A Parallel Implementation of the Spectral Bundle Method for Semidefinite Programming

***Madhu V. Nayakkankuppam**[*] and Yevgen Tymofyeyev[†]*

1 Introduction

There has been a recent, renewed interest in first order cutting-plane methods for semidefinite programs (SDP) and eigenvalue optimization (EVO), primarily owing to the fact that storage requirements and solution time of interior-point methods scale poorly with increase in problem size. Denote the space of real, symmetric, $n \times n$ matrices by $\mathbb{S}^n$, and consider the standard form of the SDP:

$$\min \ \langle C, X \rangle \ \text{ s.t. } \ AX = b; \ \ X \succeq 0, \tag{1}$$

where $C \in \mathbb{S}^n$, $b \in \mathbb{R}^m$ and the linear operator

$$A : \mathbb{S}^n \to \mathbb{R}^m : X \mapsto \begin{bmatrix} \langle A_1, X \rangle \\ \vdots \\ \langle A_m, X \rangle \end{bmatrix}$$

are given. The constraint $X \succeq 0$ means that X is restricted to be positive semidefinite. Helmberg and Rendl [4] observed that any SDP whose feasible set has constant

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

[†]Department of Mathematics & Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. E-mail: ytymof1@math.umbc.edu.

(more generally, bounded) trace a is equivalent to the unconstrained, nonsmooth but convex EVO

$$\min_{y\in\mathbb{R}^m} \quad f(y), \quad f(y) = a\lambda_{\max}\left(A^*y - C\right) - \langle b, y\rangle\,, \tag{2}$$

where

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

denotes the adjoint of A. The function f is amenable to minimization by the classical subgradient bundle methods of convex programming. The resulting algorithm, called the *spectral bundle method*, has been very successful in solving large-scale SDP relaxations from combinatorial optimization [2–4].

We describe a robust, portable, parallel implementation of the spectral bundle method for SDP, based on the Message Passing Interface (MPI). We discuss how the Lanczos method used to compute subgradients may be modified to handle the block-diagonal structure frequently encountered in applications, and how these modifications lead to computational savings when multiple blocks are involved. We also present preliminary numerical results on some large-scale problem instances.

2 Spectral Bundle Method

The spectral bundle method specializes the proximal bundle method with aggregation [6, 7] to (2), where we may assume without loss of generality that $a = 1$. Rayleigh's variational formulation of the maximal eigenvalue

$$\lambda_{\max}(Z) = \max_{\|p\|=1} \langle p, Zp\rangle = \max_{\|p\|=1} \langle pp^*, Z\rangle = \max_{W\succeq 0;\mathrm{tr}(W)=1} \langle W, Z\rangle$$

as the support function of the compact, convex set $\{W \in \mathbb{S}^n \,:\, W \succeq 0, \mathrm{tr}(W) = 1\}$ leads to the well known [5, 9] ϵ-subdifferential formula

$$\partial_\epsilon f(y) = \{AW - b \,:\, \langle W, A^*y - C\rangle \geq \lambda_{\max}(A^*y - C) - \epsilon\} \quad (\epsilon \geq 0).$$

Thus, ϵ-subgradients of f at y^k may be constructed out of eigenvectors to the nearly maximal eigenvalue of $Z = A^*y^k - C$. Now, given an iterate y^k and an inner approximation $W^k \subseteq \partial_\epsilon f(y^k)$, the spectral bundle method uses a proximal cutting-plane model to approximate (2) by the subproblem

$$\min_{y\in\mathbb{R}^m} \max_{W\in W^k} \langle A^*y - C, W\rangle - \langle b, y\rangle + \frac{\rho^k}{2}\left\|y - y^k\right\|^2. \tag{3}$$

Let the solution to this subproblem be $(\bar{y}, \overline{W})$. If $\bar{y}$ produces significant descent in the objective function, y^{k+1} is taken to be $\bar{y}$, this move being termed a *serious step*. Otherwise, we have a *null step*, and y^{k+1} is set to y^k. In either case, a new cutting-plane model W^{k+1} containing $\overline{W}$ and a subgradient $W' \in \partial f(\bar{y})$ is chosen, and (3) is solved again in the next iteration. Several other features, such as aggregating insignificant subgradients in W^k, updating the parameter ρ^k, *etc.* need to be incorporated to develop a practically successful algorithm. For all these details, including the termination criteria, we refer the reader to [4, 6, 7]. The sequence of serious steps thus generated by the algorithm is a minimizing sequence for f.

3 Parallelization

We now consider SDP's with the usual block-diagonal structure, *i.e.* the matrices C and $A_i \quad (i = 1, \ldots, m)$ are block-diagonal with block sizes $n_1, \ldots, n_s$, and $\mathbb{S}^n$ in (1) is replaced by $\mathbb{S}^{n_1} \times \ldots \times \mathbb{S}^{n_s}$. We first discuss data distribution, then the modifications required in the Lanczos method to handle block-diagonal structure.

3.1 Data Distribution

We distribute the constraint matrices $A_i \quad (i = 1, ..., m)$ among, say, t available processors, *i.e.,* the index set $\{1, \ldots, m\}$ is partitioned into a disjoint union $M_1 \cup \ldots \cup M_t$, with processor k containing the matrices $A_i \quad (i \in M_k)$. Without further *a priori* information about problem structure, this distribution scheme allows an equitable distribution of computation and memory usage.

3.2 Block Structure

The dominant cost in each iteration of the bundle method is the calculation of the eigenvectors of the nearly maximal eigenvalues of the sparse, symmetric, block-diagonal matrix $Z = C - \sum_{i=1}^{m} y_i A_i$; these eigenvectors may then be used to construct ϵ-subgradients. This can be done efficiently with the Lanczos method without explicitly forming Z, but using only matrix-vector products; see [10, 11] for full details.

In the present context, some modifications need to be made to the Lanczos method to handle block-diagonal structure effectively. To see this clearly, let $Z \in S^{n_1} \times \ldots \times S^{n_s}$ have s diagonal blocks. Suppose we want the r largest eigenvalues and corresponding eigenvectors of Z. A naïve approach is to merely compute the r largest eigenvalues of each block using the Lanczos method and then pick the r largest eigenvalues out of them. This is clearly inefficient as we compute rs eigenvalues, but use only r of them.

Alternatively, we may use a Lanczos process on the whole matrix Z, exploiting the block structure of Z in the matrix vector products. The principal difficulty here lies in multiple eigenvalues that are split across blocks — a frequent phenomenon near the minimizer, where the largest eigenvalue of Z is generically multiple. In this case, the Lanczos method will invariably converge to an eigenvector *without* block structure. While such an eigenvector still yields a legitimate subgradient, there is a significant linear algebra overhead to be incurred in other parts of the algorithm owing to the loss of block structure. The real situation is somewhat more complicated because the Lanczos process, upon convergence, yields a set of r eigenvectors with nonzeros of varying magnitudes in each block. Without *a priori* knowledge of the multiplicity of the maximal eigenvalue of Z (let alone its blockwise multiplicities), constructing *eigenvectors with block structure* from the *Lanczos vectors without block structure* is a tricky task.

We incorporate modifications into the Lanczos process to handle block-diagonal matrices; we call this the Block Structured Lanczos Method. This method essentially performs a synchronized set of s ordinary Lanczos processors, one for each

block, maintaining Lanczos vectors independently for each block. The j^{th} iteration of the k^{th} Lanczos process (for the k^{th} block) requires matrix-vector products of the form $w_j^k = Z^k u_j^k$ for some Lanczos vector $u_j^k \in \mathbb{R}^{n_k}$. Recall that $Z = \sum_{i=1}^{m} y_i A_i - C$, which depends on the distributed constraint matrices A_i, is not available explicitly. Each processor q, which contains A_i $(i \in M_q)$, computes the partial sum

$$v_j^k = \sum_{i \in M_q} y_i A_i u_j^k$$

for each block $k = 1, \ldots, s$. These contributions v_j^k are subsequently collected on the root processor which computes the desired matrix-vector product

$$w_j^k = C u_j^k - \sum_q v_j^k$$

for each block $k = 1, \ldots, s$. These ideas are sketched in the algorithm below.

Algorithm 3.1 (Block Structured Lanczos Method).

1: *Given* l, d $(l < d)$
2: *Start with an initial vector* u_0 *and* $w_0 = Zu_0$
3: ***for*** $j = 1, 2, \ldots$ *(until convergence)* ***do***
4: ***for*** $k = 1, 2, ..., s$ ***do***
5: ***if*** *length of Lanczos factorization is* d ***then***
6: *Perform implicit restart*
7: ***end if***
8: *Use* u_j^k *and* w_j^k *to expand the Lanczos factorization by one*
9: *Obtain* u_{j+1}^k *needed for next Lanczos iteration*
10: ***end for***
11: *Compute* $w_{j+1} = Zu_{j+1}$ *in parallel*
12: ***end for***

In Step 6, we employ ARPACK's implicit restart mechanism in our implementation; see [8] for details. In Step 8, we perform a complete reorthogonalization of the Lanczos vectors by incorporating Householder transformations into the Lanczos procedure; see Golub and van Loan [1, p.482].

3.3 Active Block Strategy

By monitoring the Ritz values and their corresponding error bounds [10] in every block prior to each implicit restart, we can ignore those blocks which are no longer candidates for producing one of the largest r eigenvalues. The remaining blocks are called "active blocks", and subsequent computation is restricted to these blocks only. This can produce significant savings (as demonstrated in §4.2) in problems with many blocks, but where the largest r eigenvalues are concentrated in a small number of blocks. This is usually the case in the early iterations when the maximal eigenvalue is well separated from the second largest one.

Finally, we remark that we do not employ the conventional methods of accelerating convergence of the Lanczos process (*e.g.* Chebyshev acceleration), as these involve several (depending on the degree of the chosen polynomial) additional matrix-vector products. While these techniques are beneficial in the serial context, they appear not to be viable in a distributed setting, where every matrix-vector product involves inter-processor communication.

3.4 Solution of the Subproblem

To form the data defining the subproblem (3) at each iteration, each processor computes its contribution using the constraint matrices available in its local memory. These contributions are all amalgamated on the root processor. The subproblem is solved with SDPT3 [12], a Matlab-based code that implements a primal-dual, path-following interior-point algorithm. Since the subproblem is of a fixed size (depending on the number of subgradients in the model W^k), solving it with an interior-point method, even to high accuracy, is a negligible cost compared to the overall cost of one bundle iteration.

4 Numerical Results

4.1 Test Problems and Hardware

Our overall test set consists of ten problems; the characteristics of each problem are described in Table 1. This set includes randomly generated problems (sparsity denotes the approximate ratio of nonzero entries to total number of entries in each A_i), SDP relaxations of the Lovász ϑ-function and the Max-Cut problem, and SDP relaxations arising in quantum chemistry. The graphs on which G11, G51, G32, G60 and th1 are based were generated using the graph generator program `rudy`, written by G. Rinaldi. Some of these problems are available on the SDPLIB web page `http://www.nmt.edu/~sdplib`. The quantum chemistry problem (q20) was generated based on a description provided by B. J. Braams (New York University). All runs of our code were conducted on Chiba City, Argonne National Lab's 512-processor scalable Beowulf cluster. Each computing node on Chiba City has dual 500 MHz Pentium III processors, with a 512 KB cache (per processor), 512 MB of RAM, and 9GB of local disk storage. Our runs were conducted over switched fast Ethernet (although Chiba City is fully Myrinet-enabled). Detailed specifications of this system are available on the Chiba City web page `http://www-unix.mcs.anl.gov/chiba`.

4.2 Results

We report here some preliminary computational results. In Table 2, we show results on Max-Cut problems solved in serial. The first two columns tabulate the problem name and the number of serious / total iterations (Ser/Tot). In the remaining columns, we tabulate the subgradient norm $\|g\|$, the value of the penalty parameter ρ, and the objective value (obj), all measured at the end of the last serious step.

Prob	m	$n_1, ..., n_s$	Description
G11	800	800	Max-Cut
G51	1,000	1000	Max-Cut
G32	2,000	2000	Max-Cut
G60	7,000	7000	Max-Cut
r1	16,384	500, 500, 500, 500	Random (sparsity 1.0e-06)
r2	8,192	150 blocks of size 50	Random (sparsity 1.0e-03)
r3	8,192	250, 250, 250, 250	Random (sparsity 1.0e-04)
th1	19,899	2001	Lovász ϑ-function
th2	67,489	5001	Lovász ϑ-function
q20	98,556	20, 190, 190, 400	Quantum chemistry

Table 1. *Description of problems in test set.*

We do not report solution times, as our data distribution scheme and present implementation do not fully exploit the structure of these problems effectively. The purpose of this table is rather to illustrate the typical values of important quantities associated with the computed solution.

Prob	Ser/Tot	$\|g\|$	ρ	obj
G11	52/123	0.002	3.5e-05	629.93
G51	42/117	0.002	4.5e-04	4008.88
G32	125/228	0.003	2.4e-04	1574.86
G60	190/190	0.006	5.0e-06	15979.11

Table 2. *Max-Cut problems solved on 1 processor.*

In Table 3, we show the performance of the Block Structured Lanczos Method for eigenvalue computation of block-diagonal matrices on 16 processors. We tabulate the problem name, the number of diagonal blocks in the matrix, the CPU time taken to compute the 5 largest eigenvalues with and without the active block strategy, and the corresponding savings in computational time.

Prob	#	time for e.v.		%
	blocks	on	off	saving
q20	4	1:48:14	1:53:15	5
r1	4	1:31:32	1:58:54	23
r2	50	1:48:56	2:29:00	27

Table 3. *Block Structured Lanczos Method with the Active Block Strategy on 16 processors.*

In Table 4, we provide profiling and speed-up studies on four problems, varying the number of processors from 1 up to 32 in powers of two. For each problem, we

have three rows corresponding to total solution time, time for eigenvalue and eigenvector computation, and the time taken to form the data for the subproblem. (The time taken to solve the subproblem is not a significant fraction of the overall solution time.) The first three columns tabulate the problem name, the percentage of total solution time used by the dominant parts of the algorithm. The remaining columns, titled 1 through 32 in powers of two, denote the number of processors used. For one processor, we report the actual wall-clock time taken (in the hours:minutes:seconds format), whereas the remaining columns record speed-up factors:

$$\text{Speed-up on } p \text{ processors} = \frac{\text{Time taken on 1 processor}}{\text{Time taken on } p \text{ processors}}.$$

Prob	task	%	1	2	4	8	16	32
r1	total	100	23:30:24	1.92	3.01	5.70	8.51	10.1
	eig.v.	93	21:50:47	1.97	2.94	5.74	8.90	10.8
	subpr.	.001	00:05:48	1.98	3.12	5.92	10.0	10.5
th1	total	100	01:51:00	2.04	3.70	5.79	9.65	9.11
	eig.v.	83	01:32:43	1.97	3.43	5.64	10.1	9.50
	subpr.	2	00:02:36	1.60	3.10	5.37	9.75	10.4
th2	total	100	—	—	—	10:33:03	11.2	20.6
	eig.v.	91	—	—	—	09:34:12	11.1	21.1
	subpr.	3	—	—	—	00:21:17	12.2	25.3
q20	total	100	—	—	15:19:14	13.5	17.3	36.6
	eig.v.	66	—	—	8:35:36	11.0	11.7	27.8
	subpr.	9	—	—	1:18:49	22.1	50.0	167
r3	total	100	13:39:38	1.91	2.43	5.27	8.68	8.60
	eig.v.	94	12:57:58	1.93	2.46	5.69	10.1	10.0
	subpr.	1	00:07:55	1.94	2.58	6.08	12.1	12.1

Table 4. *Speed-up and profiling studies.*

Several remarks are in order.

– The code does not produce exactly the same sequence of iterates when run of different numbers of processors. Rounding errors in the Lanczos process sometimes produce slightly different eigenvectors. In some cases, this difference is enough to change a null step into a serious step or vice versa. This may further trigger or undo an update of the penalty parameter ρ. As a consequence, the observed iterate sequence could be quite different on different numbers of processors. In particular, there are variations in the number of serious steps and the total number of iterations. In Table 4, we truncate (*a posteriori*) all runs of a given problem at the same number of iterations.

– For q20, the solution time on one and two processors was unaffordable, so we started with four processors, and extrapolated, as is customary in scalability

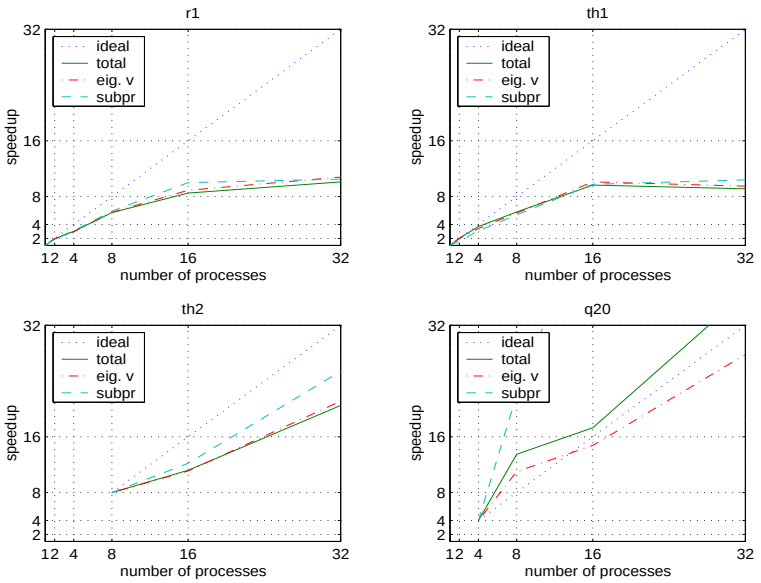

Figure 1. *Speed-up plots for Table 4.*

studies, the solution time on one processor as four times the solution time on four processors. Subsequent speed-up factors are based on this extrapolation. It should be noted that the block structure for this problem is small enough that the QR method could likely be faster than a Lanczos method for eigenvector computation. (However, for consistency, we used the Block Structured Lanczos Method in all our runs.)

The algebraic multiplicity of the largest eigenvalue at the optimal solution is very high. This feature, as we have observed in other smaller instances, seems to be typical in the quantum chemistry application. In this case alone, the algorithm terminated with the warning that the maximum number of iterations allowed (300) had been reached. Hence, the computed objective is likely to be only a coarse bound on the optimal objective value.

At the optimal point, the highly multiple maximal eigenvalue is split across all four blocks of the problem, thus diminishing the advantages of the active block strategy, at least towards the final stages of the algorithm.

The timings (wall-clock time) for 4 and 8 processors are unreliable as about 25% and 20% respectively of the overall time is unexplained. We are presently unable to pinpoint the reasons for these anomalies, which are not observed for 16 and 32 processors. It is possible that overall solution times for 4 and 8 processors are overestimated due to unusually high network congestion or other system problem.

Nevertheless, we observe excellent run times of 3:31:32 and 1:40:25 on 16 and 32 processors, respectively. In fact, this problem is sufficiently large that we can anticipate good speed-up up to 128 processors.

5 Conclusions

We have described a general-purpose, portable, parallel implementation of the spectral bundle method for semidefinite programs. We discussed modifications needed in the Lanczos process to handle the commonly encountered block-diagonal structure. These modifications result in the Active Block Strategy, which could yield computational savings in problems with multiple blocks. Our preliminary numerical results indicate that the capabilities of the spectral bundle method may be significantly extended by properly exploiting parallelism. These numerical experiments were conducted over a fast Ethernet connection. Improved speed-up factors may be obtained with a high-speed interconnect such as Myrinet, and by using more advanced communication primitives in MPI. Further investigations are needed to explore the possibility of overlapping communication with computation within the algorithm.

Acknowledgments

We thank Jorge Moré and Argonne National Lab for the computational resources used for the preliminary numerical results reported here. This work was partially supported by UMBC's Designated Research Initiative Fund.

Bibliography

[1] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, second edition, 1989.

[2] C. Helmberg. Numerical evaluation of SBmethod. *Mathematical Programming*, 95(2):381–406, 2003.

[3] C. Helmberg and K. Kiwiel. A spectral bundle method with bounds. *Mathematical Programming*, 93(2):173–194, 2002.

[4] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 1999.

[5] J–B. Hiriart–Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms*, volume I & II. Springer–Verlag, 1993.

[6] K. C. Kiwiel. An aggregate subgradient method for nonsmooth convex minimization. *Mathematical Programming*, 1983.

[7] K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46:105–122, 1990.

[8] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[9] M. L. Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 9(2), 1988.

[10] B. M. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1998.

[11] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halsted Press, New York, 1992.

[12] R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 1995(2):189–217, 2003.