Mathematical Programming manuscript No.
(will be inserted by the editor)

Madhu V. Nayakkankuppam

Solving Large-Scale Semidefinite Programs in Parallel

Dedicated to the memory of Jos Sturm.

the date of receipt and acceptance should be inserted later

Abstract We describe an approach to the parallel and distributed solution of large-scale, block structured semidefinite programs using the spectral bundle method. Various elements of this approach (such as data distribution, an implicitly restarted Lanczos method tailored to handle block diagonal structure, a mixed polyhedral-semidefinite subdifferential model, and other aspects related to parallelism) are combined in an implementation called LAMBDA, which delivers faster solution times than previously possible, and acceptable parallel scalability on sufficiently large problems.

Keywords Semidefinite programming, eigenvalue optimization, subgradient bundle methods, Lanczos method, parallel computing.

Mathematics Subject Classification (2000) 90C22, 90C06, 65F15, 65Y05.

1 Introduction

Background. There has been a recent renewed interest in first order subgradient bundle methods for semidefinite programming (SDP) and eigenvalue optimization owing to the fact that storage requirements and solution time of interior-point methods scale poorly with increasing problem size. In contrast, even for large-scale problems, subgradient bundle methods can produce a solution of low accuracy relatively quickly using only a modest amount of memory. To see this difference clearly, denote by $\mathbb{S}^n$ the space of real, symmetric $n \times n$ matrices equipped with the trace inner product, and consider the standard primal form of an SDP

$$\min_{X \in \mathbb{S}^n} \ \langle C, X \rangle \qquad \text{s.t.} \quad AX = b; \ X \succeq 0, \tag{1}$$

This work was supported in part by NSF grants DMS-0215373 and DMS-0238008.

Computational Finance Group, Bloomberg LP, 731 Lexington Avenue, New York, NY 10022, USA. Email: Madhu.Nayakkankuppam@gmail.com.

where $b \in \mathbb{R}^m$, $A : \mathbb{S}^n \to \mathbb{R}^m : X \mapsto [\langle A_1, X\rangle, \ldots, \langle A_m, X\rangle]^T$ is a linear operator, the matrices $C, X, A_i \ \ (i = 1, \ldots, m) \in \mathbb{S}^n$, and $X \succeq 0$ means that X is constrained to be positive semidefinite. The subproblem of finding a search direction in every iteration of a typical primal-dual interior-point method requires the solution of an $m \times m$ linear system involving the so-called *Schur complement* matrix. It requires $O(m^2n^2 + mn^3)$ operations to form, and $O(m^3)$ operations to factorize, the Schur complement matrix, while storing this matrix and the primal variable X requires $O(m^2)$ and $O(n^2)$ memory respectively. In practice, substantial savings in both storage requirements and computational time may be realized by exploiting sparsity [9, 24], resorting to dual-only methods [1], employing a low-rank factorization technique [3, 4], or using carefully preconditioned iterative methods [2, 19, 31, 32] for this linear system. Nevertheless, as a rule of thumb, serial implementations of primal-dual interior-point methods are limited to problems approximately as large as $n \approx 2,000$ and $m \approx 5,000$, with such an instance requiring a few hundred megabytes of memory and a few hours of solution time on a typical workstation; see [23] for a recent benchmarking of SDP codes.

On the other hand, Helmberg and Rendl [14] observed that whenever the primal feasible matrices X are constrained to have constant trace $a > 0$, the dual of (1) is equivalent to the nonsmooth, convex, unconstrained, eigenvalue optimization problem

$$\min_{y \in \mathbb{R}^m} \ a\,\lambda_{\max}(A^T y - C) - \langle b, y\rangle\,, \tag{2}$$

where

$$A^T : \mathbb{R}^m \to \mathbb{S}^n : y \mapsto \sum_{i=1}^{m} y_i A_i$$

is the adjoint of A, and $\lambda_{\max}(\cdot)$ is the maximal eigenvalue function. Subgradient bundle methods, pioneered by C. Lemaréchal and further developed by K. C. Kiwiel and others, approximate this nonsmooth objective with a small number of cutting planes, each derived from a subgradient of the objective function. Hence, the storage requirements are $O(m)$. The subproblem of determining a search direction typically requires the solution of a quadratic program whose size depends on the number of cutting planes retained. The *spectral bundle method* of [14], obtained by specializing the *proximal bundle method* of Kiwiel [18], has been remarkably successful in solving SDP relaxations — as large as $n \approx 6,000$ and $m \approx 20,000$ — of combinatorial optimization problems, within tens of minutes on a typical workstation.

Summary. In this paper, we describe LAMBDA, a *general-purpose*, *portable*, *parallel* implementation of the spectral bundle method, *i.e.* a code that handles most types of SDP's, particularly those with block diagonal structure (general-purpose), that is based on the Message Passing Interface (MPI) for execution on a variety of parallel platforms (portable), and that operates on distributed data (parallel). Thus LAMBDA is suited for problems which cannot be accommodated within the memory of a single computer, or which cannot be solved within a reasonable time on a single computer, or both. Some preliminary studies to this end were conducted with an earlier prototype [26] based partly on Matlab. Numerical results from the much improved present implementation show that the techniques proposed here

significantly extend current limits on problem sizes, with acceptable scalability of solution times on sufficiently large problems.

Beginning with the proximal bundle method (§2), we describe how the problem is formulated (§3) and how problem data are distributed (§4). We then explain how each step of the algorithm is implemented (§5 – §8), focusing on block structure and parallelism. Numerical results illustrating various aspects of the code are supplied in §9 prior to concluding perspectives in §10.

Notation. As usual, lowercase letters denote vectors and uppercase letters denote matrices. The j^{th} coordinate vector is denoted e_j, and I_n is the $n \times n$ identity matrix. Parentheses are used to refer to diagonal blocks of a block diagonal matrix (*e.g.* $C(k)$ for the k^{th} diagonal block of C) or of a block vector. A superscript may denote an iteration index or an exponent, with usage to be inferred from context. Subscripts index an element in a set (*e.g.* A_i $(i = 1\ldots,m)$), an entry in a vector, a column vector, or an entry in a matrix named with the same uppercase letter (*e.g.* $P = [p_1,\ldots,p_r]$, or c_{ij} for the (i,j) entry of C, or $c_{ij}(k)$ for the (i,j) entry of $C(k)$); again, usage should be evident from context. Additionally, some Matlab-like array indexing notation (*e.g.* $1:n$ for indices 1 through n), as well as some Matlab built-in function names are employed in pseudocode for algorithms. The $\text{svec}(\cdot)$ operator isometrically maps a symmetric matrix to a vector. Throughout, we use the 2-norm on vectors.

2 The Algorithm

In this section, we describe the basic ideas behind the proximal bundle method of Kiwiel [18], and its adaptation to the formulation (2) by the spectral bundle method.

2.1 The Proximal Bundle Method

Consider the minimization of a finite-valued, convex function $f : \mathbb{R}^m \to \mathbb{R}$. Although f may be nondifferentiable, it is well known that its Moreau-Yosida regularization

$$f_\rho(x) = \min_{y\in\mathbb{R}^m} \left\{ f(y) + \frac{\rho}{2}\,\|y-x\|^2 \right\} \quad (\rho > 0) \tag{3}$$

is a Lipschitz continuously differentiable convex function with the same set of minimizers as f [15]. The unique minimizer $\tilde{x}$ in (3), called the proximal point of x, satisfies

$$\nabla f_\rho(x) = \rho\,(x-\tilde{x})\,,$$

and hence represents a displacement of x along the negative gradient of f_ρ, with ρ^{-1} being a step length. The proximal point algorithm for minimizing f is then to minimize f_ρ by updating the current point x to its proximal point $\tilde{x}$.

The proximal bundle algorithm approximates $\tilde{x}$ by replacing the $f(y)$ term in the right hand side of (3) with a *cutting plane model* based on the subdifferential [15, 29] of f. Given a current iterate y^k and some subset of past iterations $J^k \subseteq \{1,\ldots,k\}$, the so-called "bundle", namely function values $f(y^j)$ $(j \in J^k)$ and

a set $G^k = \left\{ g^j \in \partial f(y^j) \ : \ j \in J^k \right\}$ of subgradients, furnishes a polyhedral cutting plane model

$$\hat{f}^k(y) = \max_{j \in J^k} f(y^j) + \left\langle g^j, y - y^j \right\rangle, \tag{4}$$

which minorizes f. Using $\hat{f}^k(y)$ from (4) in lieu of $f(x)$ and setting $x = y^k$ in (3) yields

$$\min_{y \in \mathbb{R}^m} \ \left\{ \max_{j \in J^k} f(y^j) + \left\langle g^j, y - y^j \right\rangle \right\} + \frac{\rho}{2} \left\| y - y^k \right\|^2 \tag{5}$$

as the subproblem, which is readily formulated as a quadratic program. By solving this program, one obtains the optimal subgradient g^j in (5), call it $\bar{g}$, as well as the unique minimizer $\bar{y}$ which approximates the proximal point of y^k. If $\bar{y}$ meets the 'sufficient descent' criterion

$$f(\bar{y}) \leq f(y^k) + m_L \left(\hat{f}^k(\bar{y}) - f(y^k) \right) \tag{6}$$

for a given constant $m_L \in (0, 1/2)$, it is accepted as the new iterate by setting $y^{k+1} = \bar{y}$; this is termed a *serious step*. Otherwise, $y^{k+1} = y^k$, and we have a *null step*. In either case, the cutting plane model G^k is improved by adding $\bar{g}$ and a new subgradient $g \in \partial f(\bar{y})$ to yield G^{k+1}, thus making a serious step more likely when (5) is solved in the next iteration. The stopping criterion

$$f(y^k) - \hat{f}^k(\bar{y}) \leq \delta \left(\mid f(y^k) \mid + 1 \right) \tag{7}$$

tests if the upper bound on the maximum attainable descent in this step (left hand side of (7)) falls below a small fraction δ of the magnitude of the objective value (right hand side of (7)), in which case y^k is returned as the computed solution.

Two more steps need to be addressed to complete the description of the algorithm. First, to keep storage bounded, less significant subgradients in G^k may be lumped together into an *aggregate subgradient*. Second, for good practical performance, the penalty parameter ρ has to be judiciously varied at every iteration, as was already recognized in the early computational works of Lemaréchal. Both issues have been addressed by Kiwiel [17, 18], but we defer their discussion to §8, and outline the essence of the method in Algorithm 2.1.

The convergence of the algorithm, which requires only that the updated bundle contain $\bar{g}$ and one new subgradient from $\partial f(\bar{y})$, may be summarized as follows.

Theorem 1 (Kiwiel [17, 18]) *Assume that Algorithm 2.1 is executed with $\delta = 0$ in the stopping test* (7)*. If the algorithm performs a finite number of serious steps, i.e. performs only null steps from some iteration k onwards, then $y^k \in \arg\min f$. If the algorithm generates an infinite sequence of serious steps, then $\left\{ y^k \right\}$ is a minimizing sequence for f, and is convergent to a minimizer of f, if one exists.*

2.2 The Spectral Bundle Method

From an algorithmic standpoint, it is convenient and more general to treat the exact subgradients $g^j \ \ (j \in J^k)$ as ε-subgradients of f at y^k for an appropriate $\varepsilon \geq 0$, and to then view the cutting plane model (4) as being derived from a particular inner

Algorithm 2.1 (Proximal Bundle Method)

1: **(Input)** Given a starting point y^0, an improvement parameter $m_L \in (0, 1/2)$, a termination parameter $\delta > 0$, a penalty parameter $\rho^0 \geq \rho_{\min} > 0$, where $\rho_{\min}$ is a lower bound for the ρ^k sequence.
2: **(Output)** An approximate minimizer $\tilde{y}$ to (2).
3: **(Initialization)** Set the iteration counter $k = 0$. Compute a subgradient $g^0 \in \partial f(y_0)$, and set $G^0 = \{g^0\}$.
4: **for** $k = 0, 1, \ldots$ until termination **do**
5: **(Subproblem)** Solve (5) to obtain $\bar{g}$ and $\bar{y}$.
6: **if** (6) holds **then**
7: Set $y^{k+1} = \bar{y}$. {serious step}
8: **else**
9: Set $y^{k+1} = y^k$. {null step}
10: **end if**
11: **if** (7) is satisfied **then**
12: Stop. Return $\tilde{y} = y^k$ as the computed solution.
13: **end if**
14: **(Update)** Add $\bar{g}$ and a new subgradient $g \in \partial f(\bar{y})$ to G^k, and perform aggregation (if necessary) to obtain G^{k+1}. Update ρ^k to its new value ρ^{k+1}. (Details are deferred to §8.)
15: **end for**{—- k loop —-}

approximation of the ε-subdifferential of f at y^k; this opens up the possibility of choosing other inner approximations designed to better exploit the structure of f and its ε-subdifferential. Each choice for such an inner approximation yields a different model for the ε-subdifferential of f with its own 'bundling process', yet the resulting algorithms can all be analyzed within the same unified framework.

The *spectral bundle method* of Helmberg and Rendl [14] specializes the proximal bundle method to the objective function in (2) using an ε-subdifferential model tailored to exploit the structure of the maximal eigenvalue function. There is no loss of generality in assuming $a = 1$, so let us consider the objective function

$$f(y) = \lambda_{\max}(A^T y - C) - \langle b, y \rangle . \tag{8}$$

Rayleigh's variational formulation of the maximal eigenvalue

$$\lambda_{\max}(Z) = \max_{W \succeq 0;\, \mathrm{tr}(W)=1} \langle W, Z \rangle$$

as the support function of the compact, convex set $\{W \succeq 0 \,:\, \mathrm{tr}(W) = 1\}$ leads to the well known [6, 15, 16, 27] ε-subdifferential formula

$$\partial_\varepsilon \lambda_{\max}(Z) = \{W \succeq 0 \,:\, \mathrm{tr}(W) = 1;\ \langle W, Z \rangle \geq \lambda_{\max}(Z) - \varepsilon\} \tag{9}$$

for $\varepsilon \geq 0$. Exact and ε-subgradients $W \in \partial_\varepsilon \lambda_{\max}(Z)$ are obtained by taking $W = pp^T$, with p being a normalized eigenvector to the maximal eigenvalue (exact subgradient) or a nearly maximal eigenvalue (ε-subgradient) of Z. A standard chain rule from nonsmooth calculus [15] then provides

$$\partial_\varepsilon f(y) = \left\{ AW - b \,:\, W \in \partial_\varepsilon \lambda_{\max}(A^T y - C) \right\} . \tag{10}$$

Thus eigenvectors corresponding to the maximum eigenvalue of $Z = A^T y - C$ give rise to exact subgradients ($\varepsilon = 0$) of f, while eigenvectors corresponding to nearly

maximum eigenvalues yield ε-subgradients of f. By a harmless abuse of terminology, we refer to both exact and ε-subgradients as simply 'subgradients', dropping the ε prefix in the sequel. In similar vein, we loosely use the term 'maximal eigenpairs' to denote the largest few eigenvalues of a matrix and their corresponding eigenvectors, the latter being called 'maximal eigenvectors'.

At a given iteration k, suppose $P^k = [p_1, \ldots, p_r]$ is an orthonormal matrix whose r columns are maximal eigenvectors of $A^T y - C$ at the current iterate $y = y^k$, and possibly at some past iterates $y = y^j$ $(j \in J^k)$. Whereas the traditional proximal bundle method employs a polyhedral cutting plane model G^k (see above (4)) with a typical subgradient of the form $A(p_i p_i{}^T) - b$, a novel feature of the spectral bundle method is the *semidefinite cutting plane model*

$$G^k = \left\{ AW - b \,:\, W = \alpha \overline{W} + P^k V (P^k)^T,\ \alpha \geq 0,\ V \succeq 0,\ \alpha + \mathrm{tr}(V) = 1 \right\},$$

where $\overline{W}$ represents an aggregate subgradient.

In terms of computation, the evaluation of the objective function and its subgradients amounts to computing maximal eigenpairs of the matrix $Z = A^T y - C$. The Lanczos method (further details in §5) efficiently computes the largest eigenpair when intrinsic structure in problem data allows fast multiplications by Z. In fact, incurring only a slightly larger computational expense, the method can extract several leading eigenpairs, provided these eigenvalues are not too closely clustered. Although such a clustering is inevitable as the iterates y^k approach a minimizer of f, the Lanczos method provides an effective way to add several ε-subgradients in Step 14 (Update) of the algorithm at least in the early stages, when the maximal eigenvalue of Z is typically well separated from the rest of the spectrum. The net result is an enhanced subdifferential model and convergence in significantly fewer iterations, with a slightly increased cost per iteration.

Furthermore, the dominant expense of computing maximal eigenvectors of Z is better justified by using them within a rich semidefinite cutting plane model, rather than within a crude (by comparison) polyhedral model. Consequently, the subproblem — traditionally a quadratic program — now becomes a conic program involving linear, convex quadratic, and semidefinite constraints. However, this conic program is much smaller than the original semidefinite program, and can be solved quickly with an interior-point method. The additional computational expense incurred in solving this conic program is amply compensated by convergence in many fewer iterations.

Beginning with the problem formulation in the next section, we describe how each step of the algorithm may be parallelized and adapted to block structured semidefinite programs.

3 Problem Formulation

LAMBDA treats SDP's with the usual block diagonal structure, *i.e.* the data matrices C, A_i $(i = 1, \ldots, m)$ are block diagonal with block sizes $n_1, \ldots, n_s$, and $\mathbb{S}^n$ in (1) now denotes by $\mathbb{S}^{n_1} \times \ldots \times \mathbb{S}^{n_s}$. For SDP's whose primal feasible matrices may not have constant trace, LAMBDA allows the user to specify an upper bound on the trace of some optimal primal solution. This bound has to be inferred by the

user based on problem data, or must be guessed large enough so that at least one primal solution satisfies the bound. In this case, the primal problem (1) may be rewritten as

$$\min_{X\in\mathbb{S}^n} \ \langle C, X\rangle \qquad \text{s.t.} \quad AX = b;\ \ \mathrm{tr}(X) \le a;\ \ X \succeq 0,$$

and its dual as

$$\min_{y\in\mathbb{R}^m} \ a \max\left\{\lambda_{\max}(A^T y - C),\ 0\right\} - \langle b, y\rangle\,. \tag{11}$$

This is transformed without loss of sparsity to the form in (2) by adding an extra 1×1 block to the X, C, A_i $(i = 1,\ldots,m)$ matrices:

$$X' = \begin{bmatrix} X & 0 \\ 0 & x \end{bmatrix}, \quad C' = \begin{bmatrix} C & 0 \\ 0 & 0 \end{bmatrix}, \qquad A_i' = \begin{bmatrix} A_i & 0 \\ 0 & 0 \end{bmatrix} \ \ (i = 1,\ldots,m),$$

resulting in the final form in which the SDP is stored internally. We assume that $a = 1$ and use the simplified form (8) in the remainder of the paper.

4 Data Distribution

LAMBDA distributes the matrices C, A_i $(i = 1,\ldots,m)$ and the entries of the vector b_i to multiple processors in the following simple way. If p processors are available, the index set $\{1,\ldots,m\}$ is partitioned into p mutually disjoint subsets:

$$\{1,\ldots,m\} = M_0 \cup \ldots \cup M_{p-1}, \tag{12}$$

and processor k is allocated the matrices A_i $(i \in M_k)$, and that portion of the vector b with entries b_i $(i \in M_k)$. Processor 0, designated the root (master) processor, also holds the matrix C in addition to its share of the A_i $(i \in M_0)$ matrices. This admittedly simple scheme offers some key advantages. First, distributing entire matrices (rather than portions thereof) facilitates a simple implementation. The scheme may be consistently applied to all problems, whether block structured or not, whereas partitioning within a matrix involves some detailed book-keeping, especially when such partitions straddle block boundaries. Second, scalability relies on the requirement that m (the number of primal constraints) is a large multiple of the number of available processors, while n_i (the block sizes) may be modest in comparison; this is usually the case in many applications. Third, based on sparsity or presence of other structure in the data, it is easy to determine *a priori* a suitable partitioning in (12) that ensures a good load balance among the processors. (Some limitations in this regard are mentioned in §10.) Fourth, by parallelizing the application of the $A(\cdot)$ and the $A^T(\cdot)$ operators, the computationally most intensive parts of the algorithm (subgradient computation, forming the data defining the subproblem) may be effectively parallelized, leaving the serially executed portion (solution of the subproblem, bundle update *etc.*) at an acceptable level. The root processor bears the additional overhead of executing the serial portion. Finally, as a fringe benefit, this scheme conserves some memory by allowing distributed storage of the polyhedral part (details in §7) of the subgradient bundle.

5 Subgradient Computation

Since maximal eigenvectors of $Z = A^T y - C$ furnish ε-subgradients of f at y, the Lanczos method may be used to compute maximal eigenpairs of Z — a matrix which inherits the combined sparsity of the $C, A_i\ (i = 1,\ldots,m)$ matrices. LAMBDA implements an *implicitly restarted block structured Lanczos method* with an *active block strategy*. We provide only a brief overview of the implicitly restarted Lanczos method, referring to the excellent exposition in [20, 28] for full details. Our focus here is rather on efficiently handling block structure.

5.1 The Implicitly Restarted Lanczos Method

The Lanczos method is essentially a Rayleigh-Ritz procedure that approximates invariant subspaces of $Z = A^T y - C$ from a sequence $K^j(Z, v^1)\ (j = 0, 1, \ldots)$ of expanding Krylov subspaces

$$K^j(Z, v^1) = \text{Span}\left\{v^1, Zv^1, Z^2v^1, \ldots, Z^{j-1}v^1\right\},$$

where v^1 is a vector of unit norm with a nontrivial component in the desired eigenspace. The method uses Z only via matrix-vector products, and is particularly effective if the structure in Z allows these products to be quickly computed. Each step of the method involves augmenting a *Lanczos factorization*, which consumes progressively increasing storage and computing time. The speed of convergence improves if v^1 has a large component in the maximal eigenspace and the maximal eigenvalue is well separated from the rest of the spectrum, but in practice, some form of restarting must be employed. If p maximal eigenpairs are desired, a simple scheme would be to restart the Lanczos process every p steps, using the last Lanczos vector v^p as the starting vector v^1 for the next round of iterations. At this stage, spectral transformations (such as Chebyshev acceleration) may be employed to enhance the component of v^1 in the space spanned by the desired eigenvectors, and hence to speed up convergence.

In our present setting, the matrix $Z = A^T y - C$ is distributed. Inter-processor communication adds overhead to matrix-vector products, making spectral transformations via explicit use of matrix polynomials quite expensive. Hence we have adopted *implicit restarting* [30], a technique by which a Lanczos factorization of length $p + d$ is compressed to one of length p in a way that retains the information most relevant to the desired eigenvectors of Z. The ARPACK Users' Guide [20] clearly explains how this procedure is tantamount to a spectral transformation with a filter polynomial of high degree.

In exact arithmetic, the method generates an expanding orthogonal basis — the Lanczos basis — for the nested Krylov subspaces $K^j(Z, v^1)\ (j = 0, 1, \ldots)$. With rounding errors, periodic reorthogonalization is necessary [28]. LAMBDA uses *complete reorthogonalization* of the Lanczos basis prior to each restart.

5.2 Handling Block Diagonal Structure

When the matrices $C, A_i\ \ (i = 1,\ldots,m)$ are block diagonal, $Z = A^T y - C$ and its eigenvectors inherit the same block diagonal structure, and it is important to ex-

ploit this structure in all stages of the algorithm. To the see the inadequacy of the standard Lanczos method, let s be the number of diagonal blocks, and denote the k^{th} block $(k = 1,\ldots,s)$ of these matrices by $C(k), A_i(k)$ $(i = 1,\ldots,m)$ and $Z(k)$ respectively. Suppose p maximal eigenpairs of Z are desired. On the one hand, the naïve approach (call this Method 1) of computing the p largest eigenvalues of each diagonal block $Z(k)$ retains block structure, but is obviously inefficient, since ps eigenpairs are computed. On the other hand, a Lanczos process that treats the entire block diagonal matrix Z as a single linear operator on $\mathbb{R}^{n_1+\ldots+n_s}$ can produce exactly p maximal eigenpairs (call this Method 2), but will likely converge to Ritz vectors (and consequently, eigenvectors) without block structure when the associated maximal eigenvalue is multiple, and split across different blocks of Z. For example, if Z has a multiple maximum eigenvalue that occurs in every block, then the Lanczos method will likely produce an associated eigenvector that is fully dense. Although such eigenvectors, whether possessing block structure or not, yield legitimate ε-subgradients, the loss of block structure degrades the overall efficiency of the algorithm.

Fortunately, this can be rectified by incorporating simple modifications into the Lanczos method; for ease of reference, let us call the modified algorithm the Block Structured Lanczos Method.[1] Here we perform a *synchronized* set of s Lanczos processes, one for each block, maintaining Lanczos vectors independently for each block (akin to Method 1). However, none of these processes is allowed to completely resolve p maximal eigenpairs of any block of Z. Instead, after $p+d$ steps, the restarting process uses Ritz values from *all* blocks to determine unwanted eigenvalues, and hence also implicit shifts for the entire matrix Z (akin to Method 2). Thus the proposed method may be viewed as a hybridization of Methods 1 and 2, with the restarting procedure synchronizing the independent Lanczos processes. Equivalently, it is a method that incorporates block structure in the Lanczos basis. This method requires storage for $p+d$ Lanczos vectors of length $n_1+\ldots+n_s$ (plus a bit more for some auxiliary arrays) and, upon convergence, produces p (approximate) maximal eigenpairs of Z, not counting multiplicities, *i.e.* the method may resolve only one copy of a multiple eigenvalue (since it is not a block Lanczos method), but a single eigenvector corresponding to the maximum eigenvalue is all that is needed for convergence of the bundle method.

5.3 Active Block Strategy

It is obvious that blocks of Z for which the Lanczos process has converged (for instance, small blocks whose sizes are smaller than the size chosen for the Lanczos basis) may be excluded from further computation. By monitoring the Ritz values and their corresponding error bounds [28] in every remaining block prior to each implicit restart, we can ignore those blocks which are no longer candidates for producing one of the largest p eigenvalues; such blocks are deemed *inactive*. The remaining blocks are called *active*, and subsequent computation is restricted to these blocks only. This can produce significant savings (as demonstrated in §9) in problems with many blocks, but where the largest p eigenvalues are concentrated in a small number of blocks in most of the iterations.

[1] Not to be confused with the block Lanczos method.

5.4 Null Steps

We include a time-saving inexact evaluation procedure for null steps suggested by Helmberg [11]. Since the Ritz values provide progressively better lower bounds on the maximal eigenvalues as the Lanczos method progresses, the error bounds on the Ritz values may be used to determine if a null step is inevitable. In this case, further resolution of eigenpairs is prematurely terminated, and the algorithm performs a null step; see [11] for details.

5.5 Parallelism

Parallelism is easily exploited in Step 14 in the matrix-vector products $w = Zv^{j+1}$. Recall that $Z = \sum_{i=1}^{m} y_i A_i - C$ is not available explicitly, as the C, A_i $(i = 1, \ldots, m)$ matrices reside on different processors. If $\mathscr{A} \subseteq \{1, \ldots, s\}$ denotes the index set of active blocks at a particular iteration, then each processor q, including the root processor 0, uses the locally available A_i $(i \in M_q)$ to compute the partial sum

$$w_q(k) = \sum_{i \in M_q} y_i A_i(k) v^{j+1}(k) \qquad (k \in \mathscr{A}). \tag{13}$$

These partial sums are subsequently collected on the root processor which computes the desired matrix-vector product

$$w(k) = \sum_{q} w_q(k) - C(k) v^{j+1}(k) \qquad (k \in \mathscr{A}), \tag{14}$$

without explicitly forming the blocks $Z(k)$. Implicit restarts and complete reorthogonalization are executed in serial by the root processor. These operations depend on the number of Lanczos vectors stored (which is small in comparison to the block sizes $n_1, \ldots, n_s$) and the number of primal constraints m. For large-scale problems, these serial operations are dominated by the cost of the matrix-vector products.

We now combine all these features to give a full description of the eigenvector computation procedure in Algorithm 5.1. Steps 4 – 20 implement the basic Lanczos process, while Steps 28 – 40 deal with the implicit restarts, for which LAMBDA uses some ARPACK [20] routines. The portion in Steps 21 – 27 keeps track of the desired eigenvalues, computes the implicit shifts for the unwanted ones, and implements the active block strategy. More precisely, the error bounds for each eigenvalue produced by the Lanczos processes are used to infer which eigenvalues could never possibly be one of the p largest ones (not counting multiplicities). These unwanted eigenvalues then determine the implicit shifts and the inactive blocks.

As a general-purpose SDP solver, LAMBDA assumes no particular prior information about the maximal eigenvector, and hence initializes v^1 to be a random vector normalized within each block.

Algorithm 5.1 Block Structured Lanczos Method with Active Block Strategy

(Input) A vector y, a blockwise normalized initial vector v^1, a convergence tolerance `tol`, the number p of maximal eigenpairs desired, the number d of additional Lanczos vectors allowed.
(Output) p triples (θ, x, k) such that θ is an (approximate) maximal eigenvalue of Z, with (θ, x) being an (approximate) eigenpair of $Z(k)$.
Initialize the active blocks index set $\mathscr{A} = \{1, \ldots, s\}$, the inactive blocks index set $\mathscr{I} = \emptyset$, and the converged blocks index set $\mathscr{C} = \emptyset$.
Compute $w = Zv^1 = (A^T y - C)v^1$ in parallel using (13) and (14).
$\alpha^1 = (v^1)^T w; \quad r^1 = w - \alpha^1 v^1; \quad V^1 = v^1; \quad T^1 = \alpha^1; \quad \beta^1 = \|r^1\|.$
$j = 1$.
for rst $= 1, 2, \ldots$ until convergence **do**
  {—- begin restart loop —-}
  while $j < p + d$ **do**
    {—- begin Lanczos factorization —-}
    for $k \in \mathscr{A}$ **do**
      {—- augment Lanczos factorization for each block —-}
      $v^{j+1}(k) = r^j(k)/\beta^j(k); \quad V^{j+1}(k) = [V^j(k) \quad v^{j+1}(k)]; \quad \widehat{T}^j(k) = \begin{bmatrix} T^j(k); \ \beta^j(k)(e_j(k))^T \end{bmatrix}.$
      Compute $w(k) = Z(k)v^{j+1}(k)$ in parallel using (13) and (14).
      $t(k) = V^{j+1}(k)^T w(k); \quad r^{j+1}(k) = w(k) - V^{j+1}(k)t(k); \quad \beta^{j+1}(k) = \|r^{j+1}(k)\|.$
      $T^{j+1}(k) = [\widehat{T}^j(k) \ \ t(k)]$.
    end for {—- k loop —-}
    $j = j + 1$.
  end while
  Compute the spectral factorization $T^{p+d}(k) = S(k)\Theta(k)S(k)^T$ for $k \in \mathscr{A}$.
  For each $(i,k) \in \{1, \ldots, p+d\} \times \{1, \ldots, s\}$, test the convergence condition $\beta^{p+d}(k)\left|s_{p+d,i}(k)\right| < \mathtt{tol}$ to find indices (i,k) corresponding to converged eigenpairs.
  If all eigenpairs in any block $k \in \mathscr{A}$ have converged, set $\mathscr{C} = \mathscr{C} \cup \{k\}$.
  Compute upper bounds

$$\bar{\lambda}(k) = \max_{i=1\ldots,p+d;\, k \in \mathscr{A} \cup \mathscr{C}} \theta_i(k) + \beta^{p+d}(k)\left|s_{p+d,i}(k)\right|$$

  Set $\mathscr{W}$ to be the index pairs (i,k) for the p largest values among the upper bounds $\theta_i(k) + \beta^{p+d}(k)\left|s_{p+d,i}(k)\right|$, $(i = 1, \ldots, p+d \ ; k \in \mathscr{A} \cup \mathscr{C})$.
  Define ω to be the p^{th} largest of the $\theta_i(k) - \beta^{p+d}(k)\left|s_{p+d,i}\right|$ $(i = 1, \ldots, p+d;\ k \in \mathscr{A})$. Set $\mathscr{I} = \mathscr{I} \cup \left\{k \in \mathscr{A} \ : \ \bar{\lambda}(k) < \omega\right\}$.
  Update $\mathscr{A} = \mathscr{A} \setminus \{\mathscr{C} \cup \mathscr{I}\}$. {—- active block strategy —-}
  Compute lower bound on maximal eigenvalue

$$\underline{\lambda} = \max_{i=1,\ldots,p+d;\, k \in \mathscr{A}} \theta_i(k) - \beta^{p+d}(k)\left|s_{p+d,i}(k)\right|.$$

  If $\underline{\lambda}$ is large enough to violate (6) by the margin required for a null step, break out of the rst loop.
  If the Ritz values $\theta_i(k)$ for all $(i,k) \in \mathscr{W}$ have converged to the prescribed tolerance `tol`, break out of the rst loop.
  {—- synchronize Lanczos processes via implicit restarts —-}
  For each $k \in \mathscr{A}$, set $\widehat{Q}(k) = I_{p+d}$, and select the indices of the 'unwanted' eigenvalues

$$\mathscr{U} = \left\{(i,k) \ : \ \theta_i(k) + \beta^{p+d}(k)\left|s_{p+d,i}\right| < \omega\right\}. \tag{15}$$

  for $(i,k) \in \mathscr{U}$ **do**
    $[Q(k), R] = \mathrm{qr}(T^{p+d}(k) - \theta_i(k) I_{p+d})$. {—- QR factorization with implicit shifts —-}
    $T^{p+d}(k) = Q(k)^T T^{p+d}(k) Q(k)$.
    $\widehat{Q}(k) = \widehat{Q}(k)\, Q(k)$.
  end for
  for $k \in \mathscr{A}$ **do**
    $\beta_+^p(k) = T_{p+1,p}^{p+d}(k); \quad \sigma^p = \widehat{Q}_{p+d,p}(k)$.
    $r^p(k) = \beta_+^p(k) v^{p+1}(k) + \sigma^p r^p(k)$.
    $V^p(k) = V^{p+d}\widehat{Q}_{:,1:p}(k); \quad T^p(k) = T_{1:p,1:p}^{p+d}(k)$.
    Reorthogonalize $V^p(k)$.
  end for {—- k loop —-}
  $j = p$.
end for {—- rst loop —-}
For each k such that $(i,k) \in \mathscr{W}$, reorthogonalize $V^{p+d}(k)$.
For each of the p index pairs $(i,k) \in \mathscr{W}$, compute an approximate eigenvector $x_i(k) = V^{p+d}(k)s_i(k)$. Return the p triples $(\theta_i, x_i(k), k)$ for each $(i,k) \in \mathscr{W}$.

6 Subdifferential Model

LAMBDA employs a mixed polyhedral-semidefinite ε-subdifferential model. At a given iteration, the bundle consists of a polyhedral part $G = [g_1, \ldots, g_l] \in \mathbb{R}^{m \times l}$ and a semidefinite part $P \in \mathbb{R}^{n \times r}$, resulting in a subdifferential model of the form

$$\left\{ \sum_{i=1}^{l} \alpha_i g_i + A(PVP^T) - b \ : \ \alpha_i \geq 0,\ V \succeq 0,\ \alpha_1 + \ldots + \alpha_l + \mathrm{tr}(V) = 1 \right\},$$

with some columns in G and in P possibly retained from earlier iterations, *i.e.* each $g_i = A(W_i)$ for some subgradient $W_i \in \partial_\varepsilon \lambda_{\max}(Z)$, each column p_i of P is a maximal eigenvector of Z, and $Z = A^T y - C$ is derived from the current iterate $y = y^k$ or an earlier iterate $y = y^j$ $(j \in J^k)$.

To enrich the model (in the Update step in Algorithm 2.1), maximal eigenvectors to $Z = A^T \bar{y} - C$ are orthogonalized against, and subsequently added to, existing columns in P. This model may be viewed as a hybrid combination of that used by traditional bundle methods (obtained by restricting V to be diagonal) and the one used by the spectral bundle method (obtained by restricting $l = 1$, a minimal requirement for aggregation).

Following [25], we can write the model minimization subproblem (5) as

$$\begin{aligned} \min_{(\alpha;V) \in \mathbb{R}^l \times \mathbb{S}^r} \quad & \beta \\ \text{s.t.} \quad & \frac{1}{2} \left\langle [\alpha; \mathrm{svec}(V)], Q\,[\alpha; \mathrm{svec}(V)] \right\rangle + \left\langle [u;\ v], [\alpha; \mathrm{svec}(V)] \right\rangle - \beta \leq 0 \qquad (16) \\ & \sum_{i=1}^{l} \alpha_i + \mathrm{tr}(V) = 1 \\ & \alpha \geq 0;\ V \succeq 0, \end{aligned}$$

where the matrix Q and the vector $[u;\ v]$ are given by

$$u_i = \left\langle y - \frac{1}{\rho} b, A(W_i) \right\rangle - \langle C, W_i \rangle \quad (i = 1, \ldots, l), \qquad (17)$$

$$v = \mathrm{svec}(P^T (A^T (y - \frac{1}{\rho} b) - C) P)], \ \text{and} \qquad (18)$$

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix}, \ \text{with} \qquad (19)$$

$$Q_{11} = \frac{1}{\rho} G^T G, \qquad (20)$$

$$Q_{12} = \frac{1}{\rho} \begin{bmatrix} (\mathrm{svec}(P^T (A^T g_1) P))^T \\ \vdots \\ (\mathrm{svec}(P^T (A^T g_l) P))^T \end{bmatrix}, \ \text{and} \qquad (21)$$

$$Q_{22} = \frac{1}{\rho} \sum_{i=1}^{m} \mathrm{svec}(P^T A_i P) \left(\mathrm{svec}(P^T A_i P)\right)^T. \qquad (22)$$

It is obvious from our data distribution scheme and the formulas above that the evaluations of $A(\cdot)$ and $A^T(\cdot)$ in (17), (18) and (21), as well as the orthogonal conjugations in (21) and (22) may be performed in parallel. Further, observe that the polyhedral part of the bundle can be distributed in exactly the same manner as the C, A_i $(i = 1, \ldots, m)$, *i.e.* processor k holds those rows of G whose indices $i \in M_k$. Hence the Gram matrix in (20) may also be computed in parallel. Note that each processor needs the matrix P to compute the orthogonal conjugations defining Q_{22}, hence the semidefinite portion of the bundle must be stored on all processors, if the costs of communicating it are to be avoided.

7 Subproblem

The data defining the subproblem (Q, u, v) thus computed in parallel are assembled on the root processor, which solves the subproblem — a conic program — in serial using SeQuL, a primal-dual interior-point code written in the C language by J.-P. Haeberly. Conversion of (16) to standard, dual form requires a Cholesky factorization of Q. Pivoting is employed to deal with (nearly) linearly dependent subgradients that cause rank deficiency in Q. Finally, since SeQuL accepts only inequalities in its dual formulation, we incorporated modifications to the algorithm to accommodate the equality constraint in (16).

Whereas in a serial code, solving the subproblem takes a negligible fraction of the time spent in any iteration, this may not be the case in a parallel context. Dominant computational costs (subgradient computation) could be sped up so much with multiple processors (as we will demonstrate in §9) that, at least for some range of problem sizes and number of available processors, the time spent in solving the subproblem and other serial portions of the code is no longer negligible, and will eventually limit scalability. The mixed polyhedral-semidefinite subdifferential model helps in this regard by allowing a small semidefinite model to be compensated, to some extent, by a large polyhedral model, thus allowing the subproblem to be solved quickly.

8 Update

To keep storage bounded, subgradients in both the polyhedral and semidefinite parts of the bundle that are less relevant to the model need to be lumped together to make room for the addition of new subgradients in subsequent iterations. This update of the bundle is termed *aggregation* [17], and is implemented as described in [25].

The final ingredient is a suitable update of ρ^k, the penalty parameter. Roughly speaking, one would expect to increase ρ^k after a null step and decrease it after a serious step, but in any case, ensuring that it never violates the lower bound $\rho_{\min}$ required for convergence. Based on Helmberg's numerical experiments [12], LAMBDA uses a starting value of $\rho^0 = \left\|g^0\right\| / \sqrt{m}$, but updates it using a rather involved scheme developed by Kiwiel [18].

The updates to the bundle and to the penalty parameter are executed in serial by the root processor.

9 Numerical Results

Our description of the experimental setup (collection of test problems, hardware details, default parameters) is followed by numerical experiments evaluating convergence behavior, the proposed block structured Lanczos method for eigenvector computation, the effect of the polyhedral component in the subdifferential model, and finally, parallel scalability.

9.1 Experimental Setup

9.1.1 Test Problems

Our overall test set consists of 13 problems (see Table 1), including:

- a fully dense, randomly generated SDP,
- four SDP relaxations (max-cut and Lovász ϑ-function) of graph problems[2], and
- eight SDP relaxations arising in quantum chemistry from *ab initio* calculations of electronic structure.[3]

The maxG11 problem is a max-cut relaxation on a graph with 800 nodes. For the theta problems, the names indicate the sizes of the graphs, *e.g.* theta-5k-67k is for a graph with 5,000 nodes and about 67,000 edges. The quantum chemistry problems have 23 blocks each. Using $a(b)$ to denote b blocks each of size $a \times a$, the block structure for these problems is [6(1) 10(4) 45(4) 100(4) 120(2) 200(1) 230(1) 450(4) 1450(2)] with $m = 7230$ constraints, except in $BH^+$where the block structure is [6(5) 15(4) 20(2) 36(4) 72(1) 90(4) 94(1) 306(2)] with $m = 948$ constraints. The total number of nonzeros in the C, A_i $(i = 1, \ldots, m)$ for the quantum chemistry problems is about 0.85 million, except for BH^+, where it is 0.067 million. For the dense, randomly generated problem rand-1k-8k, the total number of nonzeros is about 1.17 millon.

The problem $BH^+$is small enough to be solved quickly by interior-point methods even in serial. (Indeed SDPA solves it to high accuracy within about 30 minutes, when executed in serial on one of the nodes of the KALI cluster described below.) Nevertheless, its solution provides insight into the convergence behavior of the bundle method. In particular, the quantum chemistry problems, though modest in size, have solutions where the maximal eigenvalue has high multiplicity — a feature that poses a significant challenge for bundle methods.

9.1.2 Hardware

All runs of our code were conducted on KALI,[4] a 64-cpu Beowulf cluster consisting of 32 dual Intel Xeon nodes operating at 2 GHz. Each processor has a 512 KB L2 cache. Each node has 1 GB RAM (shared by two processors), with the entire

[2] Graphs generated using the graph generator program `rudy`, written by G. Rinaldi.

[3] Available from `http://www.is.titech.ac.jp/~mituhiro/software.html` in the SDPA format.

[4] See `http://kali.math.umbc.edu` for details.

Table 1 Description of problems in test set.

Prob	$n_1,...,n_s$	m	Description
rand-1k-8k	[250 250 250 250]	8192	Fully dense, random
maxG11	800	800	Max cut
theta-1k-4k	1,001	5996	Lovász ϑ-function, random graph
theta-5k-67k	5001	67489	Lovász ϑ-function, random graph
theta-5k-100k	5001	105,106	Lovász ϑ-function, random graph
BH^+	(see §9.1.1)	948	Quantum chemistry
B_2		7230	Quantum chemistry
BeO		7230	Quantum chemistry
C_2		7230	Quantum chemistry
C_2^+		7230	Quantum chemistry
Li_2		7230	Quantum chemistry
LiF		7230	Quantum chemistry
NaH		7230	Quantum chemistry

cluster interconnected by a Myrinet network (2 Gbit/sec peak bandwidth). One of these 32 nodes also acts as a *storage node* with access to a 0.5 TB RAID array.

9.1.3 Default Parameter Values

The default tolerance for eigenvalue computations is 10^{-12}. The parameter m_L for determining a serious step is set to 0.1. The parameter $\rho_{\min} = 10^{-6}$, and Kiwiel's update [17] of ρ requires another parameter $m_R \in (m_L, 1)$, which we set to 0.5. The maximum number of iterations allowed is $10,000$. The default accuracy level is $\delta = 0.01$. In every iteration, $p = 5$ new eigenvectors are computed using $p + d = 50$ Lanczos vectors. The subdifferential model contains $l = 10$ subgradients in the polyhedral part, and $r = 15$ eigenvectors in the semidefinite part. The starting point is always $y^0 = 0$. Exceptions to these default values are mentioned when appropriate.

In all tables, the following naming conventions are employed: 'Prob' denotes problem name, 'Ser' and 'Tot' denote the number of serious steps and the total number of iterations, 'err' is the relative error in the objective value, and 'Time' is computational time. Timings are reported entirely in seconds or as hh:mm:ss (hours:minutes:seconds). All timings are elapsed wall-clock time, and exclude the time taken to read problem data.

9.2 Convergence

In Figure 1, we plot the objective function values for four of the smaller problems, with each plotted point denoting a serious step. The circles on the plot indicate the number of iterations needed for stopping tolerances of $\delta = 0.1, 0.01, 0.001$. For all problems, the plots exhibit the characteristic tailing off effect of first order bundle methods, *i.e.* rapid initial progress, with the $\delta = 0.1$, and $\delta = 0.01$ accuracy levels attained within a few iterations, and the $\delta = 0.001$ accuracy level attained between 100 and 1000 iterations.

A notable exception is BH^+ which takes an order of magnitude more iterations to attain the $\delta = 0.001$ accuracy level. This problem is block structured with 22

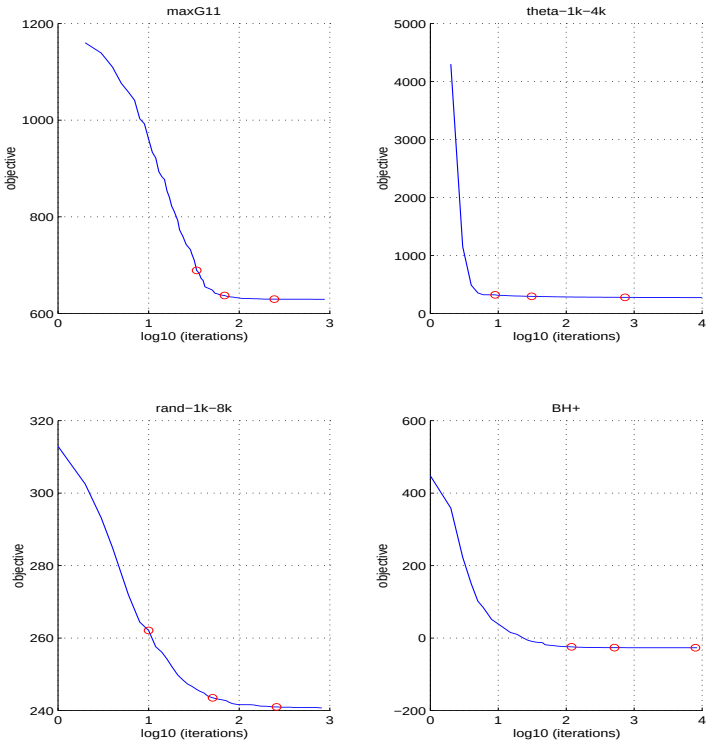


Fig. 1 Typical convergence behavior (serious steps only shown in plot) on four problems.

blocks making up a matrix of total dimension 1406. Yet the multiplicity of the maximal eigenvalue of $Z^* = A^T y^* - C$ at any optimal solution y^* is at least 368, as confirmed by a solution of high accuracy computed by SDPA [7]. The slow convergence witnessed here is an inevitable outcome of the fact that the dimension of our subdifferential model ($l = 10$ vectors in the polyhedral part G, and $r = 15$ eigenvectors in the semidefinite part P) falls severely short of the dimension of the subdifferential at the optimal solution. Using a computed solution $\tilde{y}$ and the true optimal value f^*, we calculate the relative error as

$$\text{err} = \frac{|f(\tilde{y}) - f^*|}{1 + |f^*|}.$$

For the three problems other than BH^+, this quantity is less than 1% for $\delta = 0.01$, and less than 0.1% for $\delta = 0.001$. For BH^+, this error is 1.69% with $\delta = 0.01$. With $\delta = 0.001$, this relative error decreases to 0.7% (at the cost of about 7900 extra iterations), but a solution satisfying the $\delta = 0.0001$ accuracy level was unattainable within 10,000 iterations. Relative errors for the other instances in this

Table 2 Relative errors in the optimal value for the quantum chemistry problems with $\delta = 0.01$.

Problem	err
BH^+	0.0169
B_2	0.0179
BeO	0.0183
C_2	0.0187
C_2^+	0.0195
Li_2	0.0128
LiF	0.0153
NaH	0.0255

problem class are similar and are given in Table 2; for f^*, we used the highly accurate optimal values computed in [34].

9.3 Active Block Strategy

In Table 3, we show the effect of the active block strategy (described in§5.3) on the time required for eigenvector calculation and overall run time. Each problem has two rows: (i) overall solution time, and (ii) time for eigenvalue / eigenvector computation. The active block strategy produces savings on every problem, and sometimes by significant amounts. This is remarkable for the quantum chemistry problems, since the maximal eigenvalue at the optimal solution not only has high multiplicity but also occurs in most of the blocks. For example, this maximal eigenvalue for BH^+, earlier noted to have multiplicity at least 368, occurs in 18 of the 22 blocks. Consequently, close to the optimal solution, only 4 blocks may be eliminated prematurely from the matrix-vector products in the Lanczos method. Yet the savings on computation and communication realized in the early iterations of the algorithm (when the maximal eigenvalue is simple or occurs only in a few blocks) are enough to speed up eigenvector computation and total solution time by more than 3.5 times. Savings on other problems range from barely noticeable (Li_2, NaH), through significant (C_2^+, `rand-1k-8k`), to dramatic (B_2, BeO, C_2, LiF).

9.4 Subdifferential Model

Next, we show the effect of the mixed polyhedral-semidefinite subdifferential model on parallel run time using two sets of experiments on BH^+.

In the first set (Table 4), we use a coarse accuracy level of $\delta = 0.01$. The columns titled 'Total' and 'Sub' denote total solution time and time occupied by the subproblem (in seconds); the last column expresses the subproblem solution time as a percentage of total solution time. The first row is a serial run taking 310 seconds, of which subproblem solution occupies under 12%. In a parallel context, solving the subproblem could become a serial bottleneck. Table 4 shows a series of runs, all conducted on 8 processors, but with varying sizes of polyhedral and semidefinite components in the subdifferential model. All runs require more or less the same number of serious steps to achieve the specified accuracy, but the richer models (larger r, smaller l) do so in fewer total iterations, as expected. Nevertheless, the simpler models (smaller r, larger l) solve the problem faster.

Table 3 Effect of the active block strategy on run time. All problems were solved with $\delta = 0.01$, $l = 15$ and $r = 10$ on 4 processors.

Name	Time (without ABS)	Time (with ABS)
BH^+	00:25:09 00:25:00	00:07:40 00:07:00
B_2	06:14:06 06:06:40	01:33:26 01:28:28
BeO	13:25:22 13:20:01	03:20:50 03:20:08
C_2	06:54:15 06:51:23	01:41:43 01:36:40
C_2^+	01:48:39 01:43:20	01:31:39 01:28:21
Li_2	02:31:03 02:26:00	02:28:28 02:23:20
LiF	01:25:13 01:21:40	00:50:03 00:48:21
NaH	01:28:21 01:25:01	01:23:44 01:20:01
rand-1k-8k	00:20:36 00:19:51	00:15:36 00:14:42

Table 4 Parallel runs on BH^+ for $\delta = 0.01$ showing the effect of the subdifferential model on overall solution time. The first row alone is a serial run; the remaining ones used 8 processors.

Ser / Tot	l	r	Total	Sub	%
24 / 282	10	25	310	37	11.94
23 / 283	10	25	130	38	29.23
25 / 335	15	20	95	22	23.16
25 / 351	20	15	84	11	13.10
22 / 315	25	10	72	6.3	8.75
24 / 402	30	5	65	2.8	4.31

In the second set, we repeat a similar sequence of runs, but using a finer accuracy level of $\delta = 0.001$. Since the optimal multiplicity is high, a large subdifferential model will be required, so we choose $l = 10$ and $r = 50$ for the serial run shown in the first row of Table 5. Here solving the subproblem already occupies a substantial 45% of the total solution time, since the blocks are small. Similar remarks as in the first set apply to iteration counts and solution times, but note that the last row (with $l = 50$ and $r = 10$) shows an increase in the solution time achieved by the penultimate row (with $l = 40$ and $r = 20$). Thus there is a tradeoff between saving computational time per iteration by using a simpler model and incurring an increase in iteration count, with a commensurate increase in overall solution time.

The speed-up factors of 4 – 5 times obtained by a suitable choice of the subdifferential model are due to the small size of this problem; the subproblem solution occupies a signicant fraction of overall solution time in the serial run because the blocks are small. With increasing problem size, the speed-up factors become less dramatic, and the choice of the subdifferential model becomes less crucial, as evidenced by Table 6.

Table 5 Parallel runs on BH^+ for $\delta = 0.001$ showing the effect of the subdifferential model on overall solution time. The first row alone is a serial run; the remaining ones used 8 processors.

Ser / Tot	l	r	Total	Sub	%
44 / 1156	10	50	3085	1400	45.38
44 / 1236	10	50	1900	1400	73.68
48 / 1554	20	40	1600	970	60.62
45 / 1688	30	30	920	370	40.22
42 / 1635	40	20	650	130	20.00
44 / 2449	50	10	820	58	7.07

Table 6 Parallel runs on LiF for $\delta = 0.01$ showing the effect of the subdifferential model on overall solution time. The first row alone is a serial run; the remaining ones used 16 processors.

Ser / Tot	l	r	Total	Sub	%
32 / 354	10	25	5685	49	0.86
33 / 351	10	25	550	45	8.18
31 / 367	15	20	474	22	4.64
32 / 402	20	15	609	12	1.97
30 / 433	25	10	676	5.4	0.80
33 / 386	30	5	457	2.2	0.48

9.5 Scalability

To study parallel scalability, we solved the 10 largest problems in our test set on processors numbering 1 through 64 in powers of two. The results are reported in Table 7, where the columns denote the number of processors used. Each row contains three lines in the following order: (i) total solution time; (ii) time used in calculating eigenvalues (objective function values) and eigenvectors (subgradients); and (iii) time spent in forming the data for the subproblem. Since eigenvector computation is the dominant expense, the scalability of overall solution time essentially follows that of eigenvector computation.

A curious phenomenon in these experiments is that the same problem solved on different numbers of processors could yield quite different iterate sequences, producing seemingly anomalous entries in Table 7. Rounding errors in the Lanczos process sometimes produce different eigenvectors (albeit from the same eigenspace), and hence different subgradients, and consequently a different solution to the subproblem, which may further trigger the conversion of a null step into a serious step or vice versa. In general, there could be significant differences in iteration sequences, number of serious and null steps, and running time across the columns of Table 7 for any given problem. In particular, the suprisingly fast execution time for NaH on 64 processors compared to that on 32 processors is mostly due to the fact that the 64-processor run performed 17% fewer serious steps with the benefit of the early termination heuristic. A null step is typically several times faster than a serious step, and this has resulted in a fortuitous speed-up. In contrast, the 64-processor runs of BeO and LiF consumed approximately the same number of total iterations and serious steps as their respective 32-processor runs, yet the solution times have nearly doubled. This increase in run time as the number of processors is increased — typical in parallel computing applications — is purely due to communication losses.

Table 7 Scalability studies on ten problems on up to 64 processors.

Name	1	2	4	8	16	32	64
B_2	05:50:01	03:03:23	01:33:26	01:01:47	00:43:22	00:55:51	00:33:49
	05:16:42	02:46:40	01:28:28	01:00:06	00:41:41	00:55:00	00:33:20
	00:01:06	00:00:51	00:00:59	00:00:28	00:00:05	00:00:04	00:00:03
BeO	09:26:39	04:26:48	03:20:50	02:21:49	01:28:22	01:28:25	03:03:29
	08:36:41	04:10:07	03:20:08	02:18:23	01:26:42	01:26:46	03:02:24
	00:01:50	00:01:12	00:01:39	00:00:56	00:00:08	00:00:05	00:00:04
C_2	04:43:21	02:45:30	01:41:43	00:55:00	00:43:47	00:35:58	00:45:42
	04:10:05	02:38:22	01:36:40	00:53:20	00:43:20	00:35:00	00:43:21
	00:00:55	00:00:42	00:00:49	00:00:29	00:00:04	00:00:03	00:00:02
C_2^+	05:16:48	02:26:44	01:31:39	00:58:18	00:41:56	00:48:29	00:56:47
	05:00:03	02:20:00	01:28:21	00:56:43	00:40:02	00:46:39	00:55:01
	00:00:58	00:00:52	00:00:47	00:00:35	00:00:05	00:00:04	00:00:04
Li_2	07:46:51	04:10:48	02:28:28	01:31:39	00:53:28	01:01:06	01:10:20
	07:30:11	03:53:23	02:23:20	01:30:02	00:51:41	01:00:01	01:08:31
	00:01:40	00:01:08	00:01:27	00:00:47	00:00:09	00:00:07	00:00:06
LiF	02:36:42	01:01:40	00:50:03	00:35:52	00:23:25	00:20:52	00:39:20
	02:23:17	00:58:16	00:48:21	00:33:20	00:21:45	00:18:27	00:38:20
	00:00:31	00:00:29	00:00:31	00:00:17	00:00:03	00:00:02	00:00:02
NaH	03:36:47	02:01:39	01:23:44	00:39:34	00:41:01	00:28:20	00:08:20
	03:20:09	01:55:04	01:20:01	00:38:18	00:40:00	00:28:20	00:07:40
	00:00:39	00:00:42	00:00:44	00:00:25	00:00:04	00:00:02	00:00:02
rand-1k-8k	00:48:21	00:27:42	00:15:36	00:08:38	00:06:29	00:04:30	00:03:32
	00:43:20	00:26:38	00:14:42	00:08:20	00:06:10	00:04:18	00:03:20
	00:00:37	00:00:21	00:00:12	00:00:06	00:00:03	00:00:02	00:00:01
theta-5k-67k	01:50:11	00:50:10	00:23:28	00:09:42	00:08:06	00:04:48	00:03:01
	01:45:03	00:46:41	00:18:22	00:08:13	00:06:10	00:04:10	00:02:43
	00:00:30	00:00:14	00:01:09	00:00:32	00:00:21	00:00:11	00:00:06
theta-5k-100k	07:13:28	03:53:19	02:01:08	00:58:21	00:30:02	00:15:50	00:09:24
	06:56:42	03:36:40	01:56:40	00:56:26	00:26:40	00:13:41	00:08:20
	00:08:01	00:04:13	00:02:23	00:01:31	00:01:11	00:00:55	00:00:17

The speed-up plots in Figure 2 for overall solution time provide a quick picture of scalability. In these plots, the speed-up factor $S(p)$ for p processors is calculated as

$$S(p) = \frac{\text{Time taken on 1 processor}}{\text{Time taken on } p \text{ processors}}.$$

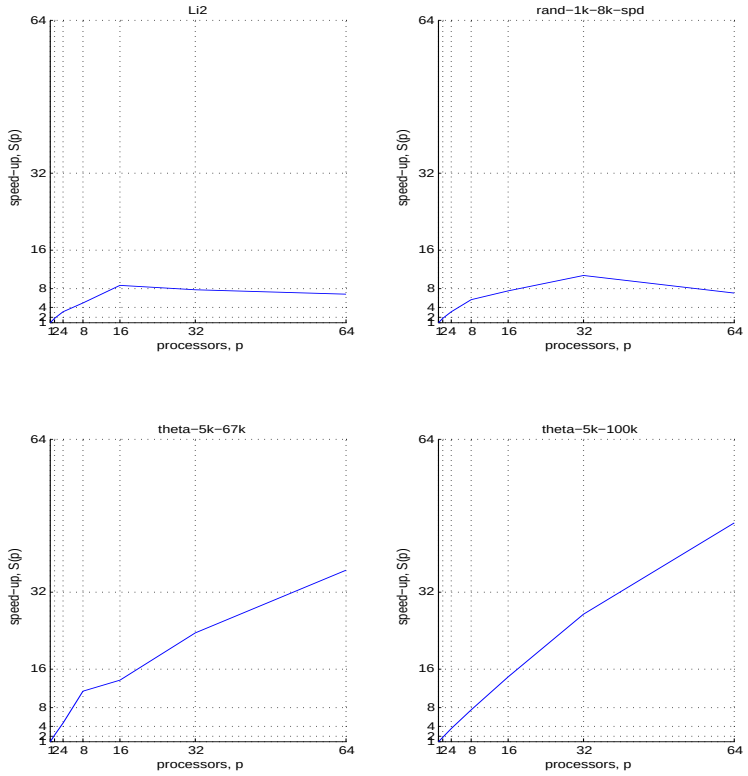
In summary, for the quantum chemistry problems, the speed-up is acceptable for up to 8 processors. (We include only one speed-up plot from this class of problems, as the plots for the remaining problems are more or less similar.) Improved solution times are obtained with up to 16 processors (and on some problems, up to 32 processors), beyond which communication costs inevitably degrade run time. But we hasten to point out two facts: (i) these problems have relatively small blocks (with only two moderately large blocks of size 1450 each) and a modest value of $m = 7230$, and hence are too small to be fully scalable up to the 64 processors available in our cluster; and (ii) although these problems were the largest instances available within their class at the time of this work, they constitute toy problems in the realm of quantum chemistry. Realistic atomic systems would yield much larger problems more amenable to scalability up to 64 processors.

The situation improves slightly with `rand-1k-8k`. Although comparable in size with the quantum chemistry problems, this problem is fully dense, hence

there is a higher computation-to-communication ratio in the matrix-vector products within the Lanczos method. Acceptable scalability is observed for up to 8 processors, with solution times improving up to 32 processors.

With increasing problem size, as in the two Lovász ϑ-function SDP relaxations, the scalability improves noticeably, with solution times improving consistently all the way up to 64 processors.

Fig. 2 Speed-up plots for scalability studies.



9.6 Fastest Solution Times

Finally, we report the fastest run times that could be obtained in solving the large quantum chemistry instances to low accuracy with $\delta = 0.01$; this corresponds to relative errors given in Table 2. To obtain the fastest solution times, we compute only $p = 2$ eigenvectors in each iteration using $p + d = 20$ Lanczos vectors. In Table 8, we show

Table 8 Fastest solution times on the large quantum chemistry problems using $\delta = 0.01, l = 10, r = 20, p = 2, q = 20$ on 16 processors. The *rst* and *ip.it* columns are in thousands, and the *ops* column is in millions.

Name	It		Eig			Sub			Time
	ser	*tot*	*rst*	*ops*	*time*	#	*ip.it*	*time*	
B_2	35	584	20.6	1.16	00:14:10	596	20.3	00:00:47	00:16:12
BeO	32	580	18.7	0.97	00:12:00	609	21.9	00:00:51	00:13:55
C_2	34	547	18.2	0.94	00:12:30	564	19.1	00:00:41	00:14:16
C_2^+	35	1655	53.6	2.33	00:30:02	1701	59.1	00:02:00	00:36:33
Li_2	30	761	33.1	1.60	00:32:03	791	26.7	00:00:50	00:34:21
LiF	31	367	12.0	0.55	00:06:50	393	10.2	00:00:22	00:08:01
NaH	35	453	13.2	0.55	00:10:08	481	14.9	00:00:32	00:11:54

- the number of iterations (serious steps, *ser*; total iterations, *tot*),
- details pertaining to eigenvalue computation (number of implicit restarts, *rst*; number of matrix-vector products counting each block separately, *ops*; time spent in eigenvalue computation, *time*),
- details pertaining to the subproblem (number of subproblems solved, #; the total number of interior-point iterations, *ip.it*; time spent in solving subproblems, *time*), and
- in the last column, the total solution time for each problem instance.

As a rough comparison with interior-point methods, we mention that these problems (in addition to others) were originally solved in [34] to high accuracy (6 or 7 digits) using the parallel interior-point code SDPARA [33] on the NERSC SMP (shared memory multiprocessor) cluster SEABORG.[5] Each problem required about 14 hours of solution time and 27 GB of memory on a 16-cpu node with 375 MHz processors (slower than KALI's 2 GHz processors), but 8 MB of L2 cache and 64 GB of shared memory (much larger than KALI's 512 KB L2 cache per processor and 1 GB of memory per dual node). Shared memory eliminates expensive communication over a network. A recent update [8] to these results, using an improved version of SDPARA called SDPARA-SMP, has reduced solution time to about 13 hours and memory usage to about 5.7 GB. With this improved code, the authors were able to solve slightly larger atomic systems (but significantly larger SDP's with $m \approx 20,000$ and a largest block of size around 3200) in about 5.3 days using 73.9 GB of memory on a 64-cpu subcluster of SEABORG. Recently, Mazziotti [21, 22] has reported results on even larger atomic systems (although using a weaker SDP relaxation than employed in [8]) based on a different type of first order method for semidefinite programming.

10 Perspectives

Subgradient methods, though used for eigenvalue optimization over 30 years ago by Cullum, Donath and Wolfe [5], had not seen widespread use in semidefinite programming until revived by the spectral bundle method of Helmberg and Rendl [14] and improved variants [13] thereof. Our implementation draws much

[5] See `http://www.nersc.gov/nusers/resources/SP` for the hardware details of this cluster.

from Helmberg's serial code `SBmethod` [10], which has been remarkably successful in solving large-scale problems, albeit restricted to those arising in graph applications.

Our efforts to extend the applicability of the bundle methodology center around efficiently handling block diagonal structure, while resorting to parallelism to solve very large-scale problems. The proposed data distribution scheme allows efficient storage of problem data and subgradients together with ease of implementation, without sacrificing performance on algorithmic components (block structured Lanczos, the active block strategy, implicit restarting, and a choice of subdifferential models). However, this scheme is not without limitations. In problems where sparsity levels are vastly different among the data matrices C, A_i $(i = 1,\ldots,m)$, this scheme results in load imbalance. Thus some important problem classes (notably maximum cut and graph bisection) cannot be effectively parallelized. No single scheme can work equally well for all types of problems, and these problem classes are best handled in a problem-dependent way. A second limitation in the present implementation is its inability to effectively handle linear inequalities and bound constraints on the variables. Such constraints serve to tighten SDP relaxations of combinatorial optimization problems, and `SBmethod` handles them by incorporating a second (inner) iterative procedure within the proximal bundle method. Thus LAMBDA is presently not fully tailored for applications in combinatorial optimization. Finally, `SBmethod` uses Lanczos vectors, even those that have not converged to eigenvectors, to construct cutting planes. We believe this is not essential in LAMBDA, since it already has the option of retaining many additional subgradients in the polyhedral part of the bundle.

The Block Structured Lanczos Method offers an advantage[6] even for problems without block structure, when combined with a preprocessing technique introduced in [9, 24]. This technique, originally proposed to exploit sparsity, uses positive semidefinite matrix completion to convert a semidefinite program with a single large block into one with several smaller blocks. Thus subgradient computation with the Block Structured Lanczos Method is likely to be more efficient on the modified problem with multiple blocks than on the original problem with a single large block.

We now comment on performance. Throughout, we have used basic blocking communication, *i.e.* a processor executing a 'Send' ('Receive') will wait until the receiving (sending) processor executes a matching 'Receive' ('Send'). Using more sophisticated MPI communication modes, it is possible to overlap computation with communication in some parts of the algorithm. Further performance improvement could come from selective orthogonalization and by using a block Lanczos method. The block Lanczos method is better at resolving multiple eigenvalues. Also, a single communication of a block of vectors incurs a smaller network latency cost than multiple communications each involving a single vector. However, the biggest gains are to be realized by fully exploiting problem structure, which is unfortunately completely lost by encoding problems in the SDPA data format. Thus, in our experiments, the data matrices C, A_i $(i = 1,\ldots,m)$ from all the quantum chemistry problems were treated as unstructured, sparse matrices. On the whole, given that the Lanczos method is an intrinsically serial process (one cannot compute Z^2v before computing Zv), and that the parallelization is

[6] We thank an anonymous referee for this observation.

fine-grained at the linear algebra level, the observed speed-up factors and solution times are better than anticipated.

Finally, we fully acknowledge that the low accuracy solutions computed for the quantum chemistry problems do not meet the 6 or 7 digits of accuracy required for ground state energy calculations. We neither claim to have 'solved' these problems, nor do we suggest that first order bundle methods are a viable solution methodology for them. We have merely used them as realistic, challenging instances which allow all aspects of this general-purpose code to be fully tested, and as such, the numerical results for this problem set are to be viewed in that light. Attaining high accuracy by incorporating some limited form of second order information, ideally in an parallelizable way, remains a topic worthy of future investigation.

Acknowledgments

We are grateful to Jean-Pierre Haeberly for allowing use of the SeQuL code; to Mituhiro Fukuda for helpful conversations about the quantum chemistry problems; and especially to Jorge Moré for access to the Chiba City cluster at Argonne National Lab during the early stages of this work. We also thank the anonymous referees for their constructive comments.

This work was done while the author was in the Department of Mathematics & Statistics at the University of Maryland, Baltimore County, and was supported in part by NSF grants DMS-0238008 and DMS-0215373. The computational results presented here would not have been possible without the high-performance cluster KALI, which was funded in part by the latter grant.

References

1. Benson, S.J., Ye, Y., Zhang, X.: Solving large-scale sparse semidefinite programs for combinatorial optimization. SIAM Journal on Optimization **10**(2), 443–461 (2000)
2. Burer, S.: Semidefinite programming in the space of partial positive semidefinite matrices. SIAM Journal on Optimization **14**(1), 139–172 (2003)
3. Burer, S., Monteiro, R.D.C.: A nonlinear programming algorithm for solving semidefinite programming via low-rank factorization. Mathematical Programming (Series B) **95**, 329–357 (2003)
4. Burer, S., Monteiro, R.D.C.: Local minima and convergence in low-rank semidefinite programming. Mathematical Programming (to appear)
5. Cullum, J., Donath, W., Wolfe, P.: The minimization of certain nondifferentiable sums of eigenvalues of symmetric matrices. Mathematical Programming Study **3**, 35–65 (1975)
6. Fletcher, R.: Semidefinite matrix constraints in optimization. SIAM Journal on Control and Optimization **23**, 493–523 (1985)
7. Fujisawa, K., Kojima, M., Nakata, K., Yamashita, M.: SDPA User's Manual — Version 6.00. Department of Mathematical and Computing Sciences, Tokyo Institute of Technology (2002)
8. Fukuda, M., Braams, B.J., Nakata, M., Overton, M.L., Percus, J.K., Yamashita, M., Zhao, Z.: Large-scale semidefinite programs in electronic structure calculations. Tech. Rep. B-413, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology (2005)
9. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting sparsity in semidefinite programming via matrix completion I: General framework. SIAM Journal on Optimization **11**(3), 647–674 (2000)

10. Helmberg, C.: SBmethod: A C++ implementation of the spectral bundle method. Tech. Rep. ZR-00-35, Konrad-Zuse-Zentrum für Informationstechnik, Berlin (2000)
11. Helmberg, C.: Semidefinite programming for combinatorial optimization. Tech. Rep. ZR-00-34, TU Berlin, Konrad-Zuse-Zentrum, Berlin (2000)
12. Helmberg, C.: Numerical evaluation of SBmethod. Mathematical Programming **95**(2), 381–406 (2003)
13. Helmberg, C., Kiwiel, K.C.: A spectral bundle method with bounds. Mathematical Programming **93**(2), 173–194 (2002)
14. Helmberg, C., Rendl, F.: A spectral bundle method for semidefinite programming. SIAM Journal on Optimization **10**(3), 673–696 (1999)
15. Hiriart–Urruty, J–B., Lemaréchal, C.: Convex analysis and minimization algorithms, vol. I & II. Springer–Verlag (1993)
16. Hiriart–Urruty, J–B., Ye, D.: Sensitivity analysis of all eigenvalues of a symmetric matrix. Numerische Mathematik **70**, 45–72 (1995)
17. Kiwiel, K.C.: An aggregate subgradient method for nonsmooth convex minimization. Mathematical Programming (1983)
18. Kiwiel, K.C.: Proximity control in bundle methods for convex nondifferentiable minimization. Mathematical Programming **46**, 105–122 (1990)
19. Kočvara, M., Stingl, M.: On the solution of large-scale SDP problems by the modified barrier method using iterative solvers. Tech. Rep. 304, Institute of Applied Mathematics, University of Erlangen (2005)
20. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK Users' Guide. SIAM, Philadelphia (1998)
21. Mazziotti, D.A.: First-order semidefinite programming for the direct determination of two-electron reduced density matrices with application to many-electron atoms and molecules. Journal of Chemical Physics **121**, 10,957–10,966 (2004)
22. Mazziotti, D.A.: Realization of quantum chemistry without wavefunctions through first-order semidefinite programming. Physical Review Letters **93**, 213,001 (2004)
23. Mittelmann, H.D.: An independent benchmarking of SDP and SOCP solvers. Mathematical Programming **95**, 407–430 (2003)
24. Nakata, K., Fujisawa, K., Fukuda, M., Kojima, M., Murota, K.: Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results. Mathematical Programming **95**(2), 303–327 (2003)
25. Nayakkankuppam, M.V.: Optimization over symmetric cones. Ph.D. thesis, New York University (1999)
26. Nayakkankuppam, M.V., Tymofyeyev, Y.: A parallel implementation of the spectral bundle method for semidefinite programming. In: Proceedings of the Eighth SIAM Conference on Applied Linear Algebra. SIAM, Williamsburg (VA) (2003)
27. Overton, M.L.: On minimizing the maximum eigenvalue of a symmetric matrix. SIAM Journal on Matrix Analysis and Applications **9**(2) (1988)
28. Parlett, B.M.: The Symmetric Eigenvalue Problem. SIAM (1998)
29. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (New Jersey) (1970)
30. Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method. SIAM Journal on Scientific Computing **13**(1), 357–385 (1992)
31. Toh, K.C.: Solving large scale semidefinite programs via an iterative solver on the augmented systems. SIAM Journal on Optimization **14**(3), 670–698 (2004)
32. Toh, K.C., Kojima, M.: Solving some large scale semidefinite programs via the conjugate residual method. SIAM Journal on Optimization **12**(3), 669–691 (2002)
33. Yamashita, M., Fujisawa, K., Kojima, M.: SDPARA: SemiDefinite Programming Algorithm: paRAllel version. Parallel Computing **29**, 1053–1067 (2003). http://grid.r.dendai.ac.jp/sdpa
34. Zhao, Z., Braams, B.J., Fukuda, M., Overton, M.L., Percus, J.K.: The reduced density matrix method for electronic structure calculations and the role of three-index representability conditions. Journal of Chemical Physics **120**(5), 2095–2104 (2004)