数理解析研究所講究録
第1614巻 2008年 136-143

SDPA Project and New Features of SDPA 7.1.0

Katsuki Fujisawa
Chuo University
Masakazu Kojima **Kazuhide Nakata** **Mituhiro Fukuda**
Tokyo Institute of Technology
Makoto Yamashita
Kanagawa University
Maho Nakata
RIKEN

Abstract The SDPA Project[1] [5] started in 1995 have provided several software packages for solving large-scale Semidefinite Programs(SDPs). The SDPA 6.2.1 [13] attained high performance for large-scale dense SDPs, however, it required much computation time compared with other software packages when the Schur complement matrix is sparse. The SDPA 7.1.0 is completely revised from the SDPA 6.2.1 as follows; (i) great performance improvements on its computation time and memory usage. In particular, it uses and stores less variables internally, and its overall memory usage is less than half of the previous version. (ii) fast sparse Cholesky factorization when the Schur Complement Matrix is sparse. As a consequence, the SDPA can efficiently solve SDPs with a large number of block diagonal matrices or non-negative constraints. (iii) some improvements on its numerical stability due to a better control in the interior-point algorithm. (iv) arbitrary precision arithmetic when the condition numbers of some matrices at any iterate become ill-conditioned. The objective of this paper is to present brief explanations of the SDPA 7.1.0 and its high performance for large-scale dense and sparse SDPs through numerical experiments compared with some other major software packages for general SDPs. We also review the major achievements of the SDPA Project on solving large-scale SDPs.

1. Introduction

The Semidefinite Program (SDP) has recently received much attention of researchers in various fields for the following three reasons: (i) It has been intensively studied in both theoretical and numerical aspects. Especially the primal-dual interior-point method is known as a useful tool for solving large-scale SDPs with accuracy and numerical stability. (ii) It has many applications covering various fields such as combinatorial optimization, polynomial optimization, control and system theory, robust optimization and quantum chemistry. (iii) Several software packages for solving SDPs and related problems (ex. Second-Order Cone Problem : SOCP) are available on the Internet.

The SDPA (SemiDefinite Programming Algorithm) is a C++ implementation of a Mehrotra-type primal-dual predictor-corrector interior-point method(PDIPM) for solving the standard form SDP and its dual. The main objective of the SDPA Project started in 1995 is to implement the PDIPM for SDPs and to solve large-scale SDPs with various

[1] http://sdpa.indsys.chuo-u.ac.jp/sdpa/

characteristics, e.g., sparsity, large number of constraints and so on. We can obtain other implementations of PDIPM such as SeDuMi [10], SDPT3 [11] and CSDP [1] from the Internet.

In recent applications of SDPs, however, we often face some difficulties in solving large-scale SDPs. There are two major time consuming components at each iteration in the PDIPM, even if we exploit the sparsity and/or special structure of the data matrices. The first component is the computation of all upper triangular elements of the Schur complement matrix. The second component is its Cholesky factorization. The SDPARA developed in 2002 replaces these two major bottleneck components of the SDPA by their parallel implementation using MPI and ScaLAPACK. The SDPARA on a parallel computer has the capability to load a large-scale SDP into distributed memory and solves it quickly compared with the SDPA. The SDPARA-C is an integration of two software packages: the SDPA-C which is a variant of the PDIPM using the positive definite matrix completion technique and the SDPARA. The SDPARA-C successfully reduced the required memory space for sparse SDPs with a large-scale matrix variable and/or a large number of equality constraints. In [9, 12], we show through numerical experiments that exploiting sparsity and parallel computation are effective in solving large-scale SDPs.

2. The SDPA Family

In this section, we briefly explain the outline and purpose of each software package which composes the SDPA Family. More detailed information of the SDPA Family is available at the SDPA Web site:

http://sdpa.indsys.chuo-u.ac.jp/sdpa/

All software packages of the SDPA Family are also available from the above Web site.

2.1. The Outline of Each Software Package

1. SDPA [2, 3, 13]

   The SDPA (SemiDefinite Programming Algorithm) is a software package for solving SDPs and plays the central role in the SDPA Family. It is based on the Mehrotra-type predictor-corrector infeasible PDIPM. The SDPA handles the standard form SDP and its dual problem stated in [13]. It is implemented in C++ language and utilizes the BLAS [2] and LAPACK [3] libraries for matrix and vector computations. The SDPA incorporates dynamic memory allocation and deallocation. Therefore, the maximum size of loaded SDPs depends on the size of the computational memory where the SDPA is installed. The SDPA has the following additional features:

   (a) Callable library for the SDPA functions
   (b) Block diagonal and sparse matrix data structures
   (c) Some information on the feasibility and optimality
   (d) Exploiting the sparsity of the data matrices

2. SDPARA [12] (parallel version)

   The SDPARA (SemiDefinite Programming Algorithm PARAllel version) is a parallel version of the SDPA written in C++ language. The SDPARA works as a stand-alone software package and solves SDPs in parallel with the help of the MPI (Message

[2] http://www.netlib.org/blas/
[3] http://www.netlib.org/lapack/

Passing Interface) and ScaLAPACK (Scalable LAPACK) [4]. However, its functions are not available as a callable library. To install and use the package, we assume that users can access to a parallel computer where the MPI is installed on.

3. SDPA-C [8] (with the positive definite matrix completion)
   The SDPA-C (SemiDefinite Programming Algorithm with the positive definite matrix Completion) is a C++ software package which solves sparse SDPs very efficiently (in computation time and memory) if the data matrices of the SDP are large-scale and sparse. For instance, SDPs with number of equality constraints $m < 2000$, matrix sizes $n < 20000$, and 5% of nonzero density. More specifically, the SDPA-C is more efficient when there is a special structure in the aggregated sparsity pattern of the data matrices. The positive definite matrix completion enables the SDPA-C to store only the sparse matrices and perform matrix computations taking advantages of the sparsity. Besides the ATLAS (or BLAS) and LAPACK libraries for matrix computations, it utilizes the METIS [5] and SPOOLES [6] libraries for finding a proper structure of the aggregated sparsity pattern.
4. SDPARA-C [9] (parallel version of the SDPA-C)
   The SDPARA-C is a parallel version of the SDPA-C: a variant of the SDPARA which incorporates the positive definite matrix completion technique. As the SDPA-C, the following packages are necessary to be installed a priori: ATLAS (or BLAS), ScaLAPACK, METIS, SPOOLES, and MPI.
5. SDPA-M [4] (MATLAB interface)
   The SDPA-M provides a MATLAB interface for the SDPA. It shares all the features with the SDPA except that no callable library is provided. The SDPA-M can be combined with your programs in a MATLAB environment.

2.2. SDPA Online Solver

Solving large-scale optimization problems requires a huge amount of computational power as shown in [9, 12]. The SDPARA on the parallel computer is a very useful tool for solving large-scale SDPs. However, not so many users have and/or log on parallel computers. Users must download some source codes from the SDPA Web site and compile them by pre-determined procedures even if they have parallel environments. For that reason, we have developed a grid portal system for some software packages of the SDPA Family. We call the system the *SDPA Online Solver*. One can access it from the following Web site:

http://sdpa.indsys.chuo-u.ac.jp/portal/

Users can easily use the SDPA, the SDPARA and the SDPARA-C without their own parallel computers. Users only have to prepare one Web client PC connected with the Internet.

In addition, we have several plans such as implementation of the IPM for the SOCP, executing the SDPARA on the Grid environment, and rebuilding the SDPA Online Solver with some recent grid computing technologies.

[4] http://www.netlib.org/scalapack/
[5] http://glaros.dtc.umn.edu/gkhome/views/metis/
[6] http://www.netlib.org/linalg/spooles/

3. New Features of SDPA 7.1.0

3.1. The Outline

We provide several software packages for users in optimization areas. There is still plenty of rooms for improvements for the software packages of the SDPA Family. So we have developed and released the new version of the SDPA. The SDPA 7.1.0 enjoys the following policies and features;

1. Extremely sparse problems:
   Note that the Schur Complement Matrix (SCM) is fully dense in general even if data matrices are sparse [5]. However, we found that the SCM often became sparse when we solve extremely sparse problems arising from the polynomial optimization [6, 7]. We need to modify the data structure for the sparse SCM, and employ the sparse Cholesky factorization.
2. Good numerical stability:
   We may adopt the arbitrary precision arithmetic when the condition numbers of some matrices at any iterate become ill-conditioned. We replace some double type variables in source codes of SDPA, LAPACK and BLAS with variables with arbitrary precision defined by the GMP [7] which is a free library for arbitrary precision arithmetic and floating point numbers.
3. Multi-thread computing:
   In the latest version of the SDPA (7.1.0), we also utilizes the GotoBLAS library [8] for computing the matrix-matrix and matrix-vector multiplication, Cholesky factorization of the matrix and so on. The GotoBLAS supports the multi-thread computing on multi-core processor. It dramatically accelerates the SDPA 7.1.0 on the dual/quad-core processor as presented in Section 3.2.

The SDPA 7.1.0 is completely revised from its source code, and there are great performance improvements on its computation time and memory usage. In particular, it uses and stores less variables internally, and its overall memory usage is less than half of the previous version. Also it performs sparse Cholesky factorization when the Schur Complement Matrix is sparse. As a consequence, the SDPA can efficiently solve SDPs with a large number of block diagonal matrices or non-negative constraints. Additional, there are some improvements on its numerical stability due to a better control in the interior-point algorithm.

3.2. Optimized BLAS and Multi-thread Computing

The SDPA 7.1.0 is also implemented in C++ language and utilizes the BLAS [9] and LAPACK [10] libraries for matrix and vector computations. We highly recommend to use optimized BLAS and LAPACK for better performance when using the SDPA 7.1.0, *e.g.*,

- Automatically Tuned Linear Algebra Software (ATLAS) [11].
- GotoBLAS [12].

---

[7] http://www.swox.com/gmp/
[8] http://www.tacc.utexas.edu/resources/software/
[9] http://www.netlib.org/blas/
[10] http://www.netlib.org/lapack/
[11] http://math-atlas.sourceforge.net/
[12] http://www.tacc.utexas.edu/resources/software/

- Intel Math Kernel Library (MKL) [13].

Table 1 and 2 show how the SDPA 7.1.0 performs on several benchmark problems when replacing the BLAS and LAPACK libraries. Typically, the SDPA 7.1.0 with optimized BLAS and LAPACK seems much faster than the one with BLAS/LAPACK 3.1.1. Furthermore, we can receive benefits of multi-thread computing when using SMP and/or multi-core machines.

Table 1: Numerical experiments(SDPA 7.1.0 + BLAS library) : time:sec.(# of iterations)

BLAS library(# of threads)	Prob(1)	Prob(2)	Prob(3)	Prob(4)
BLAS/LAPACK 3.1.1(1)	70.51(36)	128.36(15)	226.24(18)	342.42(20)
ATLAS 3.8.1(1)	53.00(35)	49.86(15)	41.24(18)	182.65(19)
ATLAS 3.8.1(4)	32.76(35)	19.66(15)	18.26(18)	81.73(19)
Intel MKL 10.0.2.018(1)	44.25(35)	39.92(15)	34.63(18)	172.05(21)
Intel MKL 10.0.2.018(2)	32.88(35)	24.43(15)	22.44(18)	110.41(22)
Intel MKL 10.0.2.018(4)	26.61(35)	16.38(15)	16.37(18)	67.78(18)
Intel MKL 10.0.2.018(8)	25.17(35)	13.38(15)	14.83(18)	63.01(19)
GotoBLAS 1.24(1)	42.94(34)	40.37(15)	32.42(18)	160.75(21)
GotoBLAS 1.24(2)	31.81(35)	23.89(15)	21.07(18)	99.76(20)
GotoBLAS 1.24(4)	26.60(36)	14.81(15)	15.59(18)	64.29(18)
GotoBLAS 1.24(8)	23.84(35)	11.57(15)	13.95(18)	67.47(21)

CPU : Intel Xeon 5345 (2.33GHz) × 2 CPUs, 8 cores
OS : CentOS Ver 5.1 64bit
Compiler : Intel (C/C++ & Fortran) 10.1.012

Table 2: Numerical experiments(SDPA 7.1.0 + BLAS library) : time:sec.(# of iterations)

BLAS library(# of threads)	Prob(1)	Prob(2)	Prob(3)	Prob(4)
BLAS/LAPACK 3.1.1(1)	49.48(36)	67.23(15)	143.84(18)	220.38(20)
ATLAS 3.8.1(1)	40.29(35)	37.27(15)	32.14(18)	145.72(20)
ATLAS 3.8.1(2)	29.76(35)	23.22(15)	21.12(18)	99.04(21)
Intel MKL 10.0.2.018(1)	33.39(35)	30.49(15)	26.68(18)	129.72(21)
Intel MKL 10.0.2.018(2)	25.00(35)	18.72(15)	17.56(18)	84.24(22)
GotoBLAS 1.24(1)	33.91(36)	40.23(15)	24.53(18)	109.72(19)
GotoBLAS 1.24(2)	24.52(35)	18.37(15)	16.11(18)	71.53(19)

CPU : Intel Core 2 E8200 (2.66GHz) × 1 CPU, 2 cores
OS : CentOS Ver 5.1 64bit
Compiler : Intel (C/C++ & Fortran) 10.1.012

Prob(1) : Structural Optimization : g1717.dat-s
Prob(2) : Combinatorial Optimization(1) : mcp1000-10.dat-s
Prob(3) : Combinatorial Optimization(2) : theta5.dat-s
Prob(4) : Quantum Chemistry : CH.2Pi.STO6G.pqg.dat-s

[13]http://www.intel.com/cd/software/products/asmo-na/eng/266858.htm

3.3. Benchmarking Results

We have performed numerical experiments among SDPA 7.1.0 [14], CSDP 6.0.1 [15], SDPT3-4.0β [16] and SeDuMi 1.1 [17]. All benchmark problems in Table 3 and 4 are available from the Web site [18]. Note that we omit some numerical experiments when a software package requires much computation time compared with other software packages. In Table 4, SDPA 7.1.0(4) means that SDPA 7.1.0 + GotoBLAS 1.24 uses 4 threads on quad-core processor. Through numerical results, we observe that some features of the SDPA 7.1.0 as follows:

1. If the Schur Complement Matrix (SCM) becomes sparse (Table 3), the SDPA 7.1.0 outperforms other software packages.
2. For dense SDPs (Table 4), the performances of the SDPA 7.1.0 and CSDP 6.0.1 are similar because they are linked against a same optimized BLAS(GotoBLAS 1.24).
3. The SDPA 7.1.0 can receive benefits of multi-thread computing for almost all dense SDPs in Table 4.

3.4. The Major Differences Between the SDPA 6.2.1 and the SDPA 7.1.0

The major differences between the SDPA 6.2.1 and the SDPA 7.1.0 are the followings:

- The source code is completely revised, unnecessary variables are eliminated, and auxiliary variables re-used. Consequently, the memory usage becomes less than half of the previous version.
- It utilizes the sparse Cholesky factorization when the Schur Complement Matrix becomes sparse. For that, it uses the SPOOLES library for sparse matrices [19] to obtain an ordering of rows/columns which possibly produces lesser fill-in. Now, the SDPA can solve much more efficiently SDPs with
  - multiple block diagonal matrices
  - multiple non-negative constraints
- There is a modification in the control subroutine in the interior-point algorithm which improved its numerical stability when compared to the previous version.

Acknowledgments

The authors are grateful to other members of the SDPA Project, Yoshiaki Futakata for developing the SDPA-M, Kazuhiro Kobayashi for improving the SDPA solver and Yasutaka Osada for developing the SDPA Online Solver.

[14] http://sdpa.indsys.chuo-u.ac.jp/sdpa/
[15] http://www.nmt.edu/ borchers/csdp.html
[16] http://www.math.nus.edu.sg/~mattohkc/sdpt3.html
[17] http://sedumi.mcmaster.ca/
[18] http://plato.asu.edu/bench.html
[19] http://www.netlib.org/linalg/spooles/spooles.2.2.html

Table 3: Numerical experiments(Sparse SDP) : time:sec.(# of iterations)

	SDPA 7.1.0	CSDP 6.0.1	SDPT3-4.0β	SeDuMi 1.1
mater-1	0.04(16)	0.06(16)	0.90(19)	0.31(21)
mater-2	0.24(20)	0.40(19)	2.30(19)	0.98(25)
mater-3	1.38(23)	5.23(23)	9.50(23)	4.53(27)
mater-4	7.98(26)	144.06(27)	44.30(30)	30.69(30)
mater-5	24.61(27)	1345.85(29)	113.30(33)	121.17(33)
mater-6	88.99(36)	-	333.40(41)	490.10(36)
r2S_broydenTri600	2.16(20)	1577.47(21)	18.40(18)	11.04(14)
r2S_broydenTri700	2.63(20)	-	22.20(18)	14.83(15)
r2S_broydenTri800	3.08(20)	-	25.60(18)	18.03(14)
r2S_broydenTri900	3.46(20)	-	29.50(18)	22.52(14)
nonc_500	0.72(29)	180.19(31)	8.60(32)	3.65(22)
ros_500	0.60(23)	129.73(23)	8.50(32)	3.03(17)
rabmo	175.38(21)	258.12(28)	747.60(51)	2044.61(21)
trto3	2.73(21)	19.10(39)	47.0(23)	67.09(59)
trto4	23.38(24)	116.56(38)	31.50(29)	840.10(72)
trto5	297.76(22)	1124.85(52)	354.80(29)	-
vibra3	8.64(32)	37.61(43)	12.40(32)	159.31(69)
vibra4	66.86(35)	288.69(50)	96.00(46)	1920.80(95)
vibra5	1024.25(40)	2088.95(60)	1499.90(68)	-

Table 4: Numerical experiments(Dense SDP) : time:sec.(# of iterations)

	SDPA 7.1.0	CSDP 6.0.1	SDPT3-4.0β	SeDuMi 1.1	SDPA 7.1.0(4)
arch8	1.36(26)	1.62(25)	2.60(24)	4.09(28)	1.38(26)
control10	56.08(40)	60.12(28)	51.60(28)	86.84(43)	51.66(40)
equalG11	24.35(18)	73.09(24)	47.10(16)	316.11(16)	10.10(17)
equalG51	50.00(20)	97.96(19)	89.70(18)	1690.10(30)	19.10(18)
gpp250-1	0.93(20)	2.66(25)	2.70(18)	24.56(33)	0.66(19)
gpp250-4	0.93(19)	1.78(20)	2.10(14)	28.93(40)	0.59(17)
gpp500-1	6.91(20)	70.05(58)	16.60(20)	212.53(40)	3.54(20)
gpp500-4	6.96(21)	15.55(21)	14.70(17)	175.73(30)	3.40(19)
maxG11	17.49(16)	17.35(16)	16.90(15)	180.02(13)	7.34(16)
maxG32	243.42(17)	182.87(17)	176.60(16)	-	88.65(17)
maxG51	32.52(16)	48.92(17)	37.30(17)	533.80(16)	13.38(16)
mcp250-1	0.55(15)	0.82(15)	1.10(14)	6.50(15)	0.41(15)
mcp250-4	0.53(14)	0.77(14)	1.20(13)	6.13(14)	0.40(14)
mcp500-1	4.26(16)	4.86(16)	3.80(15)	49.21(16)	2.14(16)
mcp500-4	4.03(15)	6.54(15)	5.80(14)	44.50(14)	2.05(14)
ss30	6.12(22)	8.38(21)	9.90(21)	30.99(27)	5.69(22)
truss8	1.64(20)	0.94(20)	2.70(17)	2.11(23)	2.16(20)
theta4	7.64(18)	7.98(17)	11.70(16)	82.59(16)	4.63(18)
theta5	23.78(18)	26.44(17)	32.90(16)	287.03(16)	12.36(18)
theta6	64.60(18)	72.56(17)	110.40(17)	865.99(16)	29.96(18)

CPU : **Intel Xeon 5635 3GHz** : 2 CPU, 8 cores
OS : RHEL5 64bit
Compiler : Intel C++/Fortran 10.1.012
MATLAB : 7.5.0.338 (R2007b)

References

[1] B. Borchers, CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, **11 & 12** (1999), 613–623.

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

[3] K. Fujisawa, M. Fukuda, M. Kojima and N. Nakata: Numerical evaluation of the SDPA (SemiDefinite Programming Algorithm). In H. Frenk, K. Roos, T. Terlaky and S. Zhang (ed.): *The High Performance Optimization* (Kluwer Academic Publishers, Massachusetts, 1999).

[4] K. Fujisawa, Y. Futakata, M. Kojima, S. Matsuyama, S. Nakamura, K. Nakata and M. Yamashita: SDPA-M (SemiDefinite Programming Algorithm in MATLAB) User's manual — Version 2.00. Research Report B-359, Dept. Math. & Comp. Sciences, Tokyo Institute of Technology, January 2000, revised May 2005.

[5] K. Fujisawa, K. Nakata, M. Yamashita and M. Fukuda: SDPA Project : Solving Large-scale Semidefinite Programs. *Journal of the Operations Research Society of Japan*, **50(4)**, (2007), 278-298.

[6] K. Kobayashi, S. Kim and M. Kojima, Correlative sparsity in primal-dual interior-point methods for LP, SDP and SOCP. Research Report B-434, Dept. Math. & Comp. Sciences, Tokyo Institute of Technology, September 2006.

[7] J. B. Lasserre: Global optimization with polynomials and the problems of moments. *SIAM Journal on Optimization*, **11** (2001), 796–817.

[8] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota: Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results. *Mathematical Programming*, **95** (2003), 303–327.

[9] K. Nakata, M. Yamashita, K. Fujisawa, M. Kojima, A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion. *Parallel Computing*, **32** (2006), 24–43.

[10] J. F. Strum, SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones, *Optimization Methods and Software*, **11 & 12** (1999), 625–653.

[11] M. J. Todd, K. C. Toh and R. H. Tütüncü: SDPT3 – a MATLAB software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, **11 & 12** (1999), 545–581.

[12] M. Yamashita, K. Fujisawa and M. Kojima: SDPARA: SemiDefinite Programming Algorithm paRAllel version. *Parallel Computing*, **29** (2003), 1053–1067.

[13] M. Yamashita, K. Fujisawa and M. Kojima: Implementation and Evaluation of SDPA6.0 (SemiDefinite Programming Algorithm 6.0). *Optimization Methods and Software*, **18** (2003), 491–505.