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

Madhu V. Nayakkankuppam

Solving Large-Scale Semidefinite Programs in Parallel

Received: March 06, 2005.

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, 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 (EVO) owing to the fact that storage requirements and solution time of interior-point methods scale poorly with increasing problem size. In contrast, subgradient bundle methods require only nominal amounts of memory and computational time to produce a solution of low accuracy. 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

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

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

Department of Mathematics & Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. E-mail: madhu@math.umbc.edu.

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, 22], resorting to dual-only methods (as opposed to primal-dual) [1], employing a low-rank factorization technique [3, 4], or using carefully preconditioned iterative methods [2, 20, 29, 30] for this linear system. Nevertheless, as a rule of thumb, 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 few hours of solution time on a typical workstation.

On the other hand, Helmberg and Rendl [15] 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. A subgradient bundle method approximates 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 suproblem of determining a search direction typically requires the solution of quadratic program whose size depends on the number of cutting-planes retained, not the original number of variables in (2). The *spectral bundle method* of [15], obtained by specializing the *proximal bundle method* of Kiwiel [19], 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, in particular 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 [24] based partly on Matlab. Numerical results from the present improved implementation show that the techniques proposed here 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 (§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 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 or entry of 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 (and some Matlab built-in function names as well) are employed in pseudocode for algorithms. The $\mathrm{Diag}(\cdot)$ operator maps its vector argument to a diagonal matrix, whereas the $\mathrm{svec}(\cdot)$ 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 [19], and its adaptation to the EVO formulation (2) in 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

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

is a Lipschitz continuously differentiable convex function with the same set of minimizers as f [16]. The unique minimizer $\overline{y}$ in (3), called the proximal point of y, satisfies

$$\nabla f_\rho(y) = \rho\,(y - \overline{y}),$$

and hence represents a displacement of y along the negative gradient of f_ρ. The proximal point algorithm for minimizing f is then to minimize f_ρ by updating the current point y to its proximal point $\overline{y}$. The proximal bundle algorithm approximates $\overline{y}$ by replacing the $f(x)$ term in (3) with a *cutting-plane model* based on the subdifferential [16, 27] 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

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

which minorizes f. Replacing $f(x)$ in (3) by $\hat{f}^k$ in (4) yields

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

as the subproblem whose solution $\bar{y}$ approximates the proximal point of y^k. By interchanging the (unconstrained) minimization and the maximization in (5) [27], this solution may be obtained as

$$\bar{y} = y^k - \frac{1}{\rho}\bar{g} \tag{6}$$

where $\bar{g}$ is an optimal solution to

$$\min_{g\in G^k} \frac{1}{2\rho} \langle \bar{g}, \bar{g} \rangle . \tag{7}$$

If $\bar{y}$ meets the "sufficient descent" criterion

$$f(\bar{y}) \le f(y^k) + m_L \left(\hat{f}^k(\bar{y}) - f(y^k) \right) \tag{8}$$

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

$$f(y^k) - \hat{f}^k(\bar{y}) \le \delta \left(|\, f(y^k) \,| + 1 \right) \tag{9}$$

tests if the upper bound on the maximum attainable descent in this step (left hand side of (9)) falls below a small fraction δ of the magnitude of the objective value (right hand side of (9)), 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 via the process of *aggregation* [19]. Second, for good practical performance, the parameter ρ has to be judiciously varied at every iteration [19]. We outline the algorithm in Algorithm 2.1, deferring these two issues to §8.

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 [18, 19]) *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 $\{y^k\}$ is a minimizing sequence for f, and is convergent to a minimizer of f, if one exists.*

Algorithm 2.1 (Proximal Bundle Method)

(Input) Given a starting point y^0, an improvement parameter $m_L \in (0, 1/2)$, a termination parameter $\delta > 0$, a weight $\rho^0 \geq \rho_{\min} > 0$.
(Output) An approximate minimizer $\tilde{y}$ to (2).
(Initialization) Set the iteration counter $k = 0$. Compute a subgradient $g^0 \in \partial f(y_0)$, and set $G^0 = \{g^0\}$.
for $k = 0, 1, \ldots$ until termination **do**
    (Subproblem) Solve (5) to obtain $\bar{g}$ and $\bar{y}$ via (7) and (6) respectively.
    if (8) holds **then**
        Set $y^{k+1} = \bar{y}$. {serious step}
    else
        Set $y^{k+1} = y^k$. {null step}
    end if
    if (9) is satisfied **then**
        Stop, and return $\tilde{y} = y^k$ as the computed solution.
    end if
    (Update) Add $\bar{g}$ and a new subgradient $g \in \partial f(\bar{y})$ to G^k to obtain G^{k+1}. Update ρ^k to $\rho^{k+1} > \rho_{\min}$.
end for {k loop}

2.2 The Spectral Bundle Method

The *spectral bundle method* of Helmberg and Rendl [15] specializes the proximal bundle method to the objective function in (2); this is the function denoted f hereafter. 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 $[6, 16, 17, 25]$ ε-subdifferential formula

$$\partial_\varepsilon \lambda_{\max}(Z) = \left\{ PVP^T \,:\, V \succeq 0;\ \mathrm{tr}(V) = 1;\ P^T P = I,\ \left\langle Z, PVP^T \right\rangle \geq \lambda_{\max}(Z) - \varepsilon \right\}, \tag{10}$$

for $\varepsilon \geq 0$, and via a chain rule from nonsmooth calculus [16], also to the formula

$$\partial_\varepsilon f(y) = \left\{ AW - b \,:\, W \in \partial_\varepsilon \lambda_{\max}(A^T y - C) \right\}. \tag{11}$$

Thus, eigenvectors to the maximal eigenvalue of $Z = A^T y^k - C$ give rise to exact subgradients ($\varepsilon = 0$), while eigenvectors to nearly maximal eigenvalues yield ε-subgradients. Hence the dominant computational expense of objective function and subgradient evaluations amount to computing a maximal eigenpair of the matrix $Z = A^T y - C$. The Lanczos method (further details in §5) is ideally suited for this purpose, and with only a slightly larger computational expense, can compute the leading few eigenpairs, with corresponding eigenvectors leading to ε-subgradients. Adding several ε-subgradients in the Update step of the algorithm results in an enhanced subdifferential model and convergence in fewer iterations (at a slightly increased cost per iteration). By a harmless abuse of terminology, we refer to both exact and ε-subgradients as simply 'subgradients', dropping the ε prefix. In similar vein, we use the term 'maximal eigenpairs' to denote the largest few eigenvalues 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 $A^T y^j - C$ at the current iterate y^k, and possibly at some past iterates $y^j \in J_k$. Whereas the traditional proximal bundle method employs a polyhedral cutting-plane model G^k (see above (4)) with a typical subgradient of the form $A(p_i^k {p_i^k}^T) - b$, a novel feature of the spectral bundle method [15] is the *semidefinite cutting-plane model*

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

where $\overline{W}$ is an aggregate subgradient. The dominant expense of computing maximal eigenvectors of Z is better justified by using them within a semidefinite cutting-plane model, rather than within a polyhedral model. Consequently, the subproblem — traditionally a quadratic program — now becomes a quadratic, semidefinite program, which can be solved efficiently with an interior-point method. The additional computational expense incurred in solving the subproblem 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) is replaced 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. Thus the primal problem (1) may be rewritten as

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

and its dual as

$$\min_{y \in \mathbb{R}^m} \ a \max \left\{ \lambda_{\max}(A^T y - C),\ 0 \right\} - \langle b, y \rangle \,. \tag{12}$$

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

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

resulting in the final form in which the SDP is stored internally.

4 Data Distribution

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

$$\{1, \ldots, m\} = M_0 \cup \ldots \cup M_{p-1}, \tag{13}$$

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 amount of 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, this scheme can easily determine *a priori* a suitable partitioning in (13) that ensures a good load balance amongst 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 nearly maximal eigenvectors of $Z = A^T y - C$ furnish ε-subgradients of f at y (see (10) and (11)), 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*, best explained in four incremental stages. We first (i) sketch the basic idea behind the Lanczos method in Algorithm 5.1, subsequently (ii) incorporating implicit shifts (Algorithm 5.2) and reorthogonalization, to develop (iii) a block structured Lanczos method that fully exploits block diagonal structure in Z with (iv) an *active block strategy*. The complete procedure is then given in Algorithm 5.3.

5.1 The Lanczos Method

Essentially a Rayleigh-Ritz procedure, the Lanczos method approximates invariant subspaces of Z 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 has unit norm. Owing to the Krylov structure, the Gram-Schmidt procedure to compute an orthonormal *Lanczos basis* $V^j = \left\{v^1, \ldots, v^j\right\}$ for $K^j(Z,v^1)$ simplifies to a three-term recurrence, expressed as a *Lanczos factorization of length* j:

$$ZV^j = V^jT^j + r^je_j{}^T \quad \text{with} \quad V^{jT}r^j = 0, \tag{14}$$

where T^j is a $j \times j$ symmetric, tridiagonal matrix and e_j is the j^{th} coordinate vector. A full spectral decomposition of T^j (computed, for instance, with the symmetric QR algorithm [10]) as

$$S^T T^j S = \Theta \qquad \text{with } S \text{ orthonormal and } \Theta = \text{Diag}(\theta_1, \ldots, \theta_j))$$

yields Ritz values Θ and Ritz vectors $X = V^jS$ as the best approximation to maximal eigenpairs of Z from the subspace $K^j(Z,v^0)$. Since the residual vector u_i for the i^{th} Ritz pair (θ_i, x_i) satisfies

$$u_i = Zx_i - \theta_i x_i = ZV^js_i - \theta_i V^js_i = \left(ZV^j - V^jT^j\right)s_i = r^je_j{}^Ts_i,$$

its norm may be computed, without explicitly forming x_i or u_i, as

$$\|u_i\| = \left\|r^j\right\| |s_{ji}| = \beta^j |s_{ji}|,$$

where $\beta^j = \left\|r^j\right\|$ and s_{ji} is the last component of s_i. Thus, a stopping criterion may check if $\beta^j |s_{ji}|$ falls below a predefined tolerance (and only then generate eigenvectors X); otherwise, the Lanczos factorization (14) is expanded by the addition of one more vector to V^j. Upon convergence, each nonoverlapping Ritz interval $\left[\theta_i - \beta^j |s_{ji}|, \theta_i + \beta^j |s_{ji}|\right]$ contains an eigenvalue of Z. (A slight modification is needed if the Ritz intervals overlap; see [26] for details.) Generally, the larger the component of the initial vector v^0 in the desired eigenspace, the faster the convergence.

We list these basic steps in Algorithm 5.1, referring to [26] for an excellent exposition of the full details.

5.2 Restarts and Reorthogonalization

As the Lanczos factorization is expanded (*i.e.* as the number of columns in V_j increases), Step 9 in Algorithm 5.1 becomes progressively more expensive with respect to computational time and storage, so some form of restarting must be employed. If p eigenpairs are desired, a simple scheme is 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. 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 $Z = A^Ty - C$ 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

Algorithm 5.1 Lanczos Method [26]

(Input) The matrix Z, a normalized initial vector v^0, and a convergence tolerance `tol`.
(Output) Approximate maximal eigenpairs (Θ, X) of Z.
Set $w = Zv^1$; $\quad \alpha^1 = (v^1)^T w$; $\quad r^1 = w - \alpha^1 v^1$; $\quad V^1 = v^1$; $\quad T^1 = \alpha^1$; $\quad \beta^1 = \|r^1\|$.
for $j = 1, 2, \ldots$ until convergence **do**
  $v^{j+1} = r^j / \beta^j$.
  $V^{j+1} \leftarrow [V^j \;\; v^{j+1}]$; $\quad \widehat{T}^j \leftarrow \begin{bmatrix} T^j \\ \beta^j (e_j)^T \end{bmatrix}$.
  $w = Zv^{j+1}$; $\quad t = V^{j+1^T} w$; $\quad r^{j+1} = w - V^{j+1} t$; $\quad \beta^{j+1} = \|r^{j+1}\|$.
  $T^{j+1} \leftarrow [\widehat{T}^j \;\; t]$.
  Compute the spectral factorization $T^{j+1} = S \Theta S^T$.
  If $\beta^{j+1} |s_{j+1,i}| <$ `tol`, break out of the loop.
end for
Compute approximate eigenvectors $X = V^{j+1} S$.

adopted *implicit restarting* [28], a technique by which a Lanczos factorization (V^{p+d}, T^{p+d}) of length $p+d$ is compressed to (V_+^p, T_+^p) of length p in a way that retains in V_+^p the information most relevant to the desired eigenvectors of Z. The ARPACK Users' Guide [21] clearly explains how this procedure is tantamount to a spectral transformation with a filter polynomial of high degree. We summarize this procedure in Algorithm 5.2.

Algorithm 5.2 Implicit Restarting [21]

(Input) A $(p+d)$-step Lanczos factorization (T^{p+d}, V^{p+d}), and the matrix Z.
(Output) A modified p-step Lanczos factorization (T_+^p, V_+^p).
Set $Q = I$. Select the d shifts $\mu_1, \ldots, \mu_d$ to be the d smallest eigenvalues of T_j.
for $i = 1, \ldots, d$ **do**
  $[Q_i, R] = \mathrm{qr}(T^{p+d} - \mu_i I)$.
  $V^{p+d} = Q^T T^{p+d} Q$.
  $Q = QQ_i$.
end for
$\beta_+^p = T_{p+1,p}^{p+d}$; $\quad \sigma^p = Q_{p+d,p}$; $\quad r^p = \beta_+^p v^{p+1} + \sigma^p v^p$.
$V_+^p = V^{p+d} Q_{:,1:p}$; $\quad T_+^p = T_{1:p,1:p}^{p+d}$.

Since rounding errors quickly destroy orthogonality among the Lanczos vectors, a periodic reorthogonalization is necessary [26]. LAMBDA reorthogonalizes the entire Lanczos basis prior to each restart.

5.3 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 exploit this structure in all stages of the algorithm. To the see the inadequacy of the Lanczos method as stated, 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 operator on $\mathbb{R}^{n_1+\ldots+n_s}$ can produce exactly p extremal eigenpairs (call this Method 2), but will likely converge to eigenvectors without block structure when the associated maximal eigenvalue multiple, and split across different blocks of Z. (For example, if Z has a multiple eigenvalue λ 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, 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. 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 exactly p (approximate) maximal eigenpairs of Z.

5.4 Active Block Strategy

It is obvious that blocks for which the Lanczos process has converged (as could happen for small blocks whose sizes are smaller than the size of the Lanczos basis) may be excluded from further computation. By monitoring the Ritz values and their corresponding error bounds [26] 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. This is generally the case in the early stages of the algorithm as the maximal eigenvalue is often simple. Clustering of this eigenvalue usually occurs only when the iterates get close to a minimizer.

5.5 Null Steps

We include a time-saving inexact evaluation procedure for null steps suggested by Helmberg [12]. Since the Ritz values provide progressively better lower bounds on the maximal eigenvalues as the Lanczos method progresses, the error bounds

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

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 [12] for details.

5.6 Parallelism

Parallelism is easily exploited in Step 7 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, other than the root processor 0, uses the locally available A_i $(i \in M_q)$ to compute the partial sum

$$w_q(k) = \sum_{i \in M_q} y_i A_i(k) v^{j+1}(k) \qquad (k \in \mathscr{A}). \tag{15}$$

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

$$w(k) = \sum_{q \neq 0} w_q(k) + \sum_{i \in M_0} y_i A_i(k) v^{j+1}(k) - C(k) v^{j+1}(k) \qquad (k \in \mathscr{A}), \tag{16}$$

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 may now combine all these features and give a full description of the eigenvector computation procedure in Algorithm 5.3. LAMBDA uses some ARPACK [21] routines for the implicit restarts. In particular, the shifts for the implicit restarts for each block are chosen as in Algorithm 5.2 rather than as in (17). As a general-purpose SDP solver, LAMBDA has no particular prior information about the maximal eigenvector, and hence initializes v^1 to be the (blockwise normalized) vector of all ones.

6 Subdifferential Model

LAMBDA employs a mixed polyhedral-semidefinite ε-subdifferential model. In the k^{th} iteration, the bundle consists of a polyhedral part $G^k = [g_1,\ldots,g_l] \in \mathbb{R}^{m \times l}$ and a semidefinite part $P^k \in \mathbb{R}^{n \times r}$, 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 P possibly retained from earlier iterations, *i.e.* each $g_i = A(W_i)$ for some subgradient W_i (of $\lambda_{\max}$) at y^j, and each p_i is a leading eigenvector of $Z = A^T y^j - C$, with y^j being the current, or an earlier, iterate.

Algorithm 5.3 Block Structured Lanczos Method with Active Block Strategy

(Input) A vector y, normalized initial vector v^1, a convergence tolerance `tol`, the number p of eigenpairs desired, the number d of additional Lanczos vectors allowed.
(Output) p triples (θ_j, x_j, k_j) such that θ_j are (approximate) maximal eigenvalues of Z, with each (θ_j, x_j) an (approximate) eigenpair of $Z(k_j)$.
Initialize the active blocks index set $\mathscr{A} = \{1,\dots,s\}$, 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 (15) and (16).
$\alpha^1 = (v^1)^T w; \quad r^1 = w - \alpha^1 v^1; \quad V^1 = v^1; \quad T^1 = \alpha^1; \quad \beta^1 = \|r^1\|.$
$j = 1$.
for rst $= 1,2,\dots$ 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}
      $v^{j+1}(k) = r^j(k)/\beta^j(k); \quad V^{j+1}(k) = [V^j(k) \quad v^{j+1}(k)]; \quad \widehat{T}^j(k) = \left[T^j(k);\ \beta^j(k)(e_j(k))^T\right].$
      Compute $w(k) = Z(k)v^{j+1}(k)$ in parallel using (15) and (16).
      $t(k) = V^{j+1}(k)^T w(k); \quad r^{j+1}(k) = w(k) - V^{j+1}(k)t(k); \quad \beta^{j+1}(k) = \|r^{j+1}(k)\|.$
      $T^{j+1}(k) = [\widehat{T}^j(k) \ \ t(k)].$
    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 $(i,k) \in \{1,\dots,p+d\} \times \{1,\dots,s\}$, test convergence condition $\beta^{p+d}(k)\,|s_{p+d,i}(k)| <$ `tol` to find indices (i,k) corresponding to converged eigenpairs. If p eigenpairs, say $(i_1,k_1),\dots,(i_p,k_p)$, have converged, break out of the rst loop with these index pairs.
  If all eigenpairs in any block $k \in \mathscr{A}$ have converged, set $\mathscr{C} = \mathscr{C} \cup \{k\}$.
  For $k \in \mathscr{A} \cup \mathscr{C}$, compute upper bounds

$$\bar{\lambda}(k) = \max_{i=1\dots,p+d} \theta_i(k) + \beta^{p+d}(k)\,|s_{p+d,i}(k)|\,.$$

  Define ω to be the p^{th} largest of the $\theta_i(k) - \beta^{p+d}(k)\,|s_{p+d,i}| \quad (i = 1,\dots,p+d;\ k \in \mathscr{A})$. 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

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

  If $\underline{\lambda}$ is large enough to violate (8) by some predefined margin, break out of the i loop.
  {null step guaranteed}
  {Lanczos processes are synchronized here}
  For each $k \in \mathscr{A}$, set $\widehat{Q}(k) = I_{p+d}$, and select the indices of the 'unwanted' eigenvalues

$$\mathscr{U} = \left\{(i,k) \ :\ \theta_i(k) + \beta^{p+d}(k)\,|s_{p+d,i}| < \omega\right\}. \tag{17}$$

  for $(i,k) \in \mathscr{U}$ **do**
    $[Q(k), R] = \mathrm{qr}(T^{p+d}(k) - \theta_i(k) I_{p+d})$.
    $T^{p+d}(k) = Q(k)^T T^{p+d}(k) Q(k)$.
    $\widehat{Q}(k) = \widehat{Q}(k)\, Q(k)$.
  end for
  for $k \in \mathscr{A}$ **do**
    $\beta_+^p(k) = T^{p+d}_{p+1,p}(k); \quad \sigma^p = \widehat{Q}_{p+d,p}(k).$
    $r^p(k) = \beta_+^p(k) v^{p+1}(k) + \sigma^p r^p(k)$.
    $V^p(k) = V^{p+d}\widehat{Q}_{:,1:p}(k); \quad T^p(k) = T^{p+d}_{1:p,1:p}(k).$
    Reorthogonalize $V^p(k)$.
  end for{k loop}
  $j = p$.
end for{rst loop}
For each $j \in \{1,\dots,p\}$, reorthogonalize $V^{p+d}(k_j)$.
For each $j \in \{1,\dots,p\}$, compute approximate eigenvectors $x_j = V^{p+d}(k_j)s_{i_j}(k_j)$ to return triples (θ_j, x_j, k_j).

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^k. 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 the aggregation).

Following [23] and dropping the superscripts on y^k, P^k, G^k, ρ^k, we can write the model minimization subproblem (5) as

$$\begin{aligned}
\min_{(\alpha;V)\in\mathbb{R}^l\times\mathbb{S}^r} \quad & \beta \\
\text{s.t.} \quad & \frac{1}{2}\langle[\alpha;\mathrm{svec}(V)], Q[\alpha;\mathrm{svec}(V)]\rangle + \langle[u;\ v], [\alpha;\mathrm{svec}(V)]\rangle - \beta \leq 0 \\
& \sum_{i=1}^{l}\alpha_i + \mathrm{tr}(V) = 1 \\
& \alpha \geq 0;\ V \succeq 0,
\end{aligned} \tag{18}$$

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

$$u_i = \left\langle y - \frac{1}{\rho}b, A(W_i)\right\rangle - \langle C, W_i\rangle \quad (i = 1,\ldots,l), \tag{19}$$

$$v = \mathrm{svec}(P^T(A^T(y - \frac{1}{\rho}b) - C)P)], \text{ and} \tag{20}$$

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix}, \text{ with} \tag{21}$$

$$Q_{11} = \frac{1}{\rho}G^TG, \tag{22}$$

$$Q_{12} = \frac{1}{\rho}\begin{bmatrix} (\mathrm{svec}(P^T(A^Tg_i)P))^T \\ \vdots \\ (\mathrm{svec}(P^T(A^Tg_l)P))^T \end{bmatrix}, \text{ and} \tag{23}$$

$$Q_{22} = \frac{1}{\rho}\sum_{i=1}^{m}\mathrm{svec}(P^TA_iP)(\mathrm{svec}(P^TA_iP))^T. \tag{24}$$

It is obvious from our data distribution scheme and the formulas above that the evaluations of $A(\cdot)$ and $A^T(\cdot)$ in (19), (20) and (23), as well as the orthogonal conjugations in (24) 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 (22) 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 (18) to standard, dual form requires a Cholesky factorization of Q. Pivoting is employed to deal with (nearly) linearly dependent subgradients. Finally, since SeQuL accepts only inequalities in its dual formulation, we incorporated modifications to the algorithm to accommodate the equality constraint in (18).

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 are 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, with 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* [18], and is implemented exactly as described in [23].

Finally, Helmberg [13] has observed, based on extensive numerical experiments, that a suitable update of the proximal parameter ρ is crucial for good practical performance. LAMBDA uses Helmberg's suggested starting value of $\rho = \left\|g^0\right\| / \sqrt{m}$, but Kiwiel's original rule [19] for updating the parameter.

These updates 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 parallel scalability.

Table 1 Description of problems in test set.

Prob	$n_1, \ldots, n_s$	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^+	22 blocks	948	Quantum chemistry
B_2	22 blocks	7230	Quantum chemistry
BeO	22 blocks	7230	Quantum chemistry
C_2	22 blocks	7230	Quantum chemistry
C_2^+	22 blocks	7230	Quantum chemistry
Li_2	22 blocks	7230	Quantum chemistry
LiF	22 blocks	7230	Quantum chemistry
NaH	22 blocks	7230	Quantum chemistry

9.1 Experimental Setup

9.1.1 Test Problems

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

– a fully dense, randomly generated SDP,
– four SDP relaxations 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 combined number of nonzeros in the C, A_i $(i = 1, \ldots, m)$ matrices for rand-1k-8k is about 1.17 millon. This number for the quantum chemistry problems is about 0.85 million, except for BH^+, where it is 0.067 million.

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.) Nevertheless, its solution provides insight into the 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 cache. Each node has 1 GB RAM (shared by two processors), with the entire cluster interconnected by a Myrinet network (2 Gbit/sec peak bandwidth).

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

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 [18] 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 default starting point is always $y^0 = 0$. Exceptions to these default values are clearly mentioned as necessary. For the reader's convenience, captions for tables also summarize key parameter values.

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 in problem data.

9.2 Convergence

In Figure 1, we plot the objective function values for four small 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 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 $r = 30$ vectors 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

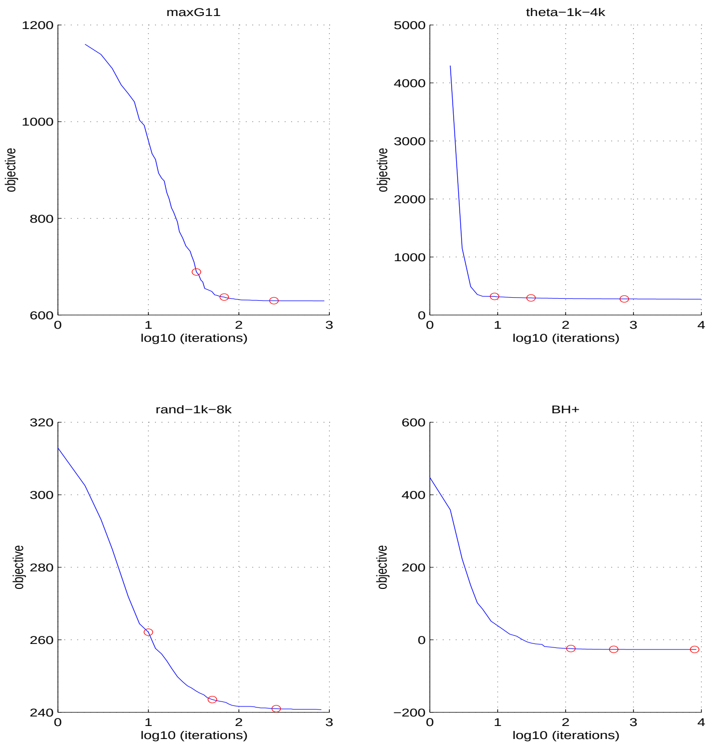


Fig. 1 Typical convergence behavior (serious steps only shown in plot) on four problems.

Table 2 Relative errors 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

7900 extra iterations), but a solution satisfying the $\delta = 0.0001$ accuracy level was unattainable in 10,000 iterations. Relative errors for the other instances in this problem class are similar and are given in Table 2; for f^*, we used the highly accurate optimal values computed in [32].

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 (w/o ABS)	Time (w/ ABS)
BH^+	00:25:09	00:07:40
	00:25:00	00:07:00
B_2	06:14:06	01:33:26
	06:06:40	01:28:28
BeO	13:25:22	03:20:50
	13:20:01	03:20:08
C_2	06:54:15	01:41:43
	06:51:23	01:36:40
C_2^+	01:48:39	01:31:39
	01:43:20	01:28:21
Li_2	02:31:03	02:28:28
	02:26:00	02:23:20
LiF	01:25:13	00:50:03
	01:21:40	00:48:21
NaH	01:28:21	01:23:44
	01:25:01	01:20:01
rand-1k-8k	00:20:36	00:15:36
	00:19:51	00:14:42

9.3 Active Block Strategy

In Table 3, we show the effect of the active block strategy (§5.4) 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 last column gives the percentage of time needed to solve subproblems out of overall 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

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

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.

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 about 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, and therefore a commensurate increase in overall solution time.

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

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 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 large 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.

The speed-up plots in Figure 2 for overall solution time provide a quick picture of scalability. 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, 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 degrade run time. (The only exception is NaH with inexplicably good run times on 32 and 64 processors.) 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) 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 high 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.

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

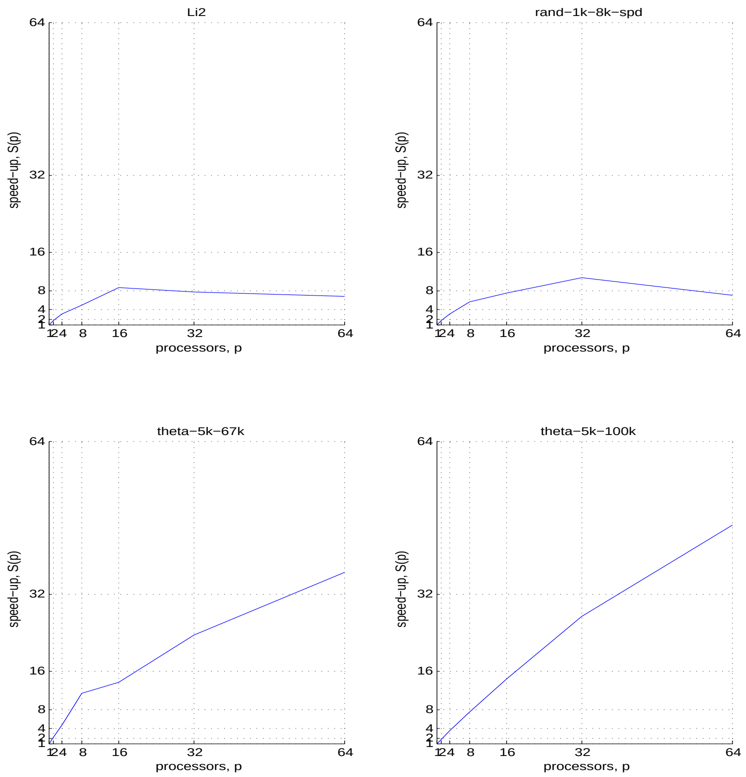
9.6 Fastest Solution Times

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

- the number of iterations (serious steps, ser; total iteratios, 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.

These problems (in addition to others) were originally solved in [32] to high accuracy (6 or 7 digits) using the parallel interior-point code SDPARA [31], with each problem requiring about 14 hours of solution time and 27 GB of memory on

Fig. 2 Speed-up plots for scalability studies.



a 16-cpu machine with slower processors (375 MHz) than KALI's, but 8 MB of L2 cache and 64 GB of shared memory. A recent update [8] to these results, based on some improvements made to SDPARA, has reduced solution time to about 13 hours and memory usage to about 5.7 GB.

10 Perspectives

Subgradient methods, though used for eigenvalue optimization over 30 years ago by Cullum-Donath-Wolfe [5], had not seen widespread use in semidefinite programming until revived by the spectral bundle method of Helmberg-Rendl [15] and improved variants [14] thereof. Our implementation draws much from Helmberg's serial code `SBmethod` [11], 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

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

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 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 keeping many additional subgradients in the polyhedral part of the bundle.

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 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. On the one hand, the block Lanczos method is better at resolving multiple eigenvalues. On the other hand, communicating a block of vectors instead of single vectors several times cuts down on network latency costs. 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 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 have little value in ground state energy calculations, which require 6 or 7 digits of accuracy. 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 second order information, ideally in a 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.

Support via NSF grants DMS-0238008 and DMS-0215373 is gratefully acknowledged. In particular, the computational results presented here would not have been possible without the high-performance cluster KALI, 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). (Technical Report B-308, December 1995)
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., Department of Mathematical and Computing Sciences, Tokyo Institute of Technology (2005). Technical Report B-413
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. Golub, G.H., Van Loan, C.F.: Matrix Computations, second edn. The Johns Hopkins University Press (1989)
11. Helmberg, C.: SBmethod: A C++ implementation of the spectral bundle method. Konrad-Zuse-Zentrum für Informationstechnik Berlin (2000). ZIB Report 00-35

12. Helmberg, C.: Semidefinite programming for combinatorial optimization. Tech. Rep. ZIB Report ZR-00-34, TU Berlin, Konrad-Zuse-Zentrum, Berlin (2000). Habilitationsschrift
13. Helmberg, C.: Numerical evaluation of SBmethod. Mathematical Programming **95**(2), 381–406 (2003)
14. Helmberg, C., Kiwiel, K.C.: A spectral bundle method with bounds. Mathematical Programming **93**(2), 173–194 (2002)
15. Helmberg, C., Rendl, F.: A spectral bundle method for semidefinite programming. SIAM Journal on Optimization **10**(3), 673–696 (1999)
16. Hiriart–Urruty, J., Lemaréchal, C.: Convex analysis and minimization algorithms, vol. I & II. Springer–Verlag (1993)
17. Hiriart–Urruty, J., Ye, D.: Sensitivity analysis of all eigenvalues of a symmetric matrix. Numerische Mathematik **70**, 45–72 (1995)
18. Kiwiel, K.C.: An aggregate subgradient method for nonsmooth convex minimization. Mathematical Programming (1983)
19. Kiwiel, K.C.: Proximity control in bundle methods for convex nondifferentiable minimization. Mathematical Programming **46**, 105–122 (1990)
20. Kočvara, M., Stingl, M.: On the solution of large-scale SDP problems by the modified barrier method using iterative solvers. Tech. Rep. Research Report 304, Institute of Applied Mathematics, University of Erlangen (2005)
21. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK Users' Guide. SIAM, Philadelphia (1998)
22. 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)
23. Nayakkankuppam, M.V.: Optimization over symmetric cones. Ph.D. thesis, New York University (1999)
24. 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)
25. Overton, M.L.: On minimizing the maximum eigenvalue of a symmetric matrix. SIAM Journal on Matrix Analysis and Applications **9**(2) (1988)
26. Parlett, B.M.: The Symmetric Eigenvalue Problem. SIAM (1998)
27. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (New Jersey) (1970)
28. Sorensen, D.C.: Implicit application of polynomial filters in a *k*-step Arnoldi method. SIAM Journal on Scientific Computing **13**(1), 357–385 (1992)
29. 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)
30. 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)
31. 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
32. 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)