point $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ such that

$$\boldsymbol{X}^0 \;=\; \mathrm{Diag}(\boldsymbol{b}),$$

Table 1.13 Computation time of major parts of SDPA applied to control and system theory problems.

	$n = 150, m = 1326$		$n = 165, m = 1596$	
part	time(sec.)	ratio(%)	time(sec.)	ratio(%)
(I)	2553.15	79.35	5582.65	80.87
(II)	572.33	16.86	1194.02	17.30
(III)	9.60	0.30	16.28	0.24
(IV)	11.10	0.34	14.32	0.21
all other parts	71.35	1.55	95.76	1.39

$$\begin{aligned} \boldsymbol{y}^0 &= -1.1 \cdot \mathrm{abs}(\boldsymbol{A}_0) \cdot \boldsymbol{e}, \\ \boldsymbol{Z}^0 &= \boldsymbol{A}_0 - \mathrm{Diag}(\boldsymbol{y}^0). \end{aligned}$$

in maximum cut problems, while we took an initial point $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ of the form $(100\boldsymbol{I}, \boldsymbol{0}, 100\boldsymbol{I})$ in graph equipartition problems. Tables 1.14, 1.15 and 1.16 show numerical results.

Table 1.14 Numerical results on maximum cut problems.

	time(sec.)		iterations		relative gap	
n	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	0.1	0.4	8	8	2.08e-09	5.51e-10
20	0.1	0.9	9	8	1.67e-09	9.16e-09
25	0.2	1.5	9	9	3.61e-09	5.68e-09
30	0.3	2.5	10	10	3.50e-09	2.95e-10
50	1.5	7.9	9	9	3.21e-09	3.66e-09
100	9.0	73.4	10	11	1.57e-09	2.54e-10
150	40.3	277.6	10	10	1.77e-09	1.89e-09
200	92.8	865.2	11	10	7.12e-10	7.13e-09
250	311.6	-	13	-	2.95e-09	-
500	2998.6	-	16	-	1.22e-09	-
1000	69490.1	-	16	-	3.59e-08	-
1250	111615.9	-	18	-	5.90e-09	-

Both maximum cut and graph equipartition problems are sparse problems. In particular, all $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ of the maximum cut problems have only one nonzero element, while exactly one $\boldsymbol{A}_i$ is a dense matrix having n^2 nonzero elements and all other $\boldsymbol{A}_i$'s involve one nonzero element in graph equipartition problems. For such sparse problems, the efficient method [7] referred in Section 2.1 is expected to work effectively. In fact, we observe that SDPA solved both types of problems much faster than SDPT3. We also see in Table 1.16 that part (I) is less than 1% of the total computation time in the maximum cut problem and it is less than 7% in the graph equipartition problem.

1.5.5 Maximum Clique Problem.

SDPA started from the initial point $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0) = (100\boldsymbol{I}, \boldsymbol{0}, 100\boldsymbol{I})$ in all maximum clique problems. Table 1.17 and 1.18 show numerical results. Recall that the number of constraints m is corresponding to

$$\text{“the number } n(n-1)/2 - |E| \text{ of node pairs having no edge”} + 1.$$

This implies that m can be of $O(n^2)$. We see from Table 1.17 that the computation time strongly depends on m. In Table 18, we see that part (II) depends more heavily on m than part(I).

Table 1.15 Numerical results on graph equipartition problems.

	time(sec.)		iterations		relative gap	
n	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	0.1	0.5	11	12	2.47e-06	9.83e-06
20	0.2	1.7	12	14	2.24e-06	4.38e-06
25	0.3	2.5	12	13	7.22e-06	7.23e-06
30	0.5	4.1	12	13	3.53e-06	4.36e-06
50	2.3	13.7	12	13	8.02e-07	4.31e-06
100	12.5	104.6	12	13	6.18e-06	8.67e-06
150	52.1	433.0	11	13	5.58e-06	4.25e-06
200	115.1	1174.1	12	12	2.39e-06	9.05e-06
250	418.6	-	15	-	2.61e-06	-
500	3197.9	-	15	-	1.22e-09	-
1000	63130.7	-	21	-	4.80e-06	-
1250	112375.7	-	15	-	4.16e-05	-

Table 1.16 Computation time of major parts of SDPA applied to maximum cut problems (MCP) and graph equipartition problems (GPP).

	(MCP) $n = 1250$		(GPP) $n = 1250$	
part	time(sec.)	ratio(%)	time(sec.)	ratio(%)
(I)	26.07	0.02	7570.17	6.74
(II)	197.68	0.18	171.43	0.15
(III)	20469.43	18.34	17803.15	15.84
(IV)	16644.95	14.91	14686.77	13.07
all other parts	74277.80	66.55	72144.17	64.20

Table 1.17 Numerical results on maximum clique problems.

		time(sec.)		iterations		relative gap	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
50	132	2.8	49.8	15	15	7.43e-09	6.37e-10
50	253	4.8	147.9	15	14	3.85e-09	4.91e-09
50	597	20.8	716.7	14	13	5.72e-10	4.17e-09
100	539	30.2	1102.0	16	14	9.73e-10	6.33e-09
100	1024	143.0	4298.9	16	15	3.49e-09	6.26e-09
150	562	84.9	-	16	-	7.80e-09	-
150	1102	215.2	-	16	-	1.19e-09	-
200	622	191.7	-	19	-	7.89e-09	-
200	993	237.4	-	16	-	2.63e-09	-
250	652	492.7	-	18	-	8.09e-09	-
250	973	528.4	-	17	-	1.55e-09	-
300	944	749.3	-	18	-	1.81e-09	-
300	1401	1041.9	-	18	-	8.42e-10	-
300	4568	19028.7	-	19	-	3.55e-09	-

Table 1.18 Computation time of major parts of SDPA applied to maximum clique problems.

	$n = 300$, $m = 944$		$n = 300$, $m = 4568$	
part	time(sec.)	ratio(%)	time(sec.)	ratio(%)
(I)	16.63	2.22	513.05	2.70
(II)	70.63	9.43	17773.65	93.40
(III)	104.37	13.93	107.38	0.56
(IV)	137.85	18.34	145.95	0.77
all other parts	419.85	54.99	488.67	2.57

1.6 CONCLUDING REMARKS.

A large scale sparse SDP means that m and/or n are large and that all or most of the data matrices $\boldsymbol{A}_i$ $(0 \leq i \leq m)$ are sparse. Here m denotes the number of equality constraints and n the size of the symmetric variable matrix of primal problem $\mathcal{P}$ in the standard form (1.1). For simplicity of discussion, we assume throughout this section that all data matrices $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ have at most f nonzeros, where f is a small positive integer, but $\boldsymbol{A}_0$ can be dense. In such a case, SDPA effectively utilizes the sparsity of data matrices $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ for

(I) computation of the $m \times m$ dense positive definite matrix $\boldsymbol{B}$

to reduce the number of arithmetic operations to $O(m^2 f^2)$ by applying formula $\mathcal{F}_3$ proposed in the paper [7], while SDPA still needs $O(m^3)$, $O(n^3 + mf)$ and $O(n^3)$ arithmetic operations for

(II) the LDL^T factorization of $\boldsymbol{B}$,

(III) computation of $d\boldsymbol{X}$ and $d\boldsymbol{Z}$, and

(IV) computation of the minimum eigenvalues of $\boldsymbol{X}^{-1}d\boldsymbol{X}$, $\boldsymbol{Z}^{-1}d\boldsymbol{Z}$ and $\boldsymbol{X}\boldsymbol{Z}$,

respectively. We performed numerical experiments for investigating how the major time-consuming parts of SDPA vary with the problem size, the number of equality constraints and the sparsity of data matrices.

When the size m of the dense positive definite matrix $\boldsymbol{B}$ is large, say, more than several thousands, it is difficult to perform (II), and even to store the entire matrix $\boldsymbol{B}$ in memory. A standard technique to resolve such a difficulty is to use the conjugate gradient method to solve the system $\boldsymbol{B}d\boldsymbol{y} = \boldsymbol{g}$ of equations approximately; it is carried out without storing the entire matrix $\boldsymbol{B}$ (see Concluding Remark (C) of [7], also [13, 16, 26, 27]). We report some preliminary but promising numerical results on the use of the conjugate gradient method in Table 1.19. We solved SDP relaxations of maximum clique problems with $n = 100,\ 200,\ 400,\ 500$ on DEC Alpha (CPU 21164-300MHz with 256MB memory). Recall that n is the number of nodes and m "the number of node pairs having no edge" +1. The columns in "LDL^T" of Table 1.19 denote the number of iterations of SDPA and the total CPU time in seconds when we solved $\boldsymbol{B}d\boldsymbol{y} = \boldsymbol{g}$ by the LDL^T factorization, while the columns in "CG" denote the number of (major) iterations of SDPA, the total CPU time in seconds and the number of the conjugate gradient iterations at the last (major) iteration of SDPA when we solved $\boldsymbol{B}d\boldsymbol{y} = \boldsymbol{g}$ approximately by the conjugate gradient method with simple diagonal scaling. In all cases, we stopped the iteration when the duality gap (|P−D|) got less than 0.1. We see in Table 1.19 that the use of the conjugate gradient method for problems with $n = 100$ worked more and more efficiently than the use of LDL^T method as m gets larger. In cases where $n \geq 200$ and $m \geq 8006$, we could not perform the LDL^T factorization of $\boldsymbol{B}$ because we could not store the entire matrix $\boldsymbol{B}$ in memory. Overall, the conjugate gradient method worked very efficiently. We should mention, however, that we encountered some difficulty in using the conjugate gradient method to compute more accurate approximate solutions with duality gap significantly less than 0.1.

Parts (III) and (IV) require $O(n^3 + mf)$ and $O(n^3)$ arithmetic operations, respectively. Therefore they become the most expensive parts when n gets larger, and solving a semidefinite program via SDPA seems

Table 1.19 Comparison between the use of the conjugate gradient method and the use of the LDL^T factorization in SDPA applied to maximum clique problems.

		LDL^T		CG		
n	m	iterations	time(sec.)	iterations	time(sec.)	CG iterations
100	539	7	13.3	10	24.7	336
100	1024	6	75.7	9	22.1	319
100	2029	6	580.5	9	26.1	332
100	2513	6	1116.1	8	17.7	208
200	1977	7	662.6	10	209.8	385
200	4050	7	5418.6	9	186.3	381
200	8006			9	272.1	540
200	9957			9	258.0	488
400	16077			10	3618.2	913
400	24079			10	4322.4	1114
400	32116			10	2302.5	525
400	40094			9	2994.6	641
500	25107			10	6646.0	798
500	37596			10	6973.3	838
500	50071			10	8339.8	1034
500	62499			10	6152.9	694

impractical when n exceeds several thousands; for example, the graph equipartition problem with the size $n = 1250$ given in Table 1.15 required more than 31 hours of computation time.

Sparsity is one particular form of problem structure we exploit here because it is an important feature arising from SDP relaxation in combinatorial optimization and it has a straightforward tractable structure. However, we need an accumulation of more numerical results and knowledge for exploiting other problem structures and we consider them for future work.

Suppose now that $\boldsymbol{A}_0$ is also a sparse matrix having at most $f_0 = O(n)$ nonzero elements, n is large and that m is of $O(n)$; the SDP relaxation of the maximum cut problem of a sparse graph is such a case. Then the $n \times n$ dual variable symmetric matrix $\boldsymbol{Z} = \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i y_i$ turns out to be a sparse matrix having $O(f_0 + mf)$ nonzeros although the $n \times n$ primal variable symmetric matrix $\boldsymbol{X}$ is generally dense. Hence, working only on the dual space, *i.e.*, the space of $(\boldsymbol{y}, \boldsymbol{Z}) \in R^m \times \mathcal{S}^n$, makes it possible to fully exploit the sparsity. This observation is due to Yinyu Ye et al. who suggested a dual potential reduction method for the SDP relaxation of the maximum cut problem of a sparse graph. They require only about 2 hours of computation time when they solve a problem with size $n = 5000$ using a PC-GATEWAY2000 (CPU Pentium-233MHz with 64MB memory), while we require more than 31 hours of computation time when we solve the problem with size $n = 1250$. See [4].

The SDP relaxation (1.16) of the graph equipartition problem involves $n \times n$ matrices $\boldsymbol{E}_{ii}$ $(1 \leq i \leq n)$ having one nonzero and the $n \times n$ matrix $\boldsymbol{E}$ having all elements 1 in the constraint. Although the matrix $\boldsymbol{E}$ is dense, it is a rank 1 matrix of the form $\boldsymbol{E} = \boldsymbol{e}\boldsymbol{e}^T$, where $\boldsymbol{e}$ denotes the n-dimensional vector of 1's. Also $\boldsymbol{E}_{ij}$ appeared in the semidefinite programming relaxation (1.17) of the maximum clique problem is a rank 2 matrix of the structure $\boldsymbol{E}_{ij} = \boldsymbol{e}_i\boldsymbol{e}_j^T + \boldsymbol{e}_j\boldsymbol{e}_i^T$, where $\boldsymbol{e}_k$ denotes the n dimensional unit vector with the kth element 1 and all other elements 0. In general, semidefinite programming relaxations of combinatorial optimization problems involve many rank 1 and rank 2 matrices in their constraints. When a data matrix $\boldsymbol{A}_i$ is not sparse but is a low rank matrix of the structure $\boldsymbol{A}_i = \sum_{k=1}^{q}(\boldsymbol{a}_k\boldsymbol{b}_k^T + \boldsymbol{b}_k\boldsymbol{a}_k^T)$ for some $\boldsymbol{a}_k, \boldsymbol{b}_k \in R^n$ $(1 \leq k \leq q)$ and some small positive integer q, we can utilize such a low rank structure to make the computation of the matrix $\boldsymbol{B}$ more efficient. This observation is due to C. Helmberg and S. Karisch. See [10]. Other papers also observed the importance of exploiting different kinds of structure [8, 26].

Acknowledgments

Part of this paper was presented at the HPOPT workshop (August 19 – 22, 1997) which was held in Rotterdam, the Netherlands. Three of the authors, Fujisawa, Fukuda and Kojima attended the workshop. Kojima would like to thank Professors Kees Roos, Tamas Terlaky and Shuzhong Zhang who invited him to the workshop.

References

[1] F. Alizadeh, J.-P.A. Haeberly and M.L. Overton, "Primal-dual interior-point methods for semidefinite programming," Working Paper, 1994.

[2] F. Alizadeh, J.-P.A. Haeberly and M.L. Overton, "Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results," Report 721, Computer Science Dept., New York University, New York, May 1996, to appear in *SIAM Journal on Optimization.*

[3] F. Alizadeh, J.-P.A. Haeberly, M.V. Nayakkankuppam, M.L. Overton and S. Schmieta, "SDPpack – User's Guide –," Comp. Sci. Dept., New York University, New York, June 1997. Available at http://cs.nyu.edu/cs/faculty/overton/sdppack/sdppack.html.

[4] S. J. Benson, Y. Ye and X. Zhang, "Solving large-scale sparse semidefinite programs for combinatorial optimization," Applied Mathematics and Computer Sciences, The University of Iowa, Iowa City, Iowa 52242, September 1997.

[5] B. Borchers, "CSDP, a C library for semidefinite programming," Department of Mathematics, New Mexico Institute of Mining and Technology, 801 Leroy Place Socorro, New Mexico 87801, March 1997. Available at http://www.nmt.edu/~borchers/csdp.html.

[6] K. Fujisawa, M. Kojima and K. Nakata, "SDPA (Semidefinite Programming Algorithm) – User's Manual –," Technical Report B-308, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan, December 1995, Revised August 1996. Available at ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA.

[7] K. Fujisawa, M. Kojima and K. Nakata, "Exploiting sparsity in primal-dual interior-point methods for semidefinite programming," *Mathematical Programming* **79** (1997) 235–253.

[8] P. Gahinet and A. Nemirovski, "The projective method for solving linear matrix inequalities.", *Mathematical Programming* **77** (1997) 163–190.

[9] G. H. Goulb and C. F. Van Loan, *Matrix Computations (second edition),* (The John Hopkins University Press, Baltimore, Maryland, 1989).

[10] C. Helmberg and F. Rendl, "Solving quadratic $(0,1)$-problems by semidefinite programming and cutting planes," SC-95-35, Konrad-Zuse-Zentrum Berlin, Berlin, Germany , 1995.

[11] C. Helmberg, F. Rendl, R.J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization* **6** (1996) 342–361.

[12] S. E. Karisch, F. Rendl and J. Clausen, "Solving graph bisection problems with semidefinite programming," Technical Report DIKU-TR-97/9, Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark, July 1997.

[13] M. Kojima, M. Shida and S. Shindoh, "Search directions in the SDP and the monotone SDLCP: generalization and inexact computation," to appear in *Mathematical Programming.*

[14] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," *SIAM Journal on Optimization* **7** (1997) 86–125.

[15] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization* **2** (1992) 575–601.

[16] C.-J. Lin and R. Saigal, "Semidefinite Programming and the Quadratic Assignment Problem," 16th International Symposium on Mathematical Programming, Lausanne, Switzerland, August 1997.

[17] R.D.C. Monteiro, "Primal-dual path-following algorithms for semidefinite programming," *SIAM Journal on Optimization* **7** (1997) 663–678.

[18] R.D.C. Monteiro, P.R. Zanjácomo, "Implementation of primal-dual methods for semidefinite programming based on Monteiro and Tsuchiya directions and their variants," Technical Report, School Industrial and Systems Engineering, Georgia Tech., Atlanta, GA 30332, July 1997, Revised August 1997.

[19] Yu.E. Nesterov and M.J. Todd, "Self-scaled barriers and interior-point methods in convex programming," *Mathematics of Operations Research* **22** (1997) 1–42.

[20] Yu.E. Nesterov and M.J. Todd, "Primal-dual interior-point methods for self-scaled cones," *SIAM J. on Optimization* 8 (1988) 324-364.

[21] F.A. Potra, R. Sheng and N. Brixius, "SDPHA – a MATLAB implementation of homogeneous interior-point algorithms for semidefinite programming," Department of Mathematics, University of Iowa, Iowa City, IA 52242, April 1997.
Available at http://www.math.uiowa.edu/~rsheng/SDPHA/sdpha.html.

[22] M.J. Todd, K.C. Toh and R.H. Tütüncü, "On the Nesterov-Todd direction in semidefinite programming," Technical Report, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York 14853-3801, USA, March 1996, Revised May 1996, to appear in *SIAM Journal on Optimization.*

[23] K.C. Toh, M.J. Todd and R.H. Tütüncü, "SDPT3 – a MATLAB software package for semidefinite programming," Dept. of Mathematics, National University of Singapore, 10 Kent Ridge Crescent, Singapore, December 1996.
Available at http://www.math.nus.sg/~mattohkc/index.html.

[24] K.C. Toh and L.N. Trefethen, "The Chebyshev polynomial of matrix", manuscript, Center for Applied Mathematics, Cornell University, Ithaca, NY, 1996.

[25] L. Vandenberghe and S. Boyd, " Semidefinite programming," *SIAM Review* **38** (1996) 49–95.

[26] L. Vandenberghe and S. Boyd, "A primal-dual potential reduction method for problems involving matrix inequalities," *Mathematical Programming* **69** (1995) 205-236.

[27] H. Wolkowicz and Q. Zhao, "Semidefinite programming relaxations for the graph partitioning problem", CORR Report, University of Waterloo, Ontario, Canada, Oct. 1996.

[28] S.-P. Wu and S. Boyd. "SDPSOL: a parser/solver for SDP and MAXDET problems with matrix structure", Department of Electrical Engineering, Stanford University, June, 1996. Available at http://www-isl.stanford.edu/people/boyd/SDPSOL.html