The Ongoing Development of CSDP

Brian Borchers
Department of Mathematics
New Mexico Tech
Socorro, NM 87801
borchers@nmt.edu

Joseph Young
Department of Mathematics
New Mexico Tech
(Now at Rice University)

What is SDP?

- Semidefinite programming (SDP) problems are convex optimization problems in which the variable X is a symmetric and positive semidefinite (PSD) matrix.
- If X is a block diagonal matrix then each block of X must be PSD.
- A linear function of the elements of the matrix X is maximized or minimized.
- Additional linear constraints on the elements of X may be included in the problem.
- Linear programming is a special case in which the nonnegative variables are simply 1 by 1 blocks.

Why SDP?

- Historically, research in SDP has grown out of work on interior point methods for linear programming and engineering applications involving eigenvalue optimization.
- Many convex optimization problems can be reformulated as SDPs.
- An SDP relaxation of a nonconvex optimization problem may provide a good bound on the optimal value of the nonconvex problem.
- Interior point methods for linear programming with nonnegativity constraints can easily be generalized to the semidefinite programming problem. Thus SDP can be solved efficiently (in polynomial time.)

The SDP Problem

$$\begin{array}{lrcl} \max & \text{tr}\,(CX) & & \\ (P) & A(X) & = & b \\ & X & \succeq & 0 \end{array}$$

where

$$A(X) = \begin{bmatrix} \text{tr}\,(A_1 X) \\ \text{tr}\,(A_2 X) \\ \cdots \\ \text{tr}\,(A_m X) \end{bmatrix}.$$

Note that

$$\text{tr}(CX) = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{i,j} X_{j,i} = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{i,j} X_{i,j}$$

The Dual Problem

$$
\begin{array}{lrcl}
\min & b^T y & & \\
(D) & A^T(y) - C & = & Z \\
 & Z & \succeq & 0
\end{array}
$$

where

$$A^T(y) = \sum_{i=1}^{m} y_i A_i.$$

The Algorithm

- CSDP implements a predictor–corrector variant of the primal-dual interior point method of Helmberg, Rendl, Vanderbei, and Wolkowicz (1996.) This method is also known as the HKM method, since the same algorithm was discovered by two other groups of authors (Kojima et al., 1997, Monteiro and Zhang, 1997.)
- CSDP uses an infeasible interior point version of the HKM method.
- The basic idea is to apply Newton's method to a system of equations that can be thought of as a perturbed version of the KKT conditions for the primal/dual SDP's or the KKT conditions for a pair of primal and dual barrier problems.

The Algorithm

- The perturbed KKT conditions are

$$\begin{aligned} A^T(y) - Z &= C \\ A(X) &= b \\ XZ &= \mu I \\ X, Z &\succeq 0. \end{aligned}$$

- The equations for the Newton's method step are

$$\begin{aligned} A^T(\Delta y) - \Delta Z &= C - A^T(y) + Z \\ -A(\Delta X) &= b - A(X) \\ Z\Delta X + \Delta Z X &= -XZ + \mu I. \end{aligned}$$

- These equations can be reduced to an m by m symmetric and positive definite system of equations in Δy.

Storage Requirements

- Consider an SDP problem with m constraints, and block diagonal matrix variables X and Z with blocks of size n_1, n_2, ..., n_k.
- The algorithm requires storage for an m by m Schur complement matrix. This matrix is (in most cases) fully dense.
- The algorithm requires storage for several block diagonal matrices with blocks of size n_1, n_2, ..., n_k.
- Blocks of X and related matrices are typically fully dense, while blocks of Z and related matrices may be sparse.

Storage Requirements

- In practice, the constraint matrices A_1, A_2, ..., A_m are typically quite sparse.
- Assuming that the storage required for each constraint matrix A_i is $O(1)$, the storage required by the HKM method is $O(m^2 + n^2)$.
- For example, assuming the constraint matrices are sparse, the storage required by CSDP 5.0 is approximately $8(m^2 + 11(n_1^2 + n_2^2 + \ldots + n_k^2))$ bytes.
- The parallel version of CSDP 5.0 with p processors requires additional storage of $16(p-1)\max(n_1, n_2, \ldots, n_k)^2$ bytes.

Computational Complexity

- Multiplying matrices of size n takes $O(n^3)$ time.
- Factoring matrices of size n takes $O(n^3)$ time.
- For dense constraint matrices, constructing the Schur complement matrix takes $O(mn^3 + m^2n^2)$ time.
- For sparse constraint matrices with $O(1)$ entries, constructing the Schur complement matrix takes $O(mn^2 + m^2)$ time.
- In practice, most problems have $m > n$ and sparse constraint matrices.
- Thus we would expect that the most time consuming steps in the algorithm to be the computation of the elements of the Schur complement matrix and the Cholesky factorization of this matrix.

A Parallel version of CSDP 5.0

- A 64-bit parallel version of CSDP 5.0 has been developed to run on a shared memory multiprocessor using OpenMP.
- The code makes use of parallelized BLAS and LAPACK routines such as IBM's ESSL, or Sun's performance library. Thus the Cholesky factorization of the Schur complement matrix should parallelize efficiently.
- Our first attempt used automatic compiler parallelization. However, the performance of code was unsatisfactory.
- Although the Cholesky factorization was efficiently parallelized, the computation of the elements of the Schur complement matrix was a significant bottleneck.

A Parallel version of CSDP 5.0

- The routine that computes the elements of the Schur complement matrix was rewritten using OpenMP directives.
- The matrix is split into strips, with each processor working on one strip at a time.
- With this change, the computation of the elements of the Schur complement matrix became much more efficient.
- The following computational results were obtained using an IBM p690 system with 1.3GHz processors at the National Center for Supercomputer Applications.

An Example Problem

The hamming_10_2 problem has $m = 23041$ and $n = 1024$.

Run Times	1	2	4	8	16
Elements	2629.3	1401.0	683.5	286.2	148.4
Cholesky	34083.0	17596.0	8704.3	3921.8	2070.7
Other	1693.9	1100.5	696.2	380.5	289.4
Total	38406.2	20097.5	10084.0	4588.5	2508.5

Parallel Efficiency	1	2	4	8	16
Elements	100	94	96	115	111
Cholesky	100	97	98	109	103
Other	100	77	61	56	37
Total	100	96	95	105	96

An Example Problem

The control10 problem has $m = 1326$ and $n_{\max} = 100$.

Run Times	1	2	4	8	16
Elements	172.5	106.7	41.7	16.9	12.0
Cholesky	11.0	6.5	3.3	1.9	0.9
Other	23.0	22.9	16.5	11.3	10.5
Total	206.5	136.1	61.5	30.1	23.4

Parallel Efficiency	1	2	4	8	16
Elements	100	81	103	128	90
Cholesky	100	85	83	72	76
Other	100	50	35	25	14
Total	100	76	84	86	55

An Example Problem

The maxG55 problem has $m = 5000$ and $n = 5000$.

Run Times	1	2	4	8	16
Elements	64.5	32.8	21.5	12.7	5.5
Cholesky	275.5	128.6	71.3	37.8	16.5
Other	7802.2	4943.1	4143.6	4577.2	2475.4
Total	8142.2	5104.5	4236.4	4627.7	2497.4

Parallel Efficiency	1	2	4	8	16
Elements	100	98	75	63	73
Cholesky	100	107	97	91	104
Other	100	79	47	21	20
Total	100	80	48	22	20

Results With Four Processors

Problem	m	nmax	Time	Error	Storage
CH4	24503	324	21956.6	7.7e-09	4.54G
fap12	26462	369	40583.5	4.4e-09	5.29G
hamming_8_3_4	16129	256	2305.7	6.1e-07	1.98G
hamming_9_5_6	53761	512	97388.9	1.3e-07	21.70G
hamming_10_2	23041	1024	10084.0	8.3e-07	4.13G
hamming_11_2	56321	2048	143055.4	1.1e-06	24.30G
ice_2.0	8113	8113	98667.1	5.4e-07	7.86G
LiF	15313	256	4152.0	3.3e-09	1.79G
maxG60	7000	7000	10615.3	2.4e-08	5.85G
p_auss2	9115	9115	248479.8	1.0e-08	9.93G
theta8	7905	400	398.6	2.4e-07	0.51G
theta62	13390	300	1541.5	1.6e-08	1.37G
theta82	23872	400	8533.0	2.4e-08	4.31G

Solving Large Maximum Independent Set Problems

- In 2000, Neil Sloane proposed a collection of challenging MIS problems.
- Although heuristics have found solutions that seem likely to be optimal, the larger problems have not been solved by integer programming or backtracking search methods.
- There is a semidefinite programming bound for the size of the MIS in a graph due to Lovasz.
- Using this bound and CSDP, we were able to obtain upper bounds on the size of the MIS for many of these problems.
- Using CSDP within a branch and bound code, we were able to solve some of the problems to optimality.

The Maximum Independent Set Problem

The SDP bound on the MIS problem is obtained by solving

$$
\begin{array}{rrcl}
\max & \operatorname{tr}(JX) & & \\
 & \operatorname{tr}(IX) & = & 1 \\
(MISR) & X_{i,j} & = & 0 \text{ for each edge } (i,j) \\
 & X & \succeq & 0
\end{array}
$$

where J is the matrix of all ones.

Solving Large Maximum Independent Set Problems

Problem	Nodes	Edges	Heuristic	MISR	MISR Bnd	B&B
1dc.1024	1024	24063	94	95.9847	95	*94
1dc.2048	2048	58367	172	174.7290	174	-
1et.1024	1024	9600	171	184.2260	184	*171
1et.2048	2048	22528	316	342.0288	342	326
1tc.1024	1024	7936	196	206.3042	206	*196
1tc.2048	2048	18944	352	374.6431	374	356
1zc.1024	1024	33280	112	128.6667	128	-
1zc.2048	2048	78848	198	237.4000	237	-
2dc.256	256	17183	7	7.4618	*7	-
2dc.512	512	54895	11	11.7678	*11	-

Future Work

- Although the OpenMP parallel version of CSDP works well, gcc does not yet support OpenMP. It would help to have a parallel version of CSDP that uses pthreads rather than OpenMP.
- For problems where $m = n$, "other" operations dominate the computational effort. These parts of the code need to be better optimized, particularly in the parallel version of the code.
- Currently, CSDP can be called from C, MATLAB, or Octave. CSDP has also been interfaced to the YALMIP modeling package. It should be possible to interface CSDP to CVX. There's a need to incorporate conic optimization into other modeling languages.

Future Work

- The current build system uses conventional Makefiles that often have to be edited, most commonly to specify where the BLAS and LAPACK libraries are located. It would be better to use autotools to automatically configure the Makefiles.
- In some problems, the Schur complement matrix is sparse. For those problems it would help to incorporate a sparse Cholesky factorization routine.
- CSDP does not currently solve second order cone programming problems. This extension could be done, but it would require a sparse Cholesky factorization routine.
- For problems in which the Schur complement matrix is too large for main memory, an out of core Cholesky factorization routine could be implemented.

Getting CSDP

The current stable version of CSDP is version 5.0. You can download the source code for the serial and parallel versions, binaries for Windows and Linux, and a user's guide from

http://www.nmt.edu/~borchers/csdp.html

The software is available under the Common Public License (CPL). Going forward, the CSDP will be developed as a COIN-OR project. You can get the current development version of CSDP at the project web page

http://projects.coin-or.org/Csdp/

Hans Mittelmann's benchmarks comparing SDP solvers can be found at

http://plato.la.asu.edu/pub/sparse_sdp.html