Math. Program., Ser. B 95: 407–430 (2003)

Digital Object Identifier (DOI) 10.1007/s10107-002-0355-5

H.D. Mittelmann

An independent benchmarking of SDP and SOCP solvers

Received: March 27, 2001 / Accepted: April 5, 2002
Published online: October 9, 2002 – © Springer-Verlag 2002

Abstract. This work reports the results of evaluating all computer codes submitted to the Seventh DIMACS Implementation Challenge on Semidefinite and Related Optimization Problems. The codes were run on a standard platform and on all the benchmark problems provided by the organizers of the challenge. A total of ten codes were tested on fifty problems in twelve categories. For each code the most important information is summarized. Together with the tabulated and commented benchmarking results this provides an overview of the state of the art in this field.

Key words. semidefinite programming – second order cone programming – optimization software – performance evaluation

1. Introduction

1.1. The problems solved

The primal and dual pair of conic optimization problems over a self-dual cone are defined as

$$\begin{array}{lll} & \min \quad \langle c, x\rangle & \max \quad b^T y \\ (P) & \text{s.t.} \quad x \in K & s.t. \quad z \in K \qquad (D) \\ & \quad \mathcal{A}x = b & \quad \mathcal{A}^* y + z = c \end{array}$$

where

- K is a closed, convex cone in a euclidean space X.
- $\mathcal{A} : X \to \mathbb{R}^m$ is a linear operator, and $\mathcal{A}^*$ is its adjoint.
- $b \in \mathbb{R}^m$, and $c \in X$.

In the case of a semidefinite-quadratic-linear program these are defined as follows:

- **The space** X**:** $x \in X \Leftrightarrow x = (x_1^s, \ldots, x_{n_s}^s, x_1^q, \ldots, x_{n_q}^q, x^\ell)$, where
  - $x_1^s, \ldots, x_{n_s}^s$ are symmetric matrices (possibly of various sizes).
  - $x_1^q, \ldots, x_{n_q}^q$ are vectors (again, possibly of various sizes).
  - x^ℓ is a vector.

H.D. Mittelmann: Department of Mathematics and Statistics, Arizona State University, Box 871804, Tempe, AZ 85287-1804, e-mail: mittelmann@asu.edu

This work is supported in part by grant NSF-CISE-9981984

- **The cone** K**:** $x \in K \Leftrightarrow x_j^s \succeq 0\ \forall j,\ x_k^q \succeq_q 0\ \forall k,\ x^\ell \geq 0$, where
  - $u \succeq 0$ means that the symmetric matrix u is positive semidefinite.
  - $v \succeq_q 0$ means that the vector v is in a quadratic cone (also known as the second-order cone, Lorentz cone or ice cream cone) of appropriate size. That is, if $v \in \mathbb{R}^k$, then $v \succeq_q 0 \Leftrightarrow v_1 \geq \|v_{2:k}\|_2$.
  - $w \geq 0$ means that the vector w is componentwise nonnegative.
- **The inner product** $\langle .,. \rangle$**:** For $x, z \in K$

$$\langle x, z \rangle = \sum_{j=1}^{n_s} x_j^s \bullet z_j^s + \sum_{k=1}^{n_q} x_k^q * z_k^q + x^\ell * z^\ell,$$

where
- For matrices a and b, $a \bullet b = \text{trace}\, a^T b = \sum_{i,j} a_{ij} b_{ij}$.
- For vectors a and b, $a * b = a^T b = \sum_i a_i b_i$.

Thus (P) and (D) become

$$\begin{array}{lllll} \min & \sum_{j=1}^{n_s} c_j^s \bullet x_j^s & + \sum_{k=1}^{n_q} c_k^q * x_k^q & + c^\ell * x^\ell & \\ \text{s.t.} & \sum_{j=1}^{n_s} a_{ij}^s \bullet x_j^s & + \sum_{k=1}^{n_q} a_{ik}^q * x_k^q & + a_i^\ell * x^\ell = b_i\ \forall i & (SQLP-P) \\ & x_j^s \succeq 0\ \forall j & x_k^q \succeq_q 0\ \forall k & x^\ell \geq 0 & \end{array}$$

and

$$\begin{array}{ll} \max & \sum_{i=1}^{m} b_i y_i \\ \text{s.t.} & \sum_{i=1}^{m} a_{ij}^s y_i + z_j^s = c_j^s\ \forall j \\ & \sum_{i=1}^{m} a_{ij}^q y_i + z_k^q = c_j^q\ \forall k \\ & \sum_{i=1}^{m} a_i^\ell y_i + z^\ell = c^\ell \\ & z_j^s \succeq 0 \quad z_k^q \succeq_q 0 \quad z^\ell \geq 0 \end{array} \qquad (SQLP-D)$$

Thus the feasible set is a product of semidefinite, quadratic and nonnegative orthant cones, intersected with an affine subspace. It is possible that one or more of the three parts of the problem is absent, i.e., any of n_s, n_q, or the length of x^ℓ may be zero.

The *rotated quadratic cone* is defined as

$$\{v \in \mathbb{R}^k \mid v_1 v_2 \geq \|v_{3:k}\|\}.$$

It is simply a rotation of the usual quadratic cone, but for the purpose of modeling quadratic inequalities, it is more convenient to use, thus several participating codes support this cone.

1.2. Computing errors

Suppose that we are given *approximate* optimal solutions of $(SQLP-P)$ and $(SQLP-D)$, respectively. To compute how far they are from an exact solution pair, we define a norm on X, and a minimum eigenvalue w.r.t. the cone K. Precisely, if $x \in X$, then

$$\|x\| = \sum_{j=1}^{n_s} \|x_j^s\|_F + \sum_{k=1}^{n_q} \|x_k^q\|_2 + \|x^\ell\|_2,$$

where for a matrix a, $\|a\|_F$ is the Frobenius norm of a. Also,

$$\lambda_{\min,K}(x) = \min\left\{\min_{j=1,\ldots,n_s} \lambda_{\min}(x_j^s),\ \min_{k=1,\ldots,n_q} \lambda_{\min,q}(x_k^q),\ \min_h x_h^\ell\right\}$$

where

- for a symmetric matrix a, $\lambda_{\min}(a)$ is the usual smallest eigenvalue of a.
- for a vector a, $\lambda_{\min,q}(a) = a_1 - \|a_{2:k}\|_2$.

Also, we denote by $\|a\|_1$ the absolute value of the largest component of a, for an arbitrary $a \in X$.

Then, for approximate optimal solutions x of $(SQLP-P)$ and (y, z) of $(SQLP-D)$, we define

$$\begin{aligned}
\mathrm{err}_1(x, y, z) &= \frac{\|\mathcal{A}x - b\|}{1 + \|b\|_1} \\
\mathrm{err}_2(x, y, z) &= \max\left\{0, \frac{-\lambda_{\min,K}(x)}{1 + \|b\|_1}\right\} \\
\mathrm{err}_3(x, y, z) &= \frac{\|\mathcal{A}^* y + z - c\|}{1 + \|c\|_1} \\
\mathrm{err}_4(x, y, z) &= \max\left\{0, \frac{-\lambda_{\min,K}(z)}{1 + \|c\|_1}\right\} \\
\mathrm{err}_5(x, y, z) &= \frac{\langle c, x\rangle - b^T y}{1 + |\langle c, x\rangle| + |b^T y|}
\end{aligned}$$

Furthermore, when x and z are both in K, that is $\mathrm{err}_2(x, y, z) = \mathrm{err}_4(x, y, z) = 0$, we also define

$$\mathrm{err}_6(x, y, z) = \frac{\langle x, z\rangle}{1 + |\langle c, x\rangle| + |b^T y|}$$

A few remarks are in order.

- If x and z are both feasible, then in exact arithmetic $\mathrm{err}_5(x, y, z) = \mathrm{err}_6(x, y, z)$.
- As x and z are *approximate* optimal solutions only, we may have $\mathrm{err}_5(x, y, z) < 0$. It is possible that all other error measures being the same, if $\mathrm{err}_5(x, y, z) = -\delta$ with $\delta > 0$ corresponds to a solution that is "worse", than as if $\mathrm{err}_5(x, y, z)$ was δ. Thus, we decided to report $\mathrm{err}_5(x, y, z)$, not merely the maximum of $\mathrm{err}_5(x, y, z)$ and 0 (as it is done in several papers), so as not to suppress any information.
- Several codes do not explicitly maintain z; in this case, one should set

  $$z = c - \mathcal{A}^* y$$

  Of course, then $\mathrm{err}_3(x, y, z)$ will be zero (depending on the accuracy achieved by the computer).

In the error tables below err_1 and err_5 are always listed. err_2 is only given when nonzero and err_6 only when in the digits shown different from err_5. err_6 is not available for SDPA.

From the time the benchmark problems for the Challenge were published until the time the papers for this volume were due we have performed an evaluation of the codes on all the benchmark problems each code could solve. In most cases we had the latest version of the codes which the authors themselves used and on which their contributions to the volume are based. An exception is the code SDPA. Its authors are reporting about work done after release of their software and which is not yet available in coded form. The codes BMPR and BMZ were released once and not updated. All remaining codes were updated at least once, several at the time of the workshop or thereafter. Substantial changes were done with DSDP which evolved from a special code for discrete graph problems to a general purpose SDP solver accepting also standard sparse SDPA input format. Further quite remarkable improvements were applied to SDPT3, BUNDLE, and CSDP.

These benchmark results are meant to yield a rough overview of how the tested codes performed on the same, standard platform, a Sun Ultra 60 with Solaris 8, 2 GB of memory, and a 450 MHz processor. Paging was avoided. In the following section first basic information is given on each participating code. Then, as provided by the authors, a short description follows of the code itself, the stopping criteria used, and the perceived strengths and weaknesses. In the third section the test problems are listed followed by some remarks regarding the performance of the codes. Appended are tables with problem statistics and the benchmark results.

2. The codes

The submitted codes fall naturally into two major groups and several subgroups:

- The first major group is that of primal-dual interior point methods designed for small to medium sized problems.
  - In the first subgroup are the codes SeDuMi, and SDPT3. These codes handle all 3 types of cones. SDPT3 was enabled to handle second order cones during the course of the Challenge.
  - In the second subgroup are SDPA, and CSDP, which are limited to SDP.
  - In the third subgroup are codes not designed for SDP problems but for convex (MOSEK) and nonconvex as well as convex (LOQO) nonlinear optimization. In fact, late in the Challenge an SDP interface for LOQO was provided but it did not solve satisfactorily any of the Challenge problems and these results were not included below. LOQO did solve some smaller SDPLIB problems.
- The second group is that of large-scale SDP codes designed to provide approximate solutions for large scale problems: BMPR, BMZ, BUNDLE and DSDP. The first three of these do not make use of second order derivative information, while DSDP is a dual interior point code.

In the following the codes will be listed in the above order.
The input formats are:

graph: Graph format as provided by the organizers.
Matlab: SeDuMi format in Matlab binary form as provided by organizers.
QPS: extended MPS format as explained in the MOSEK user's guide [21]; this was generated from the Matlab format with a converter provided by E. Andersen.
SDPA: the sparse SDPA format as explained in the SDPA user's guide; problems not provided in this format were converted from Matlab format to SDP formulation with the help of a program provided by B. Borchers.

2.1. SDPA

Authors: Fujisawa, Kojima, Nakata
Version 5.02, 9/2000; **Available:** yes; the software manual and the SDPA source codes can be found at http://www.is.titech.ac.jp/ yamashi9/sdpa/index.html
Key paper: [12]. For implementation and numerical results – [10].
Features: primal-dual method, tested version uses Meschach library
Language, Input format: C++; SDPA
Error computations: yes
Solves: SDP

The SDPA (SemiDefinite Programming Algorithm) is based on a Mehrotra-type predictor-corrector infeasible primal-dual interior-point method [18, 10], and is implemented in the C++ language utilizing the *Meschach* [24] library for matrix computations. The main features are: it is available in a callable library, three types of search directions can be used (H..K..M, NT and AHO), block diagonal and sparse data matrix structures are exploited, and information on infeasibility is provided. SDPA uses a set of parameters which the user can adjust to cope with numerically difficult semidefinite programs.

Stopping criteria:

- $\min\{\|\mathcal{A}x - b\|_\infty, \|\mathcal{A}^* y + z - c\|_\infty\} \leq \epsilon'$ and
- $\frac{|\langle c,x\rangle - b^T y|}{\max\{(|\langle c,x\rangle| + |b^T y|)/2.0, 1.0\}} \leq \epsilon^*$

Typical values of the parameters ϵ' and ϵ^* are $10^{-6} \sim 10^{-8}$. See [10, 12] for more details.

2.2. SeDuMi

Author: Sturm
Version: 1.04, 9/2000;
Available: yes, from http://fewcal.kub.nl/sturm/software/sedumi.html
Key papers: [25, 26]
Features: self-dual embedding, dense column handling
Language, Input format: Matlab+C; Matlab, SDPA, SDPpack
Error computations: yes
Solves: SDP/SOCP

The primal-dual interior point algorithm implemented in SeDuMi [25] is described in [26]. The algorithm has an $O(\sqrt{n}\log\epsilon)$ worst case bound, and treats initialization

issues by means of the self-dual embedding technique of [29]. The iterative solutions are updated in a product form, which makes it possible to provide highly accurate solutions.

The algorithm terminates successfully if the norm of the residuals in the optimality conditions, or the Farkas system with $b^T y = 1$ or $\langle c, x\rangle = -1$, are less than the parameter `pars.eps`. The default value for `pars.eps` is `1E-9`.

Remarks:

- SeDuMi exploits sparsity in solving the normal equations; this results in a benefit for problems with a large number of small order matrix variables, such as the *copositivity*-problems in the Dimacs set.
- However, for problems that involve a huge matrix variable (without a block diagonal structure), the implementation is slow and consumes an excessive amount of memory.

2.3. CSDP

Author: Borchers
Version 3.2, 12/15/2000;
Available: yes, from http://www.nmt.edu/ borchers/csdp.html
Key paper: [3]
Features: infeasible predictor-corrector variant of a primal dual method based on the H..K..M direction
Language, Input format: C; SDPA
Error computations: yes
Solves: SDP

CSDP consists of a callable library and standalone solver for SDP problems. It is not applicable to problems with second order cone constraints. The code uses a predictor–corrector variant of the primal–dual method of Helmberg, Rendl, Vanderbei, and Wolkowicz [15] and Kojima, Shindoh, and Hara [18]. CSDP is suited to the solution of small and medium size SDPs with general structure. The algorithm supports matrices with block diagonal structure as well as linear programming variables which are expressed as a diagonal block within the SDP problem.

CSDP is written in portable ANSI C with calls to subroutines from either the Linpack or LAPACK libraries. The required Linpack routines are supplied with the code. However, improved performance can be obtained by using BLAS and LAPACK libraries that have been optimized for a particular computer.

The stopping criteria are:

- $\frac{|\langle c,x\rangle - b^T y|}{1+|b^T y|} < 10^{-7}$ and
- $\frac{\|\mathcal{A}x - b\|_2}{1+\|b\|_2} < 10^{-7}$ and
- $\frac{\|\mathcal{A}^* y - c + z\|_2}{1+\|c\|_2} < 10^{-7}$.

In addition, CSDP maintains positive definite X and Z matrices at all times. CSDP checks this by computing the Cholesky factorizations of X and Z.

2.4. DSDP

Authors: S. Benson, Ye
Version: 3.2, 11/2000;
Available: yes, from http://www-unix.mcs.anl.gov/ benson/
Key paper: [2]
Features: Dual scaling potential reduction method, rank-1 constraints, Matlab interface, MPI parallel
Language, Input format: C; SDPA
Error computations: yes
Solves: SDP

The DSDP software package is an implementation of the dual scaling algorithm for SDP. Unlike primal-dual interior point methods, this algorithm uses only the dual solution to generate a step direction. It can generate feasible primal solutions as well, but it saves time and memory if this feature is not used. Many large problems have well structured constraints in the form of low rank matrices or sparse matrices. DSDP explicity accounts for these features by using sparse data structures for the dual matrix and the eigenvalue/eigenvector decomposition of the data matrices. Theoretical convergence results exist for feasible starting points, and strong computational results have been achieved using feasible and infeasible starting points. DSDP initially solves the Schur complement equations using the conjugate residual method, and then it switches to the Cholesky method when the conditioning of the matrix worsens.
Stopping criteria:

$$|\langle c, x\rangle - b^T y| \,/(|b^T y| + 1) \leq 10^{-3} \quad \text{and}$$
$$\|\mathcal{A}^* y + z - c\|_\infty \leq 10^{-8}$$

or

$$|b^T y| \geq 10^8 \quad \text{(unbounded dual)}$$

or

stepsizes less than 10^{-3} for more than 10 iterations (dual infeasibility)

2.5. SDPT3

Authors: Toh, Todd, Tütüncü
Version: 3.0, 7/2001;
Available: yes, from http://www.math.nus.edu.sg/ mattohkc/sdpt3.html
Key paper: [27]
Features: primal-dual method, infeasible primal-dual and homogeneous self-dual formulations, Lanczos steplength computation
Language, Input format: Matlab+C or Fortran; SDPA
Error computations: yes
Solves: SDP and SOCP

This code is designed to solve conic programming problems whose constraint cone is a product of semidefinite cones, second-order cones, and/or nonnegative orthants. It employs a predictor-corrector primal-dual path-following method, with either the H..K..M or the NT search direction. The basic code is written in Matlab, but key subroutines in Fortran and C are incorporated via Mex files. Routines are provided to read in problems in either SeDuMi or SDPA format. Sparsity and block diagonal structure are exploited, but the latter needs to be given explicitly.

The algorithm is stopped if:

- primal infeasibility is suggested because $b^T y/\|\mathcal{A}^* y + z\| > 10^8$; or
- dual infeasibility is suggested because $-\langle c, x\rangle/\|\mathcal{A}x\| > 10^8$; or
- sufficiently accurate solutions have been obtained:

$$\text{rel\_gap} := \frac{\langle x, z\rangle}{\max\{1, (\langle c, x\rangle + b^T y)/2\}}$$

and

$$\text{infeas\_meas} := \max\left[\frac{\|\mathcal{A}x - b\|}{\max\{1, \|b\|\}}, \frac{\|\mathcal{A}^* y + z - c\|}{\max\{1, \|c\|\}}\right]$$

are both below 10^{-8}; or
- slow progress is detected, measured by a rather complicated set of tests including

$$\langle x, z\rangle/n < 10^{-4} \quad \text{and} \quad \text{rel\_gap} < 5 * \text{infeas\_meas};$$

or
- numerical problems are encountered, such as the iterates not being positive definite, the Schur complement matrix not being positive definite, or the step sizes falling below 10^{-6}.

Remarks:
- SDPT3 is a general-purpose code based on a polynomial-time interior-point method.
- It should obtain reasonably accurate solutions to problems of small and medium size (for problems with semidefinite constraints, up to around a thousand constraints involving matrices of order up to around a thousand, and for sparse problems with only second-order/linear cones, up to around 20,000 constraints and 50,000 variables), and can solve some larger problems.
- Because it uses a primal-dual strategy, forms the Schur complement matrix for the Newton equations, and employs direct methods, it is unlikely to compete favorably with alternative methods on large-scale problems.

2.6. MOSEK

Author: E. Andersen
Version: pre-2.0, 2/9/2001;
Available: yes, from http://www.mosek.com/
Key paper: [1]
Features: special SOCP algorithm, OpenMP parallel on Sun, threaded on Linux/WinNT,

Matlab interface
Language, Input format: C; QPS (extended MPS), AMPL
Error computations: no
Solves: SOCP

The conic quadratic optimizer implemented in the prerelease version of MOSEK 2 employs the NT search direction. The other main features of the implementation are that it is based on a homogeneous and self-dual model, handles the rotated quadratic cone directly, employs a Mehrotra type predictor-corrector extension and sparse linear algebra to improve the computational efficiency.

For stopping criteria one may consult the author's contribution in this volume.

MOSEK version 2 is fairly similar to SeDuMi except MOSEK

- Is 100% C code for speed, but is callable from MATLAB.
- Has an extensive presolve facility to reduce problem size.
- Exploits special structure in rotated quadratic cones.
- Exploits fixed and upper bounded variables.
- Has better linear algebra for solving the normal equation system.
- MOSEK does not handle the semidefinite cone.

2.7. LOQO

Authors: H. Y. Benson, Vanderbei
Version: 5.04, 8/2000;
Available: yes, from http://orfe.princeton.edu/ loqo/
Key paper: [28]
Features: NLP approach
Language, Input format: C; SDPA, Matlab, AMPL
Error computations: no
Solves: SOCP (SDP)

LOQO is a software package for solving general (smooth) nonlinear optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) = 0, \quad i \in \mathcal{E} \\ & h_i(x) \geq 0, \quad i \in \mathcal{I}, \end{array}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{E}$ is the set of equalities, $\mathcal{I}$ is the set of inequalities, $g : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{E}|}$, and $h : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{I}|}$. It implements an infeasible-primal-dual path-following method and requires that the problem be smooth, that is f and h be twice differentiable, and g be an affine function. Even though LOQO can handle nonconvex problems in general, it performs better with convex problems, where f is convex, g are affine, and h are concave functions.

Stopping criteria

$$\begin{aligned} \|\mathcal{A}x - b\|_2 &\leq 10^{-7} \\ \|\mathcal{A}^* y + z - c\|_2 &\leq 10^{-8} \\ \log_{10}\left(\frac{|\langle c, x\rangle - b^T y|}{|\langle c, x\rangle| + 1}\right) &\leq -8 \end{aligned}$$

– LOQO can handle other types of nonlinear constraints in the problem.
– A weakness results from the use of the Cholesky factorization. LOQO works best with a sparse problem, and it does not exploit the special structure of an SOCP.
– Its SDP approach leads to dense problems.

2.8. BMPR

Authors: Burer, Monteiro
Version: 6/2000;
Available: no; **Key paper:** [4]
Features: Augmented Lagrangian algorithm applied to an NLP formulation of an SDP arising by replacing the primal variable by a low-rank factorization.
Language, Input format: C; graph
Error computations: none
Solves: maxcut, bisection, theta

The code BMPR is an infeasible, nonlinear programming method for solving any standard form primal SDP. So far, BMPR has been tested on the maximum cut SDP relaxation, the bisection SDP relaxation, and the Lovász theta SDP.

The main idea of BMPR is to eliminate the primal positive semidefiniteness constraint $X \succeq 0$ by employing an implicit factorization $X = RR^T$ that is valid for at least one optimal solution (but not necessarily for all feasible solutions). Using a theorem of Pataki, R is chosen to be an $n \times r$ matrix, where r is the smallest integer satisfying $r(r+1)/2 \geq m$ and m is the number of primal constraints (aside from the constraint $X \succeq 0$). A stationary point $\bar{R}$ of the new nonlinear problem is then found using a first-order augmented Lagrangian method, and in practice, such a stationary point reliably gives an optimal SDP solution $\bar{X} = \bar{R}\bar{R}^T$.

The stopping criterion for BMPR is problem dependent. For maxcut, BMPR can be easily altered to a primal feasible method that is terminated once the norm of the gradient is less than 10^{-5}. For bisection and theta, however, BMPR is an infeasible method that is terminated once the norm of the primal infeasibility has been reduced to a value of 10^{-6}, which, in accordance with the augmented Lagrangian algorithm, indicates an approximate stationary point.

– The main strength of BMPR is that it is a first-order algorithm and hence can attack large-scale SDPs; this property is also shared by BUNDLE and BMZ.
– The infeasible nature of BMPR is a strong disadvantage.
– BMPR does not have a formal convergence proof, while both BMZ and BUNDLE do.
– BMPR performs better on all problem classes that it shares with BMZ and BUNDLE. For theta, the Hamming problems are not representative of the generic performance of BUNDLE; see the above technical report.
– BMPR works with density of the dual matrix S (an advantage over BMZ) and moreover works with a small number of columns r (an advantage over BUNDLE). As a result, the iterations of BMPR are extremely fast.

2.9. BMZ

Authors: Burer, Monteiro, Zhang
Version: 5/2000;
Available: no;
Key paper: [5]
Features: special transformation of the dual SDP to an NLP
Language, Input format: C; graph
Error computations: no
Solves: maxcut, FAP, bisection, theta

The code BMZ is a dual feasible descent method to solve SDP problems for which all primal feasible solutions X have the same diagonal. The maximum cut SDP relaxation, the Lovász theta SDP, and the frequency assignment SDP relaxation are examples of problems that can be solved by BMZ.

The main idea of BMZ is to convert the dual SDP into a nonlinear optimization problem over feasible sets of the form $\mathbb{R}^n_{++} \times \mathbb{R}^N$, where n is the size of the primal matrix variable X and N a suitable integer, see [5]. The resulting objective function is nonconvex but can be computed in time and space proportional to the time and space required to perform the Cholesky factorization of a dual feasible slack matrix S, which is typically efficient for large, sparse SDPs. The function's gradient can also be computed efficiently. By employing ideas from interior-point methods, it is possible to optimize the nonconvex function over its open feasible set by following a "central path" towards optimality. Computationally, a first-order nonlinear optimization algorithm is used to fully exploit problem structure while still achieving a reasonable amount of accuracy.

The stopping criterion for BMZ is to terminate once the barrier parameter μ has passed below a specified threshold. For each of the three examples above, the thresholds are as follows: maxcut, 10^{-3}; theta, 10^{-3}; FAP, 10^{-4}.

- The main strength of BMZ is that it is a first-order algorithm and hence can attack large-scale SDPs; this property is also shared by BUNDLE and BMPR.
- That BMZ is a feasible method is a distinct advantage over BMPR, though BUNDLE is also a dual feasible method.
- In our opinion, an advantage of BMZ over BUNDLE is that BMZ seems to handle a large number of constraints more effectively. The current DIMACS results don't exactly show this, but the more recent results in [6] give a different viewpoint.
- A distinct disadvantage of BMZ to both BUNDLE and BMPR is that BMZ works with the Cholesky factor L of S, that is, BMZ must deal with the fill-in of the Cholesky factorization, whereas both BMPR and BUNDLE work only with the density of S itself. There are several instances in which S fills in dramatically, causing a slow-down for BMZ.

2.10. BUNDLE

Authors: Helmberg [Kiwiel, Rendl]
Version: SBmethod1.1.1, 11/2000;
Available: yes, from http://www.mathematik.uni-kl.de/ helmberg/SBmethod/

Key papers: [13, 14, 16, 17]
Features: eigenvalue optimization
Language, Input format: C++; graph
Error computations: norm of subgradient only
Solves: maxcut, FAP, bisection, theta

`SBmethod`, Version 1.1.1, is an implementation of the spectral bundle method [16, 14] for eigenvalue optimization problems of the form

$$\min_{y\in\mathbb{R}^m} \quad a\ \lambda_{\max}(C - \sum_{i=1}^{m} A_i y_i) + b^T y. \tag{1}$$

The design variables y_i may be sign constrained, C and the A_i are given real symmetric matrices, $b \in \mathbb{R}^m$ allows to specify a linear cost term, and $a > 0$ is a constant multiplier for the maximum eigenvalue function $\lambda_{\max}(\cdot)$. The code is intended for large scale problems and allows to exploit structural properties of the matrices such as sparsity and low rank structure (see the manual [13]). Problem (1) is equivalent to the dual of semidefinite programs $\max\{\langle C, X\rangle : X \succeq 0, \langle A_i, X\rangle = b_i \text{ for } i = 1, \ldots m\}$ with constant trace, $\mathrm{tr}X = a$.

The code is a subgradient method in the style of the proximal bundle method of [17]. Function values and subgradients of the nonsmooth convex function $f(y) := a\ \lambda_{\max}(C - \sum_{i=1}^{m} A_i y_i) + b^T y$ are determined via computing the maximum eigenvalue and a corresponding eigenvector by the Lanczos method. Starting from a given point $\hat{y} = y^0$, the algorithm generates the next *candidate* $y^+ := \mathrm{argmin}_y \hat{f}(\cdot) + \frac{u}{2}\|\cdot - \hat{y}\|^2$, where $\hat{f}$ is an accumulated *semidefinite cutting model* minorizing f and the dynamically updated *weight* $u > 0$ keeps y^+ near $\hat{y}$. A *descent step* $\hat{y}^+ = y^+$ occurs if $f(\hat{y}) - f(y^+) \geq \kappa[f(\hat{y}) - \hat{f}(y^+)]$, where $\kappa \in (0, 1)$. Otherwise a *null step* $\hat{y}^+ = \hat{y}$ is made but the next model is improved with the new eigenvector computed at y^+.

The algorithm stops, when $f(\hat{y}) - \hat{f}(y^+) \leq \varepsilon_{\text{term}}(|f(\hat{y})| + 1.)$ for a given termination parameter $\varepsilon_{\text{term}} > 0$ which is 10^{-5} by default.

- The code never factorizes the matrix $C - \sum_{i=1}^{m} A_i y_i$ but uses matrix vector multiplications exclusively. Thus, there is no danger of increasing memory requirements or computational work by additional fill-in.
- Experimentally, the method seems to exhibit fast convergence if the semidefinite subcone that is used to generate the semidefinite cutting surface model, spans the optimal solution of the corresponding primal problem.
- If the subcone, however, is too small, the typical tailing off effect of subgradient methods is observed.

3. The problems, input formats, and the performance of the codes

There are 50 Challenge benchmark problems in 12 groups. For details, see the appendix and the preliminary report [8] available at [9].

- The **torus** set: Max cut problems from the Ising model of spin glasses.
- The **bisection** set: Min bisection problems from circuit partitioning.

- The **fap** set: Min k-uncut problems from frequency assignment.
- The **nql** set: Quadratic problems to compute plastic collapse states: plane strain models.
- The **qssp** set: Quadratic problems to compute plastic collapse states: supported plate models.
- The **filter** set: Mixed SDP/SOCP problems from PAM (pulse amplitude modulation) filter design.
- The **hinf** set: LMI (Linear Matrix Inequality) problems.
- The **truss** set: Truss topology design problems.
- The **antenna** set: Antenna array design problems.
- The **copos** set: Checking a sufficient condition for copositivity of a matrix.
- The **hamming** set: Instances computing the theta function of Hamming graphs for which the exact value is known.
- The **sched** set: Quadratic relaxations of scheduling problems.

In each set there are about four instances; from routinely solvable ones to those at or beyond the capabilities of current solvers. The problem statistics are in Tables 1 and 2. The explanation of the entries is as follows:

- An entry [7; 3x5, 3, 4, 2x6] in the "SDP" column means that the problem has 7 SDP blocks whose sizes are 5,5,5,3,4,6,6 in this order. The meaning of the entries in the "QUADR" column is analogous.
- If an entry in the "opt. value" column has is not accompanied by a mark, or remark, then it has been computed by a primal-dual interior point code. Currently these codes provide the most accurate solutions. Still, these values are approximate, and which solver furnishes the most accurate solution can be told only after looking at the tables with the error measures, namely Tables 5, 10 and 11.
- If an entry is accompanied by the mark '*', then it has been computed by a code designed to obtain approximate solutions to large scale problems (such as BMPR, BMZ, BUNDLE, and DSDP).
- If in addition to the '*' mark, there is a 'lb, not opt' remark, this means that a lower bound on the objective value was computed by BMZ, or BUNDLE. These codes work with fully feasible dual solutions, whose value serves as a reliable lower bound, even when the termination criteria of the codes are not satisfied.
- Having a '(?)' mark means that the listed value is the currently known most accurate one; nevertheless, its accuracy is still not satisfactory, and the true value may be quite different.

As the later tables show, there are numerous examples of the first category but also some practically unsolvable ones, notably `copo68, fap25, fap36`. A problem which is solvable by just two codes is `industry2`. In the case of the `hinf12` and `hinf13` instances the results obtained by the various codes are so different, that we cannot be sure whether these problems have in fact ever been *solved*, hence the "?" in the objective column. All these problems are SDPs. The situation with the SOCPs is completely different: there is one specialized code which very efficiently solves all the Challenge problems. This code is MOSEK, with SeDuMi being a close second.

3.1. SOCP problems

The results are contained in Tables 3, 4 and 5. For all codes the primal objective values are listed. The fastest code in this category is MOSEK, with SeDuMi a close second. MOSEK uses its own input format: extended QPS, which is an extension of the QPS format for quadratic programs which in turn is an extension of the well-known MPS format. All other SOCP problems were given in SeDuMi format, and converted to QPS for input to MOSEK with the help of a converter provided by its author. SeDuMi is a close second on the SOCP problems. Only its memory usage kept it from solving all instances.

LOQO which grew out of a LP/QP code to a general NLP solver, has an AMPL interface and accepts QPS input but only for QP's. More recently, it was equipped with a Matlab-based interface to read SOCP problems and even with an interface to input and solve SDP problems in sparse SDPA format. The SOCP results for LOQO in the tables were obtained with the Matlab interface. If a run was not successful because not all of LOQO's convergence criteria were satisfied, this problem was also solved utilizing the AMPL interface and the QPS to AMPL converted files provided by us per anonymous `ftp`. These results are in parentheses. The LOQO option 'convex' was used and the maximum iteration number increased. It should be noted that LOQO is in continuous development and while here version 5.04 was used in the mean time a filter-based version 6.0 is available.

3.2. SDP problems

The results are contained in Tables 6, 7, 9, 10, and 11. For all codes, except for BMZ, BUNDLE and DSDP the primal objective values are listed. On the SDP side eight codes are competing and the collection of the test problems is naturally more diverse. Three of the codes – BMPR, BMZ, and BUNDLE – do not make use of second order information, and are therefore successful mostly on large-scale combinatorial SDPs.

The important difference between BMPR and the two other codes, BMZ and BUNDLE are as follows. BMPR is a *primal* method, that works with *primal infeasible* solutions, attaining feasibility in the limit only: its objective value is therefore mathematically less reliable, but it produces primal approximations very efficiently. BMZ and BUNDLE are both *dual* methods, that work with *dual feasible* solutions, hence any objective value they furnish is a mathematically reliable lower bound on the objective value, even before their termination criteria have been met. The difference in the approaches – primal vs. dual – explains the differences in the reported objective values in Table 6. We remark that some kind of error computation is possible for these codes as well, but this was not incorporated into our testing, and we refer to the authors' contributions in this volume.

One can draw cautiously the following conclusions concerning the first three codes. BUNDLE outperforms BMZ in the maxcut and bisection classes, generates excellent starting guesses in the class of theta problems and seems consequently very efficient there. However, in the FAP class BMZ is clearly superior to BUNDLE. This becomes

evident from the smallest instance while a further comparison was hampered by the large jump in size to the two larger instances in this class. Taking into account that BUNDLE is applicable to a wider class of SDP problems but that BMZ due to its general approach may still be generalized [7], both methods have their advantages and disadvantages, no "winner" can be declared.

A remark is in order on the results for the FAP problems in Table 7 and the codes BMZ and BUNDLE. Both codes finished with the given times and function values on the two smaller problems. On FAP36 BMZ reached the maximum function value of 63.780 after 1.2×10^7 seconds, then the value dropped slightly to 63.775 and after the given endtime the code crashed, putting out NaN's. BUNDLE, on the other hand, reached the given function value after the given time when the machine crashed and the code was not restarted.

The code BMPR, written by a subset of the BMZ authors, competes very well in the maxcut, bisection, and theta classes. While it does not have a convergence theory, it appears to be the most efficient code for the problems that are part of the Challenge in these classes. BUNDLE's performance in the theta class is largely due to the choice of starting values. It can without doubt be stated that these three first order methods have proven their value for larger instances in certain classes and that further generalizations and improvements may be expected.

It remains to assess the performance of the remaining codes, especially on the SDP problems. Among these, DSDP is the only code employing a dual approach. Of the others, SDPA, SDPT3, and CSDP were run with HKM or XZ directions although the first two allow alternative choices, while SeDuMi utilizes the NT direction or scaling. SDPA is the most traditional code. It has not been updated since 1999 but it performs in a robust way, satisfying all the basic expectations a user would have in the primal-dual approach it is based on. SeDuMi from the start was designed to solve SQL (semidefinite, quadratic, linear) problems while SDPT3 only recently was extended to this class. SeDuMi was slightly upgraded from the currently still distributed version 1.03 to the version 1.04 used for our tests. It employs a number of measures assuring a high numerical robustness while maintaining good efficiency except when the SDP problem involves large matrix variables without block-diagonal structure. Before SDPT3's extension, SeDuMi may have been considered the best broadly, to both SDP and SOCP classes, applicable program.

DSDP is the only dual code which has both advantages and disadvantages. It can exploit certain problem features to reduce storage and increase efficiency. This does not show very much with the limited selection of Challenge problems; in fact, on the larger SDPs DSDP has a comparable performance to CSDP, which is a primal-dual code, but is especially good at handling one large matrix variable. More information about DSDP's performance can be found in [2].

CSDP in its previous version did not have the robustness of the other methods but was stabilized in version 3.2. In several of the cases listed as failed it came close to satisfying the termination criteria. Its memory requirements limit it somewhat but the fact that it is entirely in C or f77 makes it still very economical in that respect and also allows to compile it in 64-bit mode on appropriate platforms. CSDP solved, for example,

the maxcut problems in about the time of DSDP and only 2–4 times slower than the more specialized BMZ.

The code SDPT3 underwent the largest changes except possibly for DSDP during the course of the Challenge. While version 2.1 was comparable to SDPA, it was modified in a minor way for the currently available version 2.2 and in a major way for the version 3.0 the authors asked us to consider. SDPT3-3.0 is a contender in both SDP and SOCP classes and in both smaller and somewhat larger problem sizes. Still, its basic primal-dual strategy and direct linear algebra and the use of Matlab set some limits on problem size.

There were initial difficulties with running SDPT3 under Matlab 6 on a Solaris platform. These lead us to provide additional benchmarking information, see the next paragraph. However, in the mean time these problems seem to have been fixed. As far as one can tell from the limited Challenge numbers, SDPT3-3.0 is comparable though not necessarily faster than SeDuMi on small SDP/SOCP problems. But, it is able to solve much larger cases limited mainly through the memory allocation of Matlab and to do this quite efficiently. This can partly be seen from the results below but was also confirmed through other computations we did.

For an additional comparison on large SDPs, we provide the following additional benchmark result. The case `neosfbr24`, an SDP relaxation of the QAP Nug24, generated by the authors of [23] was solved by us on a single processor of a Sun HPC server E6500 (24 processors, Ultra Sparc II, 400 MHz, 24 GB shared memory) in 6 days and 5 hours using CSDP and in 19 hours and 10 minutes using SDPT3. With default settings, then SDPT3 had reduced the relative duality gap to 1E-9 while CSDP's was 9E-8. For the case `neosfbr25` SDPT3 had memory allocation problems while CSDP solved it in 10 days and 2 hours. Even the very large case `neosfbr30` was solved by the 64-bit version of CSDP; this, however, took nearly eight weeks. The datafiles just mentioned were made available by us through anonymous `ftp`. The NEOS solvers for CSDP and SDPT3 are currently installed on the E6500, while we have installed SDPA, SeDuMi, and MOSEK (extended MPS input for LP/QP/SOCP problems) on another computer. To complete the list of the codes considered here, DSDP and LOQO are also installed at NEOS([22]).

In summary it can be said that the Challenge and this benchmarking effort have provided a good overview of the state of the art in the solution of SDP and SOCP problems. All participating codes proved to be valuable in their own right. Many are successful or even excel in solving a subclass of the problems considered here, others in the solution of classes that extend beyond those. The 'general' SDP respectively SQL codes SDPA and SeDuMi are both reliable and fairly efficient for small to medium problem sizes but both have not been updated, while the codes CSDP and SDPT3 have undergone changes which allow them to solve relatively large SDP respectively SQL cases. We have reason to believe that these as well as several of the other codes will soon be further enhanced and generalized.

Acknowledgements. The work of Gabor Pataki and the referees is gratefully acknowledged. They greatly helped us to present the results in this form. In particular, Tables 1 and 2 and part of the description of the problem class in the introduction were taken from reference [8] with the authors' permission.

4. Appendix:Problem statistics and computational results

Table 1. Problem Statistics: Pure SDPs

NAME	ROWS	SDP	LIN	OPT VALUE
toruspm3-8-50	512	[1; 512]	–	527.808663
toruspm3-15-50	3,375	[1; 3,375]	–	3474.7939
torusg3-8	512	[1; 512]	–	457.358179
torusg3-15	3,375	[1; 3,375]	–	3134.5683
bm1	883	[1; 882]	–	23.43982
biomedP	6,515	[1; 6,514]	–	33.6 *
industry2	12,638	[1; 12,637]	–	65.6 *
fap09	15,225	[1; 174]	14,025	10.8 *
fap25	2,244,021	[1; 2,118]	2,232,141	12.5 * (lb, not opt)
fap36	8,448,105	[1; 4,110]	8,405,931	63.7 * (lb, not opt)
fap-sup-25	322,924	[1; 2,118]	311,044	12.5 * (lb, not opt)
fap-sup-36	1,154,467	[1; 4,110]	1,112,293	63.7 * (lb, not opt)
hamming_9_8	2,305	[1; 512]	–	224
hamming_10_2	23,041	[1; 1,024]	–	102.4
hamming_11_2	56,321	[1; 2,048]	–	170 2/3
hamming_7_5_6	1,793	[1; 128]	–	42 2/3
hamming_8_3_4	16,129	[1; 256]	–	25.6
hamming_9_5_6	53,761	[1; 256]	–	85 1/3
copo14	1,275	[14; 14 × 14]	364	0
copo23	5,820	[23; 23 × 23]	1771	0
copo68	154,905	[68; 68 × 68]	50,116	0
truss5	208	[34; 33 × 10, 1]	–	132.6356779
truss8	496	[34; 33 × 19, 1]	–	133.1145891
hinf12	43	[3; 6, 6, 12]	–	−0.0398 (?)
hinf13	57	[3; 7, 9, 14]	–	−45.476 (?)

Table 2. Problem Statistics: Mixed and Pure SOCPs

NAME	ROWS	SDP/QUADR	LIN	OPT VALUE
filter48	969	[1; 48]/[1; 49]	931	1.416129
filtinf1	983	[1; 49]/[1; 49]	945	primal inf.
minphase	48	[1; 48]/−	–	5.980989
nql30	3,680	−/[900; 900 × 3]	3,602	−0.946028
nql60	14,560	−/[3600; 3600 × 3]	14,402	−0.935423
nql180	130,080	−/[32400; 32400 × 3]	129,602	−0.927717
qssp30	3,691	−/[1891; 1891 × 4]	2	−6.496675
qssp60	14,581	−/[7381; 7381 × 4]	2	−6.562696
qssp180	130,141	−/[65341; 65341 × 4]	2	−6.639527
sched_50_50_orig	2527	−/[2; 2474, 3]	2,502	26,673
sched_100_50_orig	4844	−/[2; 4741, 3]	5,002	181,889
sched_100_100_orig	8338	−/[2; 8235, 3]	10,002	717,367
sched_200_100_orig	18087	−/[2; 17884, 3]	20,002	141,360
sched_50_50_scaled	2526	−/2475	2,502	7.852038
sched_100_50_scaled	4843	−/4742	5,002	67.16628
sched_100_100_scaled	8337	−/8236	10,002	27.33145
sched_200_100_scaled	18086	−/17885	20,002	51.81247

Table 2. Continued.

NAME	ROWS	SDP/QUADR	LIN	OPT VALUE
nb	123	$-/[793; 793 \times 3]$	4	−0.050703
nb_L1	915	$-/[793; 793 \times 3]$	797	−13.01227
nb_L2	123	$-/[839; 1 \times 1677, 838 \times 3]$	4	−1.628972
nb_L2_bessel	123	$-/[839; 1 \times 123, 838 \times 3]$	4	−0.102571

Table 3. CPU times in seconds, SOCP problems; MM: memory exceeded, TT: more than 35 hrs., parentheses: AMPL input

PROBLEM	LOQO	MOSEK	SDPT3	SeDuMi
nb	56	17	42	31
nb_L1	(65)	20	79	38
nb_L2	61	39	83	44
nb_L2_Bessel	28	18	46	22
nql30	(114)	3	12	6
nql60	(1978)	20	66	33
nql180	(TT)	344	MM	MM
qssp30	15	5	24	8
qssp60	151	31	180	109
qssp180	(TT)	531	MM	MM
sched_50_50_orig	23	3	25	11
sched_100_50_orig	52	7	85	31
sched_100_100_orig	378	16	105	62
sched_200_100_orig	459	49	404	272
sched_50_50_scaled	29	3	17	9
sched_100_50_scaled	34	7	41	40
sched_100_100_scaled	138	16	73	239
sched_200_100_scaled	372	49	245	409

Table 4. Optimal values, SOCP problems

PROBLEM	LOQO	MOSEK	SDPT3	SeDuMi
nb	−.050703	−.050703	−.050679	−.050703
nb_L1	−13.0123	−13.0123	−13.0123	−13.0123
nb_L2	−1.6290	−1.6290	−1.6290	−1.6290
nb_L2_Bessel	−.10257	−.10257	−.10257	−.10257
nql30	.94603	.94603	.94605	.94604
nql60	.93505	.93505	.93512	.9351
nql180	–	.9277	–	–
qssp30	6.4967	6.4967	6.4967	6.4967
qssp60	6.5627	6.5627	6.5627	6.5627
qssp180	–	6.6395	–	–
sched_50_50_orig	26673	26673	26671	26673
sched_100_50_orig	181890	181890	181831	181890
sched_100_100_orig	717368	717368	702797	717368
sched_200_100_orig	141360	141360	140863	141360
sched_50_50_scaled	7.85204	7.85204	7.85306	7.85204
sched_100_50_scaled	67.1650	67.1663	67.1736	67.1650
sched_100_100_scaled	27.3308	27.3315	26.9810	27.3308
sched_200_100_scaled	51.8120	51.8125	51.7538	51.8120

Table 5. Error Measures, SOCP problems

PROB/ERROR	SDPT3	SeDu..	PROB/ERROR	SDPT3	SeDu..
nb/1	.18e-4	.86e-12	nb_L1/1	.69e-4	.83e-12
nb/3	.10e-7	0	nb_L1/3	.40e-8	0
nb/4	0	.14e-14	nb_L1/4	0	.57e-13
nb/5	.21e-3	.53e-15	nb_L1/5	.14e-4	.82e-11
nb/6	.22e-3	.88e-13	nb_L1/6	.14e-4	.77e-11
nb_L2/1	.68e-8	.85e-12	nb_L2_Bessel/1	.77e-8	.39e-13
nb_L2/3	.15e-9	0	nb_L2_Bessel/3	.23e-10	0
nb_L2/4	0	.12e-12	nb_L2_Bessel/4	0	.28e-13
nb_L2/5	.11e-7	.26e-10	nb_L2_Bessel/5	.88e-7	.82e-11
nql30/1	.57e-7	.97e-12	nql60/1	.40e-6	.30e-12
nql30/3	.48e-8	0	nql60/3	.10e-7	0
nql30/4	0	.11e-12	nql60/4	0	.14e-13
nql30/5	.17e-4	−.13e-11	nql60/5	.43e-7	−.67e-13
nql30/6	.37e-4	.57e-13	nql60/6	.10e-4	.70e-14
qssp30/1	.72e-7	.70e-12	qssp60/1	.46e-4	.12e-11
qssp30/3	.11e-8	0	qssp60/3	.22e-8	0
qssp30/4	0	.28e-14	qssp60/4	0	.42e-14
qssp30/5	.71e-6	−.51e-13	qssp60/5	.62e-4	.16e-12
qssp30/6	.76e-6	.47e-14	qssp60/6	.20e-4	.10e-13
sched_50_50_orig/1	.67e-3	.51e-6	sched_50_50_scaled/1	.12e-3	.93e-8
sched_50_50_orig/3	.27e-8	0	sched_50_50_scaled/3	.41e-14	0
sched_50_50_orig/4	0	0	sched_50_50_scaled/4	0	.43e-15
sched_50_50_orig/5	−.87e-4	.13e-11	sched_50_50_scaled/5	.16e-4	−.17e-12
sched_50_50_orig/6	.61e-5	.86e-11	sched_50_50_scaled/6	.30e-4	.13e-12
sched_100_50_orig/1	.62e-3	.94e-5	sched_100_50_scaled/1	.83e-3	.37e-7
sched_100_50_orig/2	0	.57e-10	sched_100_50_scaled/2	0	.68e-13
sched_100_50_orig/3	.31e-10	0	sched_100_50_scaled/3	.90e-12	0
sched_100_50_orig/4	0	.57e-10	sched_100_50_scaled/4	0	.21e-13
sched_100_50_orig/5	−.80e-3	−.11e-9	sched_100_50_scaled/5	.12e-3	.67e-11
sched_100_50_orig/6	.19e-5	−.10e-10	sched_100_50_scaled/6	.11e-3	−.10e-12
sched_100_100_orig/1	.52e-1	.15e1	sched_100_100_scaled/1	.44e-1	.11e-5
sched_100_100_orig/3	.11e-9	0	sched_100_100_scaled/3	.21e-13	0
sched_100_100_orig/4	0	.28e-8	sched_100_100_scaled/4	0	.67e-14
sched_100_100_orig/5	−.19e-1	.19e-10	sched_100_100_scaled/5	−.42e-2	.12e-10
sched_100_100_orig/6	.47e-6	.13e-6	sched_100_100_scaled/6	.14e-1	.47e-11
sched_200_100_orig/1	.61e-1	.34e-4	sched_200_100_scaled/1	.28e-2	.12e-6
sched_200_100_orig/2	0	.36e-10	sched_200_100_scaled/2	0	.15e-12
sched_200_100_orig/3	.32e-8	0	sched_200_100_scaled/3	.58e-8	0
sched_200_100_orig/4	0	.41e-10	sched_200_100_scaled/4	0	.80e-13
sched_200_100_orig/5	−.44e-2	−.78e-10	sched_200_100_scaled/5	−.84e-3	.98e-11
sched_200_100_orig/6	.32e-5	−.82e-10	sched_200_100_scaled/6	.72e-3	−.10e-12

Table 6. CPU times in seconds, small and medium SDP problems; S semidefinite, Q quadratic, L linear; MM: memory exceeded; NA: not applicable

PROB/TYPE	BMP	BMZ	BDL	CSD	DSD	SDA	SD3	SeD
toruspm3-8-50/S	6	40	15	99	42	435	80	955
torusg3-8/S	12	47	15	106	43	650	82	1311
bm1/S	53	142	110	fail	1282	6532	749	30661
filter48_socp/SQL	NA	NA	NA	504	fail	2348	53	12
filtinf1/SQL	NA	NA	NA	fail	fail	2008	fail	12
minphase/S	NA	NA	NA	fail	25	2	6	5
truss5/S	NA	NA	NA	4	17	4	7	4
truss8/S	NA	NA	NA	23	222	45	35	27
hinf12/S	NA	NA	NA	fail	1	1	5	1
hinf13/S	NA	NA	NA	fail	1	1	4	1
copo14/SL	NA	NA	NA	54	1004	226	33	30
copo23/SL	NA	NA	NA	3607	fail	38894	1575	5651
copo68/SL	NA	NA	NA	MM	MM	MM	MM	MM
hamming_7_5_6/S	4	360	1	89	115	495	52	365
hamming_9_8/S	15	383	1	328	999	fail	183	1482

Table 7. CPU times in seconds, large SDP problems; S semidefinite, L linear; MM: memory exceeded; *: see fourth paragraph in subsection 3.2

PROB/TYPE	BMPR	BMZ	BUNDLE	CSDP	DSDP
toruspm3-15-50/S	172	4182	462	15857	16450
torusg3-15/S	144	5043	701	16006	17897
fap09/SL	NA	1424	14972	fail	MM
fap25/SL	NA	9.97e5	1.38e6	MM	MM
fap36/SL	NA	1.53e7*	7.33e6*	MM	MM
biomedP/S	5407	fail	17470	MM	MM
industry2/S	9515	MM	101074	MM	MM
hamming_8_3_4/S	139	3207	16	MM	MM
hamming_9_5_6/S	569	63744	8	MM	MM
hamming_10_2/S	616	7580	116	MM	MM
hamming_11_2/S	1423	57541	243	MM	MM

Table 8. Typical Memory Requirements in MB

PROBLEM	BP	BZ	BD	CS	DS	LO	MS	SA	ST	Se
torusg3–8	–	–	–	31	9.4	–	–	82	44	113
torusg3-15	22	44.6	19	1450	205	–	–	–	–	–
fap09	–	5	10	–	–	–	–	–	–	–
bm1	4	3.1	8.7	99	21	–	–	245	263	271
nql30	–	–	–	939	–	259	3	–	–	43
qssp60	–	–	–	–	–	1136	35	–	–	295
nb	–	–	–	276	841	43	14	>335	188	47
copo14	–	–	–	16	26	–	–	18	47	48
ham_7_5_6	1.5	1.8	3.4	28	40	–	–	31	70	201
ham_9_8	3.6	3.4	5	70	103	–	–	118	156	364

Table 9. Optimal values, SDP problems; inf: infeasible

PROB	BMP	BMZ	BDL	CSD	DSD	SDA	SD3	SeD
to.3-8-50	527.65	527.85	527.81	527.81	527.82	527.81	527.81	527.81
to.-15-50	3475.1	3475.3	3475.1	3474.8	3474.8	–	–	–
to.g3-8	457.34	457.38	457.36	457.36	457.36	457.36	457.36	457.36
to.g3-15	3134.6	3134.7	3134.6	3134.6	3134.6	–	–	–
fap09	–	10.642	10.797	–	–	–	–	–
fap25	–	11.682	12.538	–	–	–	–	–
fap36	–	63.775	63.767	–	–	–	–	–
bm1	23.440	23.425	23.438	–	23.421	23.46	23.440	23.439
biomedP	33.601	–	33.600	–	–	–	–	–
industr.2	65.644	–	65.603	–	–	–	–	–
filt._socp	–	–	–	1.4161	–	1.4161	1.4161	1.4161
filtinf1	–	–	–	–	–	inf	inf	inf
mi.phase	–	–	–	–	5.59	5.98	5.98	5.98
truss5	–	–	–	132.64	132.55	132.64	132.64	132.64
truss8	–	–	–	133.11	133.00	133.15	133.11	133.11
hinf12	–	–	–	–	−.98e-6	−2.98e-1	−7.86e-1	−2.31e-2
hinf13	–	–	–	–	−44.34	−47.28	−46.64	−44.38
copo14	–	–	–	−3.5e-9	−7.9e-8	1.e-8	1.e-10	$-3.e$-11
copo23	–	–	–	$-2.e$-8	–	8.e-9	$-1.e$-8	$-2.e$-10
h._7_5_6	42.663	42.676	42.667	42.667	42.67	42.667	42.667	42.667
h._8_3_4	25.60	25.64	25.60	–	–	–	–	–
ha._9_8	224	224.03	224	224	224.01	–	224	224
h._9_5_6	85.334	85.351	256/3	–	–	–	–	–
ha._10_2	102.3	102.67	102.4	–	–	–	–	–
ha._11_2	170.67	171.86	512/3	–	–	–	–	–

Table 10. Error Measures (SDP problems); na: not available; –: does not exist

PROB/ERROR	CSDP	DSDP	SDPA	SDPT3	SeDuMi
toruspm3-8-50/1	.45e-13	.24e-11	.50e-13	.24e-10	.12e-11
toruspm3-8-50/3	.77e-15	0	.53e-15	.61e-15	0
toruspm3-8-50/4	0	0	0	0	.14e-12
toruspm3-8-50/5	.29e-7	.30e-3	.19e-7	.22e-8	.86e-15
toruspm3-8-50/6	–	–	na	–	−.19e-15
toruspm3-15-50/1	.64e-13	.14e-10	–	–	–
toruspm3-15-50/3	.20e-14	0	–	–	–
toruspm3-15-50/4	0	0	–	–	–
toruspm3-15-50/5	.29e-7	.28e-3	–	–	–
torusg3-8/1	.48e-13	.29e-11	.17e-8	.22e-10	.10e-11
torusg3-8/3	.87e-15	0	.56e-15	.74e-15	0
torusg3-8/4	0	0	0	0	.77e-12
torusg3-8/5	.35e-7	.30e-3	.43e-7	.33e-9	.17e-13
torusg3-8/6	–	–	na	–	.19e-13
torusg3-15/1	.21e-12	.14e-10	–	–	–
torusg3-15/3	.20e-14	0	–	–	–
torusg3-15/4	0	0	–	–	–
torusg3-15/5	.24e-7	.28e-3	–	–	–

Table 10. Continued.

PROB/ERROR	CSDP	DSDP	SDPA	SDPT3	SeDuMi
bm1/1	.26e-4	.17e5	.81e-3	.40e-6	.12e-5
bm1/3	.83e-10	0	.30e-12	.82e-11	0
bm1/4	0	.44e-5	0	0	.47e-10
bm1/5	.60e-3	.10e1	.83e-3	.86e-6	.20e-7
bm1/6	.61e-3	.17e-3	na	.88e-6	.98e-7

Table 11. Error Measures (SDP problems), continued

PROB/ERROR	CSDP	DSDP	SDPA	SDPT3	SeDuMi
filter48_socp/1	–	–	.35e-6	.12e-5	.80e-7
filter48_socp/3	–	–	.15e-10	.81e-13	0
filter48_socp/4	–	–	0	0	.90e-10
filter48_socp/5	–	–	.16e-5	.10e-4	.10e-9
filter48_socp/6	–	–	na	.12e-5	.51e-7
hinf12/1	.36e-11	.45e-10	.28e-7	.22e-7	.21e-9
hinf12/3	.35e-6	0	.38e-9	.31e-9	0
hinf12/4	0	0	0	0	0
hinf12/5	−.14e-2	−.96e-6	−.16e0	−.23	−.27e-2
hinf12/6	.46e-3	.90e-7	na	.10e-7	.52e-1
hinf13/1	.20e-5	.28e2	.56e-3	.90e-4	.82e-7
hinf13/2	0	.23e-12	0	0	0
hinf13/3	.32e-9	0	.35e-6	.89e-12	0
hinf13/4	0	0	0	0	0
hinf13/5	−.51e-5	.99e-2	−.16e-1	−.18e-1	−.19e-2
hinf13/6	.57e-3	.20e-3	na	.29e-4	.47e-1
minphase/1	.68e-7	.16e0	.98e-7	.81e-8	.47e-7
minphase/3	.37e-8	0	0	.57e-12	0
minphase/4	0	0	0	0	0
minphase/5	.25e-4	.14	−.68e-3	−.21e-3	−.74e-4
minphase/6	.63e-4	.15	na	.19e-7	,54e-3
truss5/1	.13e-9	.20e-8	.86e-10	.37e-6	.21e-10
truss5/3	.90e-14	0	.22e-13	.88e-14	0
truss5/4	0	0	0	0	.22e-12
truss5/5	.30e-7	.49e-3	.17e-7	−.25e-9	−.33e-13
truss5/6	–	–	na	.13e-6	.10e-10
truss8/1	.14e-8	.20e-8	.16e-9	.31e-5	.49e-11
truss8/3	.93e-14	0	.46e-8	.92e-14	0
truss8/4	0	0	0	0	.43e-14
truss8/5	.24e-7	.44e-3	.67e-8	−.27e-5	−.14e-14
truss8/6	–	–	na	.12e-6	.17e-11
copo14/1	.11e-13	.25e-8	.63e-10	.10e-9	.73e-11
copo14/3	.42e-14	0	.99e-13	.50e-14	0
copo14/4	0	0	0	0	0
copo14/5	.96e-8	.95e-7	.20e-7	−.13e-8	.14e-10
copo14/6	–	.99e-7	na	.77e-9	.53e-10
copo23/1	.21e-13	–	.21e-9	.15e-8	.29e-10
copo23/3	.96e-14	–	.22e-12	.11e-13	0
copo23/4	0	–	0	0	0
copo23/5	.51e-7	–	.11e-7	.76e-7	.57e-10
copo23/6	–	–	na	.54e-9	.33e-9

Table 11. Continued

PROB/ERROR	CSDP	DSDP	SDPA	SDPT3	SeDuMi
hamming_7_5_6/1	.17e-15	.39e-12	.52e-11	.18e-14	.10e-9
hamming_7_5_6/3	0	0	0	0	0
hamming_7_5_6/4	0	0	0	0	.35e-10
hamming_7_5_6/5	.33e-7	.41e-3	.36e-7	.85e-10	−.33e-9
hamming_7_5_6/6	–	–	na	–	.85e-8
hamming_9_8/1	.11e-15	.14e-11	.17e-9	.66e-14	.27e-10
hamming_9_8/2	0	0	.71e-3	0	0
hamming_9_8/3	0	0	0	.81e-13	0
hamming_9_8/4	0	0	0	0	.19e-10
hamming_9_8/5	.62e-8	.49e-13	.10	.37e-8	−.15e-9
hamming_9_8/6	–	–	na	–	.12e-7

References

1. Andersen, E.D., Roos, C., Terlaky, T. (2000): A primal-dual interior-point method for conic quadratic optimization. this volume
2. Benson, S.J, Ye, Y. (2001): DSDP3: Dual scaling algorithm for general positive semidefinite programming. Preprint ANL/MCS-P851-1000, Argonne National Labs
3. Borchers, B. (1999): CSDP, A C library for semidefinite programming. Optimization Methods and Software **11,** 613–623
4. Burer, S., Monteiro, R.D.C. (2001): A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. this volume
5. Burer, S., Monteiro, R.D.C., Zhang, Y. (2001): Solving a class of semidefinite programs via nonlinear programming. Computational and Applied Mathematics, Rice University, Houston, Revised December 1999 and May 2001, submitted to Mathematical Programming
6. Burer, S., Monteiro, R.D.C., Zhang, Y. (2001): A computational study of a gradient-based log-barrier algorithm for a class of large-scale SDPs. this volume
7. Burer, S., Monteiro, R.D.C., Zhang, Y. (2001): Interior–point algorithms for semidefinite programming based on a nonlinear formulation. Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, submitted to Computational Optimization and Applications
8. Pataki, G. and Schmieta, S., The DIMACS library of semidefinite-quadratic-linear programs, Computational Optimization Research Center, Columbia University, New York, NY, USA, 1999.
9. DIMACS 7th Challenge website, http://dimacs.rutgers.edu/Challenges/Seventh/
10. Fujisawa, K., Fukuda, M., Kojima, M., Nakata, K. (1999): Numerical evaluation of SDPA (SemiDefinite Programming Algorithm). High Performance Optimization, Kluwer Academic Publishers, 267–301
11. Fujisawa, K., Kojima, M., Nakata, K. (1997): Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. Mathematical Programming **79**, 235–253
12. Fujisawa, K., Kojima, M., Nakata, K. (2000): SDPA (Semidefinite Programming Algorithm) – User's Manual. Technical Report B-308, Tokyo Institute of Technology, http://is-mj.archi.kyoto-u.ac.jp/~fujisawa/sdpa_doc.pdf
13. Helmberg, C. (2000): SBmethod — a C++ implementation of the spectral bundle method. Manual to Version 1.1, ZIB-Report ZR 00-35, Konrad-Zuse-Zentrum für Informationstechnik Berlin, http://www.mathematik.uni-kl.de/ helmberg/SBmethod/
14. Helmberg, C., Kiwiel, K.C. (1999): A spectral bundle method with bounds. ZIB Preprint SC-99-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, to appear in Mathematical Programming
15. Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H. (1996): An interior-point method for semidefinite programming, SIAM Journal on Optimization **6**, 342–361
16. Helmberg, C., Rendl, F. (2000): A spectral bundle method for semidefinite programming. SIAM Journal on Optimization **10**, 673–696
17. Kiwiel, K.C. (1990): Proximity control in bundle methods for convex nondifferentiable minimization. Mathematical Programming **46**, 105–122
18. Kojima, M., Shindoh, S., Hara, S. (1997): Interior-point methods for the monotone semidefinite linear complementarity problems. SIAM Journal on Optimization **7**, 86–125

19. Mittelmann, H.D. (2002): Decision Tree for Optimization Software. http://plato.la.asu.edu/guide.html
20. Mittelmann, H.D. (2002): Benchmarks for Optimization Software. http://plato.la.asu.edu/bench.html
21. Andersen, E. (2002): MOSEK User's Guide: The MPS file format. http://www.mosek.com/download/doc/html/2/tools/manual/node15.html
22. NEOS Server for Optimization. http://www-neos.mcs.anl.gov/neos/
23. Rendl, F., Sotirov, R., Wolkowicz, H. (2001): Exploiting sparsity in interior point methods: Applications to SDP and QAP. Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada, in progress
24. Stewart, D.E., Leyk, Z. (1994): Meschach: Matrix Computation in C. Proceedings of the Center for Mathematics and Its Applications, The Australian National University, Volume 32
25. Sturm, J.F. (1999): Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optimization Methods and Software **11**, 625–653
26. Sturm, J.F. (2000): Central region method, in High Performance Optimization, J.B.G. Frenk, C. Roos, T. Terlaky, S. Zhang (eds.), Kluwer Academic Publishers, 157–194
27. Tütüncü, R.H., Toh, K.C., Todd, (2002): Solving semidefinite-quadratic-linear programs using SDPT3. this volume
28. Benson, H.Y., Vanderbei, R.J. (2002): Solving Problems with Semidefinite and Related Constraints Using Interior-Point Methods for Nonlinear Programming. this volume
29. Ye, Y., Todd, M.J., Mizuno, S. (1994): An $O(\sqrt{n}L)$-iteration homogeneous and self–dual linear programming algorithm. Mathematics of Operations Research **19**, 53–67