CSDP, a C library for semidefinite programming.

Brian Borchers

Department of Mathematics, New Mexico Tech, Socorro, NM 87801, USA

(Received date to be inserted)

1 Introduction

A number of codes for semidefinite programming (SDP) are already available, including [1, 3, 8, 9, 10]. Why introduce yet another code for SDP?

CSDP is written in C for efficiency and portability. The code is designed to make use of highly optimized linear algebra routines from the LINPACK or LAPACK libraries. CSDP is distributed with version of the necessary LINPACK routines that have been translated into C. The package also includes an optimized version of the BLAS routine DGEMM [6, 7].

CSDP is designed to handle constraint matrices with general sparse structure. CSDP can accommodate linear inequality constraints as well as linear equality constraints. In addition to its SDP solver, the CSDP library contains routines for reading and writing SDP problems and solutions. The code has been designed for use both as a stand alone solver and as a callable subroutine for use within larger programs that require the solution of SDP subproblems. We present results from the solution of the SDPLIB test problems [2]. CSDP has also been used in a code for the solution of MAX-2-SAT problems [5].

The remainder of this paper is organized as follows. First, we discuss the formulation of the semidefinite programming problem used by CSDP. We then describe the predictor corrector algorithm used by CSDP to solve the SDP. We discuss the storage requirements of the algorithm as well as its computational complexity. Finally, we present results from the solution of a number of test problems.

2 The SDP Problem

We consider semidefinite programming problems of the form

$$\begin{array}{rrcl} \max & \operatorname{tr}(CX) & & \\ & A(X) & = & a \\ & B(X) & \leq & b \\ & X & \succeq & 0 \end{array} \tag{1}$$

where

$$A(X) = \begin{bmatrix} \text{tr } (A_1 X) \\ \text{tr } (A_2 X) \\ \dots \\ \text{tr } (A_k X) \end{bmatrix} \tag{2}$$

and

$$B(X) = \begin{bmatrix} \text{tr } (B_1 X) \\ \text{tr } (B_2 X) \\ \dots \\ \text{tr } (B_l X) \end{bmatrix}. \tag{3}$$

All of the matrices are assumed to be symmetric.

The dual of this SDP is

$$\begin{array}{lrcl} \min & a^T y + b^T t & & \\ & A^T(y) + B^T(t) - C & = & Z \\ & Z & \succeq & 0 \\ & t & \geq & 0 \end{array} \tag{4}$$

where

$$A^T(y) = \sum_{i=1}^{k} y_i A_i \tag{5}$$

and

$$B^T(t) = \sum_{i=1}^{l} t_i B_i. \tag{6}$$

3 The Predictor Corrector Algorithm

The algorithm for SDP discussed in this section is a predictor corrector variant of the algorithm presented by Helmberg, Rendl, Vanderbei, and Wolkowicz [4]. We will make frequent reference to this paper, and use its notation. We begin with the dual barrier problem

$$\begin{array}{lrcl} \min & a^T y + b^T t - \mu(\log\det Z + e^T \log t) & & \\ & A^T(y) + B^T(t) - C & = & Z \\ & Z & \succeq & 0 \\ & t & \geq & 0. \end{array} \tag{7}$$

The algorithm works by taking steps towards a solution to (7), and slowly reducing the parameter μ. In the limit as μ goes to 0, we obtain a solution to (1). It can be shown that an optimal solution to (7) has

$$\mu = \frac{\text{tr}(ZX) + t^T(b - B(X))}{(n+m)}. \tag{8}$$

In order to drive μ to zero, at each iteration of the algorithm we adjust μ to

$$\mu = \frac{\operatorname{tr}(ZX) + t^T(b - B(X))}{2(n+m)}. \tag{9}$$

The first order necessary optimality conditions for (7) are

$$\begin{array}{rcl} Z + C - A^T(y) - B^T(t) & = & 0 \\ a - A(X) & = & 0 \\ b - B(X) - \mu t^{-1} & = & 0 \\ X - \mu Z^{-1} & = & 0. \end{array} \tag{10}$$

Our algorithm is designed to work with a starting solution that may not satisfy $A(X) = a$ or $Z = A^T(y) + B^T(t) - C$, so we define

$$F_p = a - A(X) \tag{11}$$

and

$$F_d = -A^T y - B^T(t) + C + Z. \tag{12}$$

The predictor step is the Newton's method step for these equations with $\mu = 0$.

$$\begin{array}{rcl} \Delta\hat{Z} - A^T(\Delta\hat{y}) - B^T(\Delta\hat{t}) & = & -F_d \\ -A(\Delta\hat{X}) & = & -F_p \\ \Delta\hat{t} \circ (b - B(X)) - t \circ B(\Delta\hat{X}) & = & -t \circ (b - B(X)) \\ Z\Delta\hat{X} + \Delta\hat{Z}X & = & -ZX. \end{array} \tag{13}$$

These equations are solved as in [4]. We reduce (13) to

$$\begin{array}{rcl} A(Z^{-1}A^T(\Delta\hat{y})X) + A(Z^{-1}B^T(\Delta\hat{t})X) & = & -a + A(Z^{-1}F_dX) \\ B(Z^{-1}A^T(\Delta\hat{y})X) + (b - B(X)) \circ t^{-1} \circ \Delta\hat{t} + B(Z^{-1}B^T(\Delta\hat{t})X) & = & -b + B(Z^{-1}F_dX). \end{array} \tag{14}$$

In matrix form, this system of equations can be written as

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} \begin{bmatrix} \Delta\hat{y} \\ \Delta\hat{t} \end{bmatrix} = \begin{bmatrix} -a + A(Z^{-1}F_dX) \\ -b + B(Z^{-1}F_dX) \end{bmatrix} \tag{15}$$

where

$$\begin{array}{rcl} O_{11} & = & \left[\ A(Z^{-1}A^T(e_1)X) \ \ \ldots \ \ A(Z^{-1}A^T(e_k)X)\ \right] \\ O_{12} & = & \left[\ A(Z^{-1}B^T(e_1)X) \ \ \ldots \ \ A(Z^{-1}B^T(e_l)X)\ \right] \\ O_{21} & = & \left[\ B(Z^{-1}A^T(e_1)X) \ \ \ldots \ \ B(Z^{-1}A^T(e_k)X)\ \right] \\ O_{22} & = & \left[\ B(Z^{-1}B^T(e_1)X) \ \ \ldots \ \ B(Z^{-1}B^T(e_l)X)\ \right] + \operatorname{diag}\left((b - B(X) \circ t^{-1}\right). \end{array} \tag{16}$$

As Helmberg, Rendl, Vanderbei, and Wolkowicz have shown, the O matrix is symmetric and positive definite [4]. Thus we can compute Cholesky factorization of O and then use the factorization to solve the system of equations. Once we have solved these equations for $\Delta\hat{y}$ and $\Delta\hat{t}$, we compute $\Delta\hat{X}$ and $\Delta\hat{Z}$ as

$$\Delta\hat{X} = -X + Z^{-1}F_dX - Z^{-1}(A^T(\Delta\hat{y}) + B^T(\Delta\hat{t}))X \tag{17}$$

and

$$\Delta\hat{Z} = -F_d + A^T(\Delta\hat{y}) + B^T(\Delta\hat{t}). \tag{18}$$

Note that $\Delta\hat{X}$ might not be symmetric. In order to keep our solution X symmetric, we force $\Delta\hat{X}$ to be symmetric by averaging the off diagonal entries.

For the corrector step, we compute a Newton step from $(X + \Delta\hat{X}, Z + \Delta\hat{Z}, y + \Delta\hat{y}, t + \Delta\hat{t})$ towards a solution to (10).

$$\begin{array}{rcl} \Delta\bar{Z} - A^T(\Delta\bar{y}) - B^T(\Delta\bar{t}) & = & 0 \\ -A(\Delta\bar{X}) & = & 0 \\ \Delta\bar{t} \circ (b - B(X + \Delta\hat{X})) - t \circ B(\Delta\bar{X}) & = & \mu e - t \circ (b - B(X + \Delta\hat{X})) \\ (Z + \Delta\hat{Z})\Delta\bar{X} + \Delta\bar{Z}(X + \Delta\hat{X}) & = & -(Z + \Delta\hat{Z})(X + \Delta\hat{X}) + \mu I. \end{array} \tag{19}$$

Dropping higher order terms from the left hand side, and simplifying the right hand side, we obtain

$$\begin{array}{rcl} \Delta\bar{Z} - A^T(\Delta\bar{y}) - B^T(\Delta\bar{t}) & = & 0 \\ -A(\Delta\bar{X}) & = & 0 \\ \Delta\bar{t} \circ (b - B(X)) - t \circ B(\Delta\bar{X}) & = & \mu e - t \circ (b - B(X + \Delta\hat{X})) \\ Z\Delta\bar{X} + \Delta\bar{Z}X & = & -\Delta\hat{Z}\Delta\hat{X} + \mu I. \end{array} \tag{20}$$

These equations have the same form as (13) and are solved as before to obtain $(\Delta\bar{X}, \Delta\bar{Z}, \Delta\bar{y}, \Delta\bar{t})$. Next, we add the predictor and corrector steps to compute

$$\begin{array}{rcl} \Delta X & = & \Delta\hat{X} + \Delta\bar{X} \\ \Delta Z & = & \Delta\hat{Z} + \Delta\bar{Z} \\ \Delta y & = & \Delta\hat{y} + \Delta\bar{y} \\ \Delta t & = & \Delta\hat{t} + \Delta\bar{t}. \end{array} \tag{21}$$

We would like to take full steps of length one in each of X, y, t, and Z. However, there is a chance that this would lead to an infeasible solution. Thus we perform a line search to find the maximum safe steps α_P and α_D. Finally, we move from the current point (X, y, t, Z) to $(X + \alpha_P\Delta X, y + \alpha_D\Delta y, t + \alpha_D\Delta t, Z + \alpha_D\Delta Z)$.

In practice, the system matrix O may become numerically singular even though X and Z are numerically nonsingular. In this case, CSDP returns to the previous solution, and executes a centering step with

$$\mu = \frac{\mathrm{tr}(ZX) + t^T(b - B(X))}{(n + m)}. \tag{22}$$

Users of CSDP can specify their own termination criteria. However, the default criteria are that

$$\begin{array}{rcl} \frac{|\mathrm{tr}(CX) - (a^Ty + b^Tt)|}{1 + |(a^Ty + b^Tt)|} & < & 1.0 \times 10^{-7} \\ \frac{||A(x) - a||}{1 + ||a||} & < & 1.0 \times 10^{-7} \\ \frac{||A^T(y) + B^T(t) - C - Z||_F}{1 + ||C||_F} & < & 1.0 \times 10^{-7} \\ B^T(t) & \leq & b \\ t & \geq & 0 \\ X, Z & \succeq & 0. \end{array} \tag{23}$$

4 Computational Complexity

In this section we consider the storage requirements and computational complexity of the predictor corrector algorithm. We will consider a problem with n by n matrices, X, Z, and C, and m equality constraints. The analysis is essentially unchanged by the addition of inequality constraints.

In addition to the problem data, our implementation of the algorithm requires one array of size m by m, and twelve arrays of size n by n. Assuming that the constraint matrices are sparse, and assuming that m is much larger than n, the storage required by the system matrix, O, usually dominates the total storage requirements. For example, in computing the Lovasz ϑ number of a graph with 100 nodes and 1000 edges, n is 100, while m is 1,001. In our example, the 1,001 by 1,001 matrix O occupies over 8 megabytes of storage, while the 100 by 100 matrices occupy a total of about 1 megabyte of storage. In this example, there are a total of 1,100 nonzero entries in the constraint matrices.

In practice, the number of iterations required by the algorithm is generally less than 30, and seems to grow slowly with the size of the problem. For that reason, we'll focus on the computational complexity of a single iteration of the algorithm. In the implementation of the semidefinite programming algorithm, there are three computational tasks that are of particular significance:

- Computing the system matrix O, requires $O(m(n^2m + n^3))$ time. This is in the worst case, assuming that the constraint matrices are dense.
- Factoring the system matrix O, requires $O(m^3)$ time.
- Factoring matrices of size n, requires $O(n^3)$ time.

Since m is often much larger than n, computing and factoring the O matrix is usually much harder than various operations on the n by n matrices. In our example, factoring the $1,001$ by $1,001$ matrix O is about $1,000$ times harder than factoring one of the 100 by 100 matrices.

This analysis assumes that the constraint matrices A_i are fully dense matrices. In many cases, these matrices are sparse, and considerable performance improvement is possible in the construction of O. If the individual constraint matrices have $O(1)$ nonzero entries, a simple analysis shows that we can construct O in $O(m(n^2 + m))$ time. Unfortunately, the system matrix O is normally dense, so there is no way to exploit sparsity in factoring O.

Thus if m is somewhat larger than n, and the constraint matrices are sparse, the most difficult part of each iteration is computing the Cholesky factorization of a m by m matrix. In our implementation, we have used routines from the LINPACK or LAPACK libraries to compute this factorization. On many systems, highly optimized versions of the libraries are available. Using such an optimized library can greatly improve the performance of CSDP.

It should also be noted that in some cases it is possible to greatly simplify the computation of $A(X)$, $A^T(y)$, $B(X)$, $B^T(t)$ and O. For example, if the constraints are of the form $X_{i,i} = 1$, $i = 1 \ldots n$, then $A(X) = \mathrm{diag}(X)$, $A^T(y) = \mathrm{diag}(y)$, and

$O = X \circ Z^{-1}$. CSDP allows the user to write routines that implement specialized versions of these operations.

5 Test Problems

In this section, we discuss the solution of a set of test problems taken from [2]. For comparison, we also report results from SDPA version 4.2 [3]. All computations were performed on a Sun Ultra 1/170 workstation under Solaris 2.5.1. For these runs, 256 megabytes of virtual storage were available. A time limit of 12 CPU hours was also enforced.

For these problems, CSDP used an initial solution similar to the one used in [9]. This initial solution has

$$\begin{aligned} X &= \alpha I \\ Z &= \beta I \\ y &= 0 \end{aligned} \tag{24}$$

where

$$\alpha = n \max_k (1 + |a_k|)/(1 + ||A_k||_F) \tag{25}$$

and

$$\beta = (1 + \max(\max_k(||A_k||_F), ||C||_F))/\sqrt{n}. \tag{26}$$

Computational results for the SDPLIB problems are shown in tables 1 through 3. The notation "$>$ 12 hrs" indicates that one of the codes couldn't solve the problem within the 12 CPU hour time limit. The notation "mem" indicates that the problem couldn't be solved within the 256 megabytes of available virtual storage. Problems infp1, infp2, infd1, and infd2 are infeasible problems, so no optimal objective function value is given.

In general, the two codes produced solutions of comparable quality. In some cases SDPA finds a more accurate solution while in other cases CSDP obtains a more accurate solution. There are some problems which CSDP was able to solve but SDPA was not able to solve. For the 83 problems that were solved by both codes, CSDP required roughly 99,000 CPU seconds, while SDPA required roughly 155,000 CPU seconds. CSDP ranged from about 3 times slower than SDPA (on problem ss30) to about 15 times faster than SDPA (on problem gpp250–1). The geometric mean of the ratio of CSDP CPU times to SDPA CPU times was 71%.

Problem	CSDP Time	SDPA Time	CSDP Objective	SDPA Objective
arch0	361.45	278.56	5.66517e-01	5.66517e-01
arch2	350.72	274.23	6.71515e-01	6.71515e-01
arch4	350.87	274.14	9.72627e-01	9.726274e-01
arch8	350.80	430.08	7.05698e-01	7.05698e+00
control1	0.21	0.33	1.778463e+01	1.778463e+01
control2	2.60	2.87	8.300000e+00	8.300000e+00
control3	29.33	15.37	1.36333e+01	1.363327e+01
control4	69.13	55.48	1.9794231e+01	1.979423e+01
control5	271.67	173.06	1.68836e+01	1.68836e+01
control6	589.05	420.96	3.7304e+01	3.73044e+01
control7	1911.79	1059.92	2.06251e+01	2.06251e+01
control8	2866.11	1891.44	2.0286e+01	2.0286e+01
control9	6670.55	3327.05	1.46754e+01	1.46754e+01
control10	11266.10	6386.39	3.8533e+01	3.8533e+01
control11	18660.34	10078.41	3.1959e+01	3.1959e+01
equalG11	17050.98	$>$ 12 hrs	6.29155e+02	N/A
equalG51	22401.03	mem	4.0056e+03	N/A
gpp100	20.36	28.02	-4.494356e+01	-4.494354e+01
gpp124-1	43.06	58.20	-7.3431e+00	-7.34307e+00
gpp124-2	40.34	52.31	-4.6862e+01	-4.686229e+01
gpp124-3	49.99	52.31	-1.53014e+02	-1.530141e+02
gpp124-4	39.18	49.37	-4.1899e+02	-4.189876e+02
gpp250-1	238.27	3465.69	-1.545e+01	-1.54449e+01
gpp250-2	283.99	655.61	-8.18690e+01	-8.1869e+01
gpp250-3	303.64	904.43	-3.035393e+02	-3.03539e+02
gpp250-4	308.19	691.70	-7.4733e+02	-7.4733e+02
gpp500-1	2592.63	7615.40	-2.5321e+01	-2.5321e+01
gpp500-2	2741.75	7369.71	-1.5606e+02	-1.5606e+02
gpp500-3	2807.34	8507.22	-5.1302e+03	-5.130176e+02
gpp500-4	2854.80	6921.48	-1.56702e+03	-1.567019e+03
hinf1	0.10	0.13	2.0326e+00	2.033e+00
hinf2	0.11	0.47	1.0967e+01	1.0967e+01
hinf3	0.22	0.15	5.694e+01	5.69e+01
hinf4	0.16	0.30	2.7476e+02	2.74764e+02
hinf5	0.19	0.15	3.623e+02	3.63e+02
hinf6	0.13	0.17	4.489e+02	4.490e+02

TABLE 1: Computational results for SDPLIB problems.

Problem	CSDP Time	SDPA Time	CSDP Objective	SDPA Objective
hinf7	0.16	0.11	3.90813e+02	3.91e+02
hinf8	0.14	0.15	1.16e+02	1.16e+02
hinf9	0.43	0.71	2.3624926e+02	2.36250e+02
hinf10	0.18	0.43	1.088e+02	1.088e+02
hinf11	0.37	0.81	6.59e+01	6.59e+01
hinf12	0.98	1.36	2e-03	3e-01
hinf13	1.37	1.50	4.5e+01	4.6e+01
hinf14	3.32	1.60	1.30e+01	1.30e+01
hinf15	4.27	2.64	2e+01	3e+01
hinf37	0.44	0.69	2.3624926e+02	2.3625e+02
infd1	2.27	1.62	infeasible	infeasible
infd2	2.24	1.61	infeasible	infeasible
infp1	2.41	1.51	infeasible	infeasible
infp2	2.42	1.54	infeasible	infeasible
maxG11	4753.31	33421.29	6.291648e+02	6.291648e+02
maxG32	mem	mem	N/A	N/A
maxG51	9473.27	mem	4.006256e+03	N/A
maxG55	mem	mem	N/A	N/A
maxG60	mem	mem	N/A	N/A
mcp100	7.12	21.19	2.26158e+02	2.26157e+02
mcp124-1	13.03	40.87	1.419905e+02	1.419905e+02
mcp124-2	13.38	41.28	2.6988017e+02	2.698802e+02
mcp124-3	13.55	41.13	4.677501e+02	4.677501e+02
mcp124-4	13.78	41.39	8.64412e+02	8.64412e+02
mcp250-1	110.14	482.05	3.172643e+02	3.172643e+02
mcp250-2	106.60	467.56	5.319301e+02	5.31930e+02
mcp250-3	113.07	460.65	9.811726e+02	9.81173e+02
mcp250-4	113.92	458.80	1.6819601e+03	1.681960e+03
mcp500-1	1066.92	6204.96	5.981485e+02	5.981485e+02
mcp500-2	1144.44	6425.27	1.070057e+03	1.070057e+03
mcp500-3	1106.41	5913.67	1.8479700e+03	1.847970e+03
mcp500-4	1108.68	6011.22	3.566738e+03	3.566738e+03
qap5	2.48	1.63	-4.36e+02	-4.360e+02
qap6	8.12	6.94	-3.81e+02	-3.8144e+02
qap7	20.83	15.42	-4.25e+02	-4.25e+02
qap8	48.33	44.73	-7.57e+02	-7.57e+02
qap9	141.01	103.24	-1.41e+03	-1.41e+03
qap10	499.98	326.12	-1.09e+03	-1.093e+03

TABLE 2: Computational results for SDPLIB problems.

Problem	CSDP Time	SDPA Time	CSDP Objective	SDPA Objective
qpG11	34223.08	mem	2.4486591e+03	N/A
qpG51	mem	mem	N/A	N/A
ss30	4341.35	1490.72	2.023951e+01	2.02395e+01
theta1	2.13	3.22	2.300000e+01	2.300000e+01
theta2	45.72	53.81	3.287917e+01	3.287917e+01
theta3	382.85	410.87	4.216698e+01	4.216698e+01
theta4	1892.59	2039.09	5.032122e+01	5.032122e+01
theta5	6183.24	7580.55	5.72323e+01	5.723231e+01
theta6	19457.96	21525.76	6.347709e+01	6.347709e+01
thetaG11	37060.79	$>$ 12 hrs	4.000000e+02	N/A
thetaG51	mem	mem	N/A	N/A
truss1	0.04	0.10	-9.00000e+00	-8.999996e+00
truss2	2.40	3.55	-1.2338036e+02	-1.233804e+02
truss3	0.17	0.40	-9.11000e+00	-9.109996e+00
truss4	0.07	0.15	-9.009996e+00	-9.01000e+00
truss5	51.46	49.12	-1.3263568e+02	-1.326357e+02
truss6	96.91	83.65	-9.010014e+02	-9.010014e+02
truss7	21.47	22.44	-9.000014e+02	-9.0000e+02
truss8	486.37	176.76	-1.3311459e+02	-1.331146e+02

TABLE 3: Computational results for SDPLIB problems.

REFERENCES

1. F. Alizadeh, J.-P. Haberly, M. V. Nayakkankuppam, M. L. Overton, and S. Schmieta. SDP-pack user's guide – version 0.9 beta. Technical Report TR1997–737, Courant Institute of Mathematical Sciences, NYU, New York, NY, June 1997.
2. Brian Borchers. SDPLIB 1.1, a library of semidefinite programming test problems. Submitted for publication in Optimization Methods and Software, September 1998.
3. Katsuki Fujisawa and Masakazu Kojima. SDPA (semidefinite programming algorithm) users manual. Technical Report B-308, Tokyo Institute of Technology, December 1995.
4. Christoph Helmberg, Franz Rendl, Robert J. Vanderbei, and Henry Wolkowicz. An interior–point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.
5. Steve Joy, John E. Mitchell, and Brian Borchers. Solving max-sat and weighted max-sat problems using branch–and–cut. Submitted for publication in the Journal of Combinatorial Optimization, February 1998.
6. B. Kågström, P. Ling, and C. Van Loan. Gemm-based level 3 blas: high-performance model implementations and performance evaluation benchmark. Accepted for publication in ACM Transactions on Mathematical Software, 1997.
7. B. Kågström, P. Ling, and C. Van Loan. Gemm-based level 3 blas: portability and optimization issues. Accepted for publication in ACM Transactions on Mathematical Software, 1997.
8. Franz Rendl. *A MATLAB Toolbox for Semidefinite Programming.* Graz University of Technology, 1996.
9. K. C. Toh, M. J. Todd, and R. H. Tutuncu. SDPT3– a MATLAB software package for semidefinite programming. Technical Report TR1177, Cornell University, December 1996.
10. Lieven Vandenberghe and Stephen Boyd. *SP Software for Semidefinite Programming. User's Guide.* K.U. Leuven and Stanford University, October 1994.