A parallel interior point decomposition algorithm for block angular semidefinite programs

Kartik Krishnan Sivaramakrishnan [1]
Department of Mathematics
North Carolina State University
Raleigh, NC 27695-8205
kksivara@ncsu.edu
http://www4.ncsu.edu/~kksivara

Last modified: April 2008
Accepted for publication in *Computational Optimization and Applications*

Abstract

We present a two phase interior point decomposition framework for solving semidefinite (SDP) relaxations of sparse maxcut, stable set, and box constrained quadratic programs. In phase 1, we suitably modify the *matrix completion* scheme of Fukuda et al. [11] to preprocess an existing SDP into an equivalent SDP in the block-angular form. In phase 2, we solve the resulting block-angular SDP using a *regularized interior point decomposition* algorithm, in an iterative fashion between a master problem (a quadratic program); and decomposed and distributed subproblems (smaller SDPs) in a parallel and distributed high performance computing environment. We compare our MPI (Message Passing Interface) implementation of the decomposition algorithm on the distributed *Henry2* cluster with the OpenMP version of CSDP [6] on the IBM Power5 shared memory system at NC State University. Our computational results indicate that the decomposition algorithm a) solves large SDPs to 2-3 digits of accuracy where CSDP runs out of memory; b) returns competitive solution times with the OpenMP version of CSDP, and c) attains a good parallel scalability. Comparing our results with Fujisawa et al. [10], we also show that a suitable modification of the matrix completion scheme can be used in the solution of larger SDPs than was previously possible.

Keywords: Block angular semidefinite programs, matrix completion, decomposition and nonsmooth optimization, interior point methods, parallel optimization.

[1]Supported by a startup grant from NC State University.

1 Introduction

Semidefinite programming (SDP) is the problem of maximizing a linear objective function in symmetric matrix variables subject to linear equality constraints and also convex constraints that require these variables to be positive semidefinite. The tremendous activity in SDP was spurred by the discovery of efficient interior point algorithms (see De Klerk [9]); development of associated software (see Borchers [4], Sturm [30], Toh et al. [31], and Yamashita et al. [34]) for solving large scale instances; and important applications (see De Klerk [9]) in science and engineering.

Primal-dual interior point methods (IPMs) (see De Klerk [9] and Monteiro [23]) are currently the most popular techniques for solving SDPs. However, current SDP solvers based on IPMs can only handle problems with dimension n and number of equality constraints m up to a few thousands. Each iteration of a primal-dual IPM solver needs to form a dense Schur matrix, store this matrix in memory, and finally factorize and solve a dense system of linear equations of size m with this coefficient matrix. A number of SDPs are in the *block-diagonal* form (see Toh et al. [31]) and current SDP codes exploit the numerical linear algebra in IPMs to solve larger instances of these problems. However, regardless of the sparsity of the underlying SDP, one needs to store the primal matrix variable and the Schur matrix, that are always dense matrices of sizes n and m, respectively. As a result, *even sparse* SDPs where $n > 5000$ and $m > 10000$ cannot be solved by serial implementations of primal-dual IPMs on typical workstations. Several techniques have recently been developed to solve large scale SDPs; these include first order approaches (see Burer & Monteiro [7], Helmberg et al. [14, 15, 16] and Nayakkankuppam [25]) and parallel implementations of primal-dual IPMs on shared memory (see Borchers & Young [6]) and distributed memory (see Yamashita et al. [34]) systems.

In this paper, we present a two stage decomposition framework for solving SDP relaxations of sparse maxcut, stable set, and box constrained quadratic problems with structured sparsity. We first introduce two concepts: *block-angular* SDPs and the *matrix completion* procedure that are needed in our algorithm.

Block-angular SDPs are special instances of block-diagonal SDPs where the constraints can be classified into two sets: *coupling* constraints that couple two or more of the blocks, and *independent* constraints that involve only the variables in a block. There is growing interest in *block-angular* SDPs and we introduce the basic problem in Section 2.1. To our knowledge, current IPM implementations do not exploit the block-angular structure in the underlying SDP. Block-angular SDPs also arise in stability analysis of interconnected systems (Langbort et al. [20]) and two stage stochastic semidefinite programs with recourse (Mehrotra and Özevin [21]). Moreover, Waki et al. [33] have recently developed a hierarchy of block-angular SDPs for solving sparse polynomial optimization problems.

Recently, Fukuda et al. [11] have developed a matrix completion scheme that exploits the sparsity in the data matrices of an SDP to preprocess it into an equivalent SDP having multiple but smaller matrix variables. Detailed computational experiments with the matrix completion scheme are available in Fujisawa et al. [10] and Nakata et al. [24]. One drawback of the matrix completion scheme is that the preprocessed SDP has many more equality constraints than the original SDP, and this problem can no longer be solved by serial implementations of existing primal-dual IPMs. For some SDPs, as we explore in this paper, the matrix completion scheme actually gives a preprocessed SDP that is in the block-angular form. To our knowledge, none of the existing algorithms exploit this feature to solve large scale SDPs.

We now present the main idea behind our two stage decomposition algorithm: In the first stage, we employ a modified version of the matrix completion scheme of Fukuda et al. [11] to preprocess the existing SDP into an equivalent SDP in the block-angular form. The block-angular SDP has a special structure that can be exploited by decomposition. In the second stage, we employ a regularized interior point decomposition algorithm to solve the preprocessed SDP in an iterative fashion between a master problem (a quadratic program over the coupling constraints), and *decomposed* and *distributed* subproblems (smaller SDPs over independent constraints) in a parallel and distributed computing environment. The decomposition algorithm in the second stage is reminiscent of the bundle and decomposition approaches that have been applied to solve mixed integer and stochastic LPs in the literature (see Nowak [26] and Ruszczyński & Shapiro [29]).

Our objective in this paper is to demonstrate via computational experiments that our two stage algorithm can: (a) solve large SDPs to $2-3$ digits of accuracy more quickly than serial implementations of primal-dual IPMs such as CSDP (see Borchers [4]) including problems where the latter methods run out of memory; (b) achieve solution times comparable to parallel implementations of primal-dual IPMs such as the OpenMP version of CSDP (see Borchers & Young [6]) while requiring considerably less memory; and (c) achieve a good parallel scalability on large SDPs. We also demonstrate that the matrix completion scheme can be used in the solution of much larger SDPs than was previously possible.

The paper is organized as follows: Section 2 describes a block-angular SDP and a decomposition framework for solving block-angular optimization problems. Section 3 describes our modified version of the matrix completion scheme of Fukuda et al. [11] that is used to process certain SDPs into equivalent block-angular problems. Section 4 presents the detailed decomposition algorithm for solving block-angular SDPs. Section 5 discusses our MPI (Message Passing Interface) implementation, parameter values, and computational results on large SDP relaxations of sparse maxcut, stable set, and box constrained quadratic problems. We also compare our parallel algorithm with the OpenMP version of CSDP on a shared memory system. Finally, Section 6 presents our conclusions and future work.

2 Decomposition framework for block-angular SDPs

Let $I\!R^n$ and $\mathcal{S}^n$ denote the n dimensional Euclidean space and the space of symmetric matrices of size n, respectively.

2.1 Block-angular SDPs

Consider the following block angular semidefinite program

$$\begin{array}{lrcl} \max & \sum_{i=1}^{r} C_i \bullet X_i & & \\ \text{s.t.} & \sum_{i=1}^{r} \mathcal{A}_i(X_i) & = & b, \\ & \mathcal{B}_i(X_i) & = & d_i, \;\; i=1,\ldots,r, \\ & X_i & \succeq & 0, \;\; i=1,\ldots,r \end{array} \tag{1}$$

and its dual

$$\begin{array}{lrcl} \min & b^T y + \sum_{i=1}^{r} d_i^T w_i & & \\ \text{s.t.} & \mathcal{A}_i^T y + \mathcal{B}_i^T w_i & \succeq & C_i, \;\; i=1,\ldots,r \end{array} \tag{2}$$

where $C_i, X_i \in \mathcal{S}^{n_i}$; $b, y \in I\!R^m$; $d_i, w_i \in I\!R^{m_i}$, $i = 1,\ldots,r$. For $C, X \in \mathcal{S}^n$; we have $C \bullet X = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij}$. For $i = 1,\ldots,r$, the linear operator $\mathcal{A}_i : \mathcal{S}^{n_i} \rightarrow I\!R^m$ and its adjoint $\mathcal{A}_i^T : I\!R^m \rightarrow \mathcal{S}^{n_i}$ are

$$\mathcal{A}_i(X) \;=\; \begin{bmatrix} A_{i1} \bullet X \\ \vdots \\ A_{im} \bullet X \end{bmatrix} \quad \text{and} \quad \mathcal{A}_i^T y \;=\; \sum_{j=1}^{m} y_j A_{ij},$$

respectively, where the matrices $A_{ij} \in \mathcal{S}^{n_i}$, $j = 1,\ldots,m$. The linear operators $\mathcal{B}_i : \mathcal{S}^{n_i} \rightarrow I\!R^{m_i}$ and their adjoints $\mathcal{B}_i^T : I\!R^{m_i} \rightarrow \mathcal{S}^{n_i}$, $i = 1,\ldots,r$ can be defined similarly. The constraint $X \succeq 0$ indicates that the matrix X is positive semidefinite, i.e., all its eigenvalues are nonnegative. Finally, $\text{trace}(X) = \sum_{i=1}^{n} X_{ii}$ is the trace of a matrix $X \in \mathcal{S}^n$.

We assume that (1) and its dual (2) have strictly feasible (Slater) points (see DeKlerk [9]). This ensures that the primal and dual objective values are equal at optimality (zero duality gap), and that both problems attain their optimal solution.

The constraint set of (1) comprises of r independent sets of the form $\mathcal{C}_i = \{X_i \in \mathcal{S}^{n_i} : \mathcal{B}_i(X_i) = d_i, X_i \succeq 0\}$, $i = 1,\ldots,r$. We will further assume that the sets $\mathcal{C}_i$ also have Slater points, and are bounded, i.e., every feasible X_i in each $\mathcal{C}_i$ satisfies $\text{trace}(X_i) = 1$. The constraints $\sum_{i=1}^{r} \mathcal{A}_i(X_i) = b$ couple the r subsystems. If these constraints were absent, the problem (1) can be viewed as r independent subproblems in the X_i variables.

2.2 Regularized decomposition algorithm

The aim of decomposition is to solve a *dual* problem obtained by introducing the coupling constraints into the objective function via Lagrangian multipliers $y \in I\!R^m$. This gives us the dual problem

$$\min_{y \in I\!R^m} \quad \theta(y) \quad = \quad b^T y + \sum_{i=1}^{r} \max_{X_i \in \mathcal{C}_i} ((C_i - \mathcal{A}_i^T y) \bullet X_i) \tag{3}$$

where $\theta(y)$ is a *convex* but *nonsmooth* function. Given a point $y^j \in I\!R^m$, computing the value $\theta(y^j) = b^T y^j + \sum_{i=1}^{r} \theta_i(y^j)$ requires the solution of r *independent* semidefinite programs of the form

$$\theta_i(y^j) \quad = \quad \max_{X_i \in \mathcal{C}_i} ((C_i - \mathcal{A}_i^T y^j) \bullet X_i). \tag{4}$$

Problem (4) is the ith *subproblem* in the decomposition scheme. We use the assumption that the sets $\mathcal{C}_i$ have Slater points to arrive at (4). Since the $\mathcal{C}_i$ are bounded, the subproblems (4) cannot be unbounded. If any subproblem (4) is infeasible, the original block angular SDP (1) is also infeasible. Let $D_i(y^j)$ denote the set of optimal solutions to (4). The subdifferential $\partial\theta_i(y^j)$ of $\theta_i(y)$ at $y = y^j$ (see Ruszczyński [28]) can be computed as

$$\partial\theta_i(y^j) \quad = \quad -\mathcal{A}_i(D_i(y^j)). \tag{5}$$

Typically in the jth iteration, one does not have knowledge of the entire set $\partial\theta_i(y^j)$. Instead, one obtains an $X_i^j \in D_i(y^j)$ by solving (4) with a primal-dual IPM, and then calculates a subgradient $g_i(y^j) = -\mathcal{A}_i(X_i^j)$ that is a member of this set. These subgradients can be combined together to give a subgradient

$$g(y^j) \quad = \quad (b + \sum_{i=1}^{r} g_i(y^j)) \quad = \quad (b - \sum_{i=1}^{r} \mathcal{A}_i(X_i^j)) \tag{6}$$

for the function $\theta(y)$. In the kth iteration of the algorithm, given a set of points y^j, $j = 1, \ldots, k$, a coordinating agency called the *master problem* uses the function values $\theta(y^j)$ and the subgradients $g(y^j)$ at these points to construct a model

$$\begin{aligned} \theta^k(y) \quad &= \quad b^T y + \sum_{i=1}^{r} \max_{j \in J^k(i)} \{(C_i - \mathcal{A}_i^T y) \bullet X_i^j\} \\ &= \quad b^T y + \sum_{i=1}^{r} \theta_i^k(y), \end{aligned} \tag{7}$$

where $J^k(i) \subseteq \{1, 2, \ldots, k\}$ is the set of iteration indices when $\theta_i^k(y)$ is updated. The function $\theta^k(y)$ is an *underestimate* for $\theta(y)$, i.e., $\theta^k(y) \le \theta(y)$ with equality holding at $y = y^j$, $j = 1, \ldots, k$. In the original Dantzig-Wolfe decomposition scheme (see Bertsekas [2]), the master problem minimizes $\theta^k(y)$ in every iteration. However, the Dantzig-Wolfe scheme is known to converge very slowly (see Hiriart-Urruty & Lemaréchal [17]). We consider the following regularized master problem

$$\min_{y \in I\!R^m} \quad \theta^k(y) + \frac{u^k}{2}||y - x^k||^2 \tag{8}$$

where $u^k > 0$ is a suitable weight and x^k is the current center to overcome this limitation. The center x^k is updated depending on the relation between the value of $\theta(y^{k+1})$ at the current solution y^{k+1} to (8), and the prediction $\theta^k(y^{k+1})$ provided by the current model. If these values are close, we set $x^{k+1} = y^{k+1}$ (*serious step*); else, we set $x^{k+1} = x^k$ (*null step*). In either case, we update the model $\theta^k(y)$ with the function and subgradient values at y^{k+1}.

The regularized master problem (8) can also be written as the following separable quadratic program (QP)

$$\begin{array}{lrcl} \min & b^Ty + \sum_{i=1}^{r} z_i + \frac{u^k}{2}||y - x^k||^2 & & \\ \text{s.t.} & \mathcal{A}_i(X_i^j)^T y + z_i & \geq & C_i \bullet X_i^j, \quad i = 1, \ldots, r, \;\; j \in J^k(i). \end{array} \tag{9}$$

If $\theta_i(y^{k+1}) = \theta_i^k(y^{k+1})$, then the solution y^{k+1} is optimal in the ith subproblem and we do not update the model function $\theta_i^k(y)$ in the $(k+1)$th iteration. Else, the values of $\theta_i(y^{k+1})$ and $g_i(y^{k+1}) \in \partial\theta_i(y^{k+1})$ obtained from the solution to the ith subproblem are used to update $\theta_i^k(y)$. In this case, the ith subproblem adds the following cutting plane

$$\mathcal{A}_i(X_i^k)^T y + z_i \quad \geq \quad C_i \bullet X_i^k \tag{10}$$

to (9). One can combine the cutting planes generated by all the subproblems to add only one cutting plane to the master problem in each iteration, but we prefer to add each cut individually. Although, this procedure increases the size of the master problem and requires more memory, we found that it improves the overall convergence of the algorithm. Let (y^{k+1}, z^{k+1}) and λ^{k+1} be the solution to (9) and its dual, respectively.

The function value $\theta(x^k)$ provides an upper bound on the optimal objective value of (3). On the other hand, the value $\theta^k(y^{k+1}) = b^Ty^{k+1} + \sum_{i=1}^r z_i^{k+1}$ computed from the solution (y^{k+1}, z^{k+1}) to the master problem (9) provides a lower bound on the optimal objective value. Hence, one can use the following criteria in the kth iteration

$$\frac{\theta(x^k) - \theta^k(y^{k+1})}{\max\{1, |\theta(x^k)|\}} \quad \leq \quad \epsilon \tag{11}$$

to terminate the algorithm, where $\epsilon > 0$ is a suitable parameter.

The subproblems and the master problem are much smaller than the original problem. Hence, even if the decomposition algorithm requires several iterations to reach the solution, each iteration is usually faster and requires considerably less memory storage than an IPM iteration on the original problem. We present our complete algorithm in Section 4.1.

3 Preprocessing SDPs into block-angular form via matrix completion

In this section, we briefly describe the matrix completion scheme of Fukuda et al. [11], and show that it can be suitably modified to preprocess SDP relaxations of maxcut, stable set, and box constrained quadratic programs into equivalent problems having a block-angular structure. We illustrate the procedure with a simple example below. Consider the following semidefinite relaxation

$$\begin{array}{lrcl} \max & C \bullet X & & \\ \text{s.t.} & X_{ii} & = & 1, \ \ i = 1, \ldots, n, \\ & X & \succeq & 0 \end{array} \tag{12}$$

of the maxcut problem (see Goemans & Williamson [12]) over a graph $G = (V, E)$. The graph is shown on the left hand side of Figure 1 and $n = |V|$ is the number of vertices in the graph. The coefficient matrix $C \in \mathcal{S}^n$ is the Laplacian matrix of the graph whose

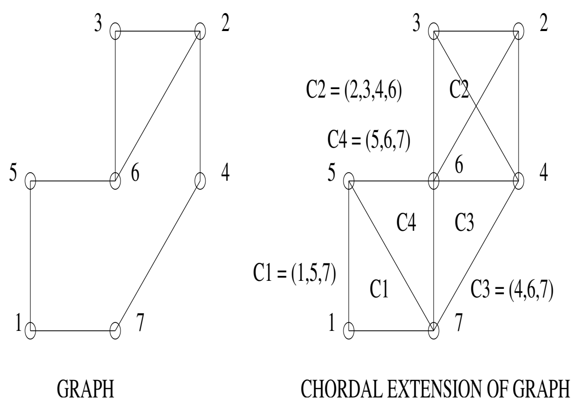

Figure 1: Sample graph and its chordal extension

diagonal entries are all nonzero. Moreover, C has a nonzero entry in the (i, j)th off-diagonal position if $\{i, j\} \in E$. The ith constraint coefficient matrix A_i has a nonzero entry only in the (i, i)th position. We define an *aggregate sparsity pattern* matrix $A(E) \in \mathcal{S}^n$ that has a nonzero entry in the (i, j)th position if any of the data matrices has a nonzero entry in the (i, j)th position. For our example, we have

$$A(E) = \begin{pmatrix} x & & & & x & & x \\ & x & x & x & & x & \\ & x & x & & & x & \\ & x & & x & & & x \\ x & & & & x & x & \\ & x & x & & x & x & \\ x & & & x & & & x \end{pmatrix}.$$

Note that both $A(E)$ and C have the same sparsity pattern that can be represented by the graph on the left hand side of Figure 1. Moreover, only the elements of X corresponding to nonzero entries in $A(E)$ participate in the objective function and the equality constraints of (12). The remaining entries of X are present only in the constraint requiring X to be positive semidefinite.

A graph is *chordal* if every cycle of length ≥ 4 has a chord, i.e., an edge joining non-consecutive vertices in the graph. We note that $G = (V, E)$ is not chordal since the cycle $(5, 6, 2, 4, 7, 1, 5)$ of length 6 does not have any chord. A chordal extension $G' = (V, E')$ of $G = (V, E)$ can be computed via a symbolic factorization of the aggregate sparsity matrix $A(E)$. The new edges added correspond to the fill-in introduced during this procedure. For our example, the symbolic factorization of $A(E)$ yields fill-in at $(i, j) = (5, 7), (3, 4), (4, 6)$, and $(6, 7)$. This gives us the chordal graph $G' = (V, E')$ shown on the right hand side of Figure 1, where $E' = E \cup \{\{5, 7\}, \{3, 4\}, \{4, 6\}, \{6, 7\}\}$. Computing the minimum chordal extension graph is NP hard. However, there are several heuristics such as minimum degree ordering (see Fukuda et al. [11]) that are used to reorder the rows and columns of $A(E)$ before carrying out the symbolic factorization. These heuristics can conceivably be used to find a minimal chordal extension $G' = (V, E')$ quickly. Let $A(E')$ denote the sparsity pattern of the symbolic factorization of $A(E)$.

A clique in a graph is a set of vertices all of which are connected together by edges. A maximal clique is a clique that is not contained in a larger clique. A vertex v is said to be *simplical* if its adjacent vertices $\text{adj}(v)$ form a clique. Consider an ordering of the vertices in the graph, where the vertices are ordered as $v_1, \ldots, v_n$. An ordering is said to be a *perfect elimination ordering* (PEO) if for $1 \leq i \leq n$, the vertex v_i is simplical in the graph consisting of vertices $v_i, v_{i+1}, \ldots, v_n$ and their incident edges. A PEO is obtained from the ordering that was employed before carrying out a symbolic factorization. For our example, the ordering $\{1, 2, \ldots, 6, 7\}$ is a perfect elimination ordering for the chordal graph on the right hand side of Figure 1. Given a PEO, one can find the maximal cliques in the graph; these maximal cliques are the maximal members of $\{v_i\} \cup (\text{adj}(v_i) \cap \{v_{i+1}, \ldots, v_n\})$, $i = 1, \ldots, n$ (see Blair & Peyton [3]). For our example, we obtain the four maximal cliques $Cl_1 = \{1, 5, 7\}$, $Cl_2 = \{2, 3, 4, 6\}$, $Cl_3 = \{4, 6, 7\}$, and $Cl_4 = \{5, 6, 7\}$ with respect to the PEO $\{1, 2, \ldots, n\}$.

Given a matrix $X \in \mathcal{S}^n$ whose entries in $A(E')$ (chordal sparsity pattern) are specified, the *positive semidefinite matrix completion* result (see Grone et al. [13]) shows that X is positive semidefinite if and only if it is also *partially positive semidefinite*, i.e, all its submatrices corresponding to maximal cliques in $G' = (V, E')$ are also positive semidefinite. This allows one to replace the constraint $X \succeq 0$ with an equivalent set of constraints $X_{Cl_i, C_i} \succeq 0$, $i = 1, \ldots, r$, where r is the number of maximal cliques in the chordal extension graph $G' = (V, E')$. The number of maximal cliques r is bounded above by the number of nodes in the graph, i.e., $r \leq n$.

Using these ideas, we rewrite (12) as the equivalent SDP

$$\begin{array}{lrcl} \max & \displaystyle\sum_{k=1}^{4}(C^k \bullet X^k) & & \\ \text{s.t.} & X^1_{23} - X^4_{13} & = & 0, \\ & X^2_{34} - X^3_{12} & = & 0, \\ & X^3_{23} - X^4_{23} & = & 0, \\ & X^k_{ii} & = & 1, \ \ i = 1, \ldots, |Cl_k|, \ \ k = 1, \ldots, 4, \\ & X^k & \succeq & 0, \ \ k = 1, \ldots, 4 \end{array} \tag{13}$$

where $|Cl_k|$ denotes the cardinality of Cl_k and $X^k \in \mathcal{S}^{|Cl_k|}$ is the submatrix obtained from the rows and columns of X corresponding to elements in Cl_k. For instance, $X^1 = \begin{pmatrix} X_{11} & X_{15} & X_{17} \\ X_{15} & X_{55} & X_{57} \\ X_{17} & X_{57} & X_{77} \end{pmatrix}$. Some of the cliques share elements; for instance Cl_1 and Cl_4 share nodes 5 and 7 in the graph and these common elements account for the constraints $X^1_{23} - X^4_{13} = 0$, etc. We use $X^1_{22} = X^2_{11} = X_{55}$ and $X_{55} = 1$ to rewrite the constraint $X^1_{22} - X^2_{11} = 0$ as two independent block constraints $X^1_{22} = 1$ and $X^2_{11} = 1$. The coefficient matrices C^k, $k = 1, \ldots, 4$ are constructed from the original Laplacian matrix C. The node 5 appears in the maximal cliques Cl_1 and Cl_4 ; it is the 2nd node in Cl_1 and the 1st node in Cl_4. Therefore, we have $C^1_{22} = C^4_{11} = \dfrac{C_{55}}{2}$. The other entries can be constructed similarly. Now, (13) is an SDP in the block angular form (1), where the sets

$$\mathcal{C}_k \ = \ \{X^k \in \mathcal{S}^{|Cl_k|} : X^k_{ii} = 1, \ \ i = 1, \ldots, |Cl_k|, \ \ X^k \succeq 0\}.$$

The constraints $X^1_{23} - X^4_{13} = 0$ represent the coupling constraints in (1). Note that each of these constraints couple only two of the subsystems.

The graph $G'' = (V, E'')$ obtained from the graph $G' = (V, E')$ by dropping the redundant edge $\{3, 4\}$ is also chordal with 5 maximal cliques $Cl'_1 = \{1, 5, 7\}$, $Cl'_2 = \{2, 3, 6\}$, $Cl'_3 = \{4, 6, 7\}$, $Cl'_4 = \{5, 6, 7\}$, and $Cl'_5 = \{2, 4, 6\}$. This allows one to rewrite the original SDP as another block-angular SDP where the size of the largest block is 3. On the other hand, the new SDP has an additional block as well as an additional coupling constraint. In general, one attempts to preprocess an SDP into a block-angular SDP with small sized blocks while simultaneously trying to keep the number of coupling constraints small.

The special structure of the maxcut SDP allows us to write the preprocessed problem in block-angular form; in many cases, the resulting SDP can be put only in the block-diagonal form. However, any SDP of the form

$$\begin{array}{lrcl} \max & C \bullet X & & \\ \text{s.t.} & X_{ij} & = & B_{ij}, \ \ (i,j) \in I, \\ & X_{pq} & \leq & F_{pq}, \ \ (p,q) \in J, \\ & A_k \bullet X & = & b_k, \ \ k = 1, \ldots, m_{\text{org}}, \\ & X & \succeq & 0 \end{array} \tag{14}$$

where I and J are disjoint index sets that together include the diagonal entries $D = \{(1,1),\ldots,(n,n)\}$ can be preprocessed via the matrix completion procedure into an equivalent block-angular problem as we illustrate in Algorithm 1 below. We consider the following examples in this paper:

1. The SDP relaxation (12) of the maxcut problem is a special case of (14), where $m_{\text{org}} = 0$, $I = D$, $B = I$ is the identity matrix of size n, and $J = \emptyset$.

2. Let $e_{ij(n+1)} \in I\!R^{n+1}$ denote the vector with ones in the ith, jth, and $(n+1)$th positions respectively, and zeros elsewhere. Let $A_{ij(n+1)} = e_{ij(n+1)} e_{ij(n+1)}^T$. Consider the Lovasz theta problem, which is the SDP relaxation

$$\begin{array}{lrcl} \max & C \bullet X & & \\ \text{s.t.} & X_{ii} & = & 1, \;\; i = 1,\ldots,n+1 \\ & A_{ij(n+1)} & = & 1, \;\; (i,j) \in E \\ & X & \succeq & 0, \end{array} \tag{15}$$

   of the stable set problem (see Benson et al. [1]), where

$$C \;=\; \begin{bmatrix} \frac{1}{2} & \cdots & & \frac{1}{4} \\ & \ddots & & \vdots \\ & & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \cdots & \frac{1}{4} & 0 \end{bmatrix} \;\in\; \mathcal{S}^{n+1}.$$

   This SDP is of the form (14), where $m_{\text{org}} = |E|$, $I = D$, $B = I$, and $J = \emptyset$.

3. Consider the SDP relaxation

$$\begin{array}{lrcl} \max & C \bullet X & & \\ \text{s.t.} & X_{ii} & \leq & 1, \;\; i = 1,\ldots,n \\ & X & \succeq & 0 \end{array} \tag{16}$$

   of box constrained quadratic programs (see Benson et al. [1]), which is of the form (14), where $m_{\text{org}} = 0$, $I = \emptyset$, $J = D$, and F is the identity matrix of size n.

The complete algorithm is summarized below:

Algorithm 1 (Processing SDP of type (14) into block-angular form)

1. *Construct aggregate sparsity pattern matrix $A(E) \in \mathcal{S}^n$ from data matrices in (14).*

2. *Reorder the rows and columns of $A(E)$ so as to minimize the fill-in in the symbolic factorization. We discuss the choice of heuristics in Section 5.*

3. *Perform a symbolic factorization of $A(E)$ with the ordering computed in Step 2. Let $A(E') \in \mathcal{S}^n$ represent the sparsity pattern of the symbolic factorization. Construct the chordal extension graph $G' = (V, E')$ from $A(E')$.*

4. *Find maximal cliques Cl_i, $i = 1, \ldots, r$ in the chordal graph $G' = (V, E')$.*

5. *Process the cliques Cl_i, $i = 1, \ldots, r$ using Algorithms 3.2 and 4.1 in Fujisawa et al. [10], achieving a suitable compromise between the reduction in the size of largest clique and keeping the number of cliques small.*

6. *Construct a block-angular SDP from clique information as follows:*

   (a) *For each nonzero element $(i, j) \in A(E')$, let $\hat{r}(i, j) = \min\{r : (i, j) \in Cl_r \times Cl_r\}$ and let i and j be elements p and q in $Cl_{\hat{r}(i,j)}$, respectively. Set $C_{pq}^{\hat{r}(i,j)} = C_{ij}$ and $[A_k]_{pq}^{\hat{r}(i,j)} = [A_k]_{ij}$, $k = 1, \ldots, m_{\text{org}}$.*

   (b) *Initialize the element count $EC_{ij} = 1$, $i, j = 1, \ldots, n$. Repeat Step 6(b) for all pairs of cliques: Suppose cliques Cl_k and Cl_l share a common element (i, j) (common node i if $i = j$ or common edge (i, j) if $i \neq j$). Further, assume that i and j are elements p and q in Cl_k; and elements s and t in Cl_l, respectively. Proceed as follows:*

      - *If $(i, j) \in I$, introduce independent constraints $X_{pq}^k = B_{ij}$ and $X_{st}^l = B_{ij}$ in block-angular SDP. Update $EC_{ij} = EC_{ij} + 1$.*
      - *If $(i, j) \in J$, add coupling constraint $X_{pq}^k - X_{st}^l = 0$ and independent constraints $X_{pq}^k \leq E_{ij}$ and $X_{st}^l \leq E_{ij}$ to block-angular SDP. Update $EC_{ij} = EC_{ij} + 1$.*
      - *If $(i, j) \notin I \cup J$, add coupling constraint $X_{pq}^k - X_{st}^l = 0$ to block-angular SDP.*

   (c) *Construct coefficient matrices for the blocks as follows: The (p, q)th entry for the coefficient matrix in kth block is given by $C_{pq}^k = \dfrac{C_{ij}}{EC_{ij}}$, $p, q = 1, \ldots, |Cl_k|$, where the elements p and q in Cl_k are nodes i and j in $G' = (V, E')$.*

We omit the details of Step 5 and instead refer the reader to Fujisawa et al. [10]. Moreover, this step requires a number of parameters and we indicate our choice for these parameters in Table 1 in Section 5.2.

4 Details of our decomposition algorithm

We present the details of the decomposition algorithm that we outlined in Section 2.

4.1 Regularized decomposition algorithm for block-angular SDPs

Algorithm 2 (Regularized decomposition algorithm)

1. **Initialize:** *Set $k = 1$, $J^0(i) = \emptyset$, $z_i^0 = -\infty$, $i = 1, \ldots, r$. Choose $m_L \in (0, 1)$, any starting point $y^0 \in I\!R^m$, and set $x^0 = y^0$. Parameter values are given in Section 5.1.*

2. **Solve the subproblems in parallel:** *For $i = 1, \ldots, r$, solve the ith subproblem*

$$\theta_i(y^k) \quad = \quad \max_{X_i \in \mathcal{C}_i} ((C_i - \mathcal{A}_i^T y^k) \bullet X_i)$$

   for X_i^k. Details are given in Section 4.2.

3. **Update master problem:** *For $i = 1, \ldots, r$ do: If $\theta_i(y^k) > z_i^k$, add constraint*

$$\mathcal{A}_i(X_i^k)^T y + z_i \quad \geq \quad C_i \bullet X_i^k$$

   to master problem (9) and set $J^k(i) = J^{k-1}(i) \cup \{k\}$. Else, $J^k(i) = J^{k-1}(i)$.

4. **Update the center x^k:** *If $k = 1$ or if*

$$\theta(y^k) \quad \leq \quad (1 - m_L)\theta(x^{k-1}) + m_L \theta^{k-1}(y^k) \tag{17}$$

   then set $x^k = y^k$ (serious step). Else, set $x^k = x^{k-1}$ (null step).

5. **Update the weight u^k:** *Use procedure discussed in Section 4.4 to adjust weights.*

6. **Solve the master problem:** *Solve master problem (9) for y^{k+1} and z^{k+1} and dual multipliers λ^k. Let $\theta^k(y^{k+1}) = b^T y^{k+1} + \sum_{i=1}^{r} z_i^{k+1}$. Details are given in Section 4.3.*

7. **Termination Check:** *If (11) is satisfied, stop. Else, continue to Step 8.*

8. **Aggregate unimportant block constraints:** *Drop all constraints in (9) whose dual multipliers $\lambda_i^k \leq 10^{-6}$. Use procedure in Section 4.5 to aggregate constraints. Set $k = k + 1$ and return to Step 2.*

4.2 Solving subproblems

Each subproblem is solved using the serial version of CSDP 6.0 [4] on a processor. Note that only the coefficient matrix in (4) changes in every iteration, so the previous primal solution to the subproblem is still feasible in the new problem. On the other hand, the previous solution to the dual subproblem is no longer feasible. We were not successful in our attempts to warm-start the subproblems with strictly feasible primal-dual starting points (using previous information) at each iteration. In our computational experiments in Section 5.2, we optimize the subproblem (4) using CSDP 6.0 from scratch in every iteration. Developing suitable strategies to warm-start subproblems is a worthy topic for future investigation.

4.3 Solving the master problem

The master problem (9) is solved serially on the root processor using the barrier optimizer in CPLEX 9.0 [8]. The dual to (9) (see Kiwiel [19]) can also be written as

$$\begin{array}{lrcl} \max & -\frac{1}{2u^k}\|\sum_{i=1}^{r}\sum_{j\in J^k(i)} \mathcal{A}_i(X_i^j)\lambda_i^j - b\|^2 - (x^k)^T\left(\sum_{i=1}^{r}\sum_{j\in J^k(i)} \mathcal{A}_i(X_i^j)\lambda_i^j\right) + \sum_{i=1}^{r}\sum_{j\in J^k(i)} (C_i \bullet X_i^j)\lambda_i^j & & \\ \text{s.t.} & \sum_{j\in J^k(i)} \lambda_i^j & = & 1,\ i=1,\ldots,r \\ & \lambda_i^j & \geq & 0,\ i=1,\ldots,r,\ j\in J^k(i). \end{array} \tag{18}$$

One can either a) solve the separable QP (9) for y^{k+1} and z^{k+1} and obtain λ^{k+1} as the dual multipliers in the solution, b) solve (18) for λ^{k+1} and dual multipliers z^{k+1}, and then obtain y^{k+1} using

$$y^{k+1} = x^k + \frac{1}{u^k}\left(\sum_{i=1}^{r}\mathcal{A}_i\left(\sum_{j\in J^k(i)} \lambda_i^j(X_i^j)\right) - b\right). \tag{19}$$

Although, the second approach requires additional work to set up (18), an advantage is that y variables have been eliminated in (18). We tried both approaches and found no comparative advantages. We use the first approach to solve the master problem in Step 6 of Algorithm 2.

A crucial feature in the solution of the master problem is the sparsity of the coupling constraints. We note in Section 3 that the coupling constraints couple only two of the r blocks. The QP barrier optimizer in CPLEX exploits this feature in solving the master problem at each iteration.

4.4 Adjusting the weight parameter

The convergence of the algorithm can be improved by dynamically changing the weights u during the course of the algorithm. The underlying idea is to either decrease the weight u or keep it unchanged during a serious step; and increase the weight or keep it unchanged during a null step.

Kiwiel (see Procedure 2.2 in [19]) presents a procedure to dynamically update the weight during the course of the algorithm and suggests a starting value $u^0 = \|g(y^0)\|$. Helmberg & Kiwiel (see Section 4.2 in [15]) describe an extension to this procedure (called level modification), that includes an additional opportunity to decrease the weight during a serious step. Moreover, Helmberg [14] suggests other starting weights in his computational experiments with SBmethod including $u^0 = \frac{\|g(y^0)\|}{\sqrt{m}}$, where m is the number of coupling constraints.

In our computational experiments, we found that $u^0 = \|g(y^0)\|$ is too large on some of the larger instances, and leads to premature termination of the algorithm. On the other hand, the choice $u^0 = \frac{||g(y^0)||}{\sqrt{m}}$ leads to very slow convergence. This is because Algorithm 1 in Section 3 considerably increases the number of constraints in the block-angular SDP.

We adopt the following rule for adjusting the weight parameter in our computational experiments in Section 5: Our starting weight is $u^0 = \frac{||g(y^0)||}{l}$ where $l > 1$ is a suitable parameter. We include the choices for l for our test instances in Tables 2 and 3. Moreover, we vary the weight during the course of the algorithm according to the procedure described in Section 4.2 of Helmberg & Kiwiel [15].

4.5 Aggregating less important constraints in master problem

An additional feature in our regularized decomposition scheme is that we aggregate less important constraints in (9) to reduce the size of the master problem. The notion of aggregation was introduced in Kiwiel [18]. The underlying idea in the aggregation scheme is to take a convex combination of the constraints (whose dual multipliers λ are small) in an aggregate constraint without affecting the solution of the master problem.

Let $c_{\max}$ be the maximum number of constraints allowed in the quadratic master problem (9). In every iteration, we sort the constraints in (9) based on their λ values and aggregate the constraints whose multipliers λ_i^k are small in r blocks. Let $J_{agg}^k(i)$, $i = 1, \ldots, r$ denote the set of these constraints for the various blocks in the kth iteration.

In the aggregation step in the kth iteration, we replace (9) with the following quadratic program

$$\begin{array}{lrcl} \min & b^T y + \sum_{i=1}^{r} z_i + \frac{u^k}{2}||y - x^k||^2 & & \\ \text{s.t.} & \mathcal{A}_i(X_i^j)^T y + z_i & \geq & C_i \bullet X_i^j, \ \ i = 1, \ldots, r, \ \ j \in (J^k(i) \cap J_{\text{agg}}^k(i)), \\ & A_i^{\text{agg}T} y + z_i & \geq & C_i^{\text{agg}}, \ \ i = 1, \ldots, r. \end{array}$$

where $A_i^{\text{agg}} \in I\!R^m$ and $C_i^{\text{agg}} \in I\!R$ are defined as

$$\begin{array}{rcl} A_i^{\text{agg}} & = & \dfrac{\lambda_{\text{agg}}^i A_i^{\text{agg}} + \sum\limits_{j \in J_{\text{agg}}^k(i)} \lambda_i^j \mathcal{A}_i(X_i^j)}{\lambda_{\text{agg}}^i + \sum\limits_{j \in J_{\text{agg}}^k(i)} \lambda_i^j} \\ C_i^{\text{agg}} & = & \dfrac{\lambda_{\text{agg}}^i C_i^{\text{agg}} + \sum\limits_{j \in J_{\text{agg}}^k(i)} \lambda_i^j (C_i^j \bullet X_i^j)}{\lambda_{\text{agg}}^i + \sum\limits_{j \in J_{\text{agg}}^k(i)} \lambda_i^j} \end{array}$$

where λ_{agg}^i is the dual multiplier corresponding to the ith aggregated constraint $A_i^{\text{agg}T} y + z_i \geq C_i^{\text{agg}}$ in the kth iteration. We initialize the procedure by setting the aggregate constraint for each block to be the 1st constraint that is added in this block. Moreover, the aggregate block constraints are never dropped during the course of the algorithm.

4.6 Computational cost and memory requirements for Algorithm 2

We evaluate the computational cost and memory requirements of the important steps of Algorithm 2.

1. **Step 2:** The data matrices for the subproblems are normally sparse. The memory storage for the ith subproblem is about $O(m_i^2 + n_i^2)$ bytes. Computation of $S_i = (C_i - A_i^T y)$ requires $O(mn_i^2)$ flops. The storage requirement for S_i is of the same order as the input. The dominant task in each iteration of the IPM is a) forming the Schur complement matrix that requires $O(m_i^2 n_i^2 + m_i n_i^3)$ flops (dense case) and $O(m_i n_i^2)$ flops (sparse case), and b) factorizing the Schur complement matrix via a Cholesky decomposition that requires $O(m_i^3)$ flops. An upper bound on the number of IPM iterations is 100. On average, we require about 15 iterations.

2. **Step 3:** The master problem (9) is updated in parallel in $O(mn_i^2)$ flops.

3. **Step 4:** Each subproblem communicates its current objective of $\theta_i(y^k)$ (from Step 2) to the root processor that uses this information to compute $\theta(y^k)$. This requires $O(m + r)$ flops, where m is the number of coupling constraints and r is the number of blocks. The value $\theta^{k-1}(y^k)$ is the objective value of the master problem from the previous iteration (see Step 6).

4. **Step 5:** The weight update requires the calculation of the subgradient $g(y)$ vector from (6). Each processor communicates a vector of size m to the root processor that assembles $g(y)$ and carries out the weight update in $O(mr)$ flops.

5. **Step 6:** The master problem is a separable QP in dual form with $(m + r)$ variables and $c_{\max}$ constraints. It is solved using the barrier optimizer in CPLEX 9.0. An upper bound on the number of iterations is 100 and on average we require about 10 iterations.

6. **Step 8:** The aggregation phase requires sorting $c_{\max}$ constraints and aggregating r constraints on average. This is done in $O(c_{\max} \log(c_{\max}) + mr)$ flops.

4.7 Convergence of the algorithm

The convergence of the regularized decomposition algorithm is analyzed in Kiwiel [19] and Ruszczysńki [28] and we can prove the convergence of Algorithm 2 as in [28]. We summarize the essential arguments below.

The algorithm generates iterates x^k such that the sequence $\{\theta(x^k)\}$ is nonincreasing.

To see this, note that

$$\begin{aligned} \theta^{k-1}(y^k) &\le \theta^{k-1}(y^k) + \frac{u^{k-1}}{2}||y^k - x^{k-1}||^2 \\ &\le \theta^{k-1}(x^{k-1}) \text{ (since } y^k \text{ minimizes (8))} \\ &\le \theta(x^{k-1}) \text{ (since } \theta^{k-1}(y) \text{ is an underestimate of } \theta(y)). \end{aligned} \tag{20}$$

Now, suppose that we have a serious step in the kth iteration of Algorithm 2. In Step 4, we have

$$\begin{aligned} \theta(x^k) &\le (1-m_L)\theta(x^{k-1}) + m_L\theta^{k-1}(y^k) \\ &\le \theta(x^{k-1}) \text{ (from (20))}. \end{aligned}$$

Moreover, the sequence $\{\theta(x^k)\}$ is bounded from below by $\theta(y^*)$. If the algorithm terminates in Step 7 of Algorithm 2, we have $\theta^k(y^{k+1}) = \theta(x^k)$, that gives $y^{k+1} = x^k$ from (20). The optimality condition for the master problem (8) is

$$\begin{aligned} -u^k(y^{k+1} - x^k) &\in \partial\theta^k(y^{k+1}) \text{ i.e.} \\ 0 &\in \partial\theta^k(y^{k+1}) \text{ (since } y^{k+1} = x^k) \text{ i.e.} \\ 0 &\in \partial\theta(x^k) \text{ (since } \theta^k(y^{k+1}) = \theta(x^k)) \end{aligned}$$

that shows that x^k is optimal in (3).

Suppose the algorithm does not terminate in Step 7. Then the algorithm performs an infinite number of iterations. Since the sets $\mathcal{C}_i$ are bounded, we have $||g(y^k)|| \le C$, for all k where $C > 0$ is a constant. The following lemma from Ruszczysński [28] (Lemma 7.15 on page 375) shows that the number of null steps is finite and is always followed by a serious step.

Lemma 1 *Suppose x^k is not an optimal solution to the dual problem (3), then there exists an $m > k$ such that $\theta(x^m) < \theta(x^k)$.*

This shows that the algorithm cannot get stuck at a suboptimal point. Moreover, it indicates that if the algorithm does not terminate in Step 7, then it performs an infinite number of serious steps. Using these arguments, Ruszczyński [28] (Theorem 7.16 on page 377) shows the convergence of the decomposition algorithm. We summarize the final result below:

Theorem 1 *Let y^* be an optimal solution to the dual problem (3). Then, the sequence of centers $\{x^k\}$ generated by the decomposition algorithm converges to y^*. Moreover, we have*

$$\begin{aligned} \lim_{k\to\infty} y^{k+1} &= y^*, \\ \lim_{k\to\infty} \theta^k(y^{k+1}) &= \theta(y^*), \\ \lim_{k\to\infty} \theta^k(y^{k+1}) + \frac{u^k}{2}||y^{k+1} - x^k||^2 &= \theta(y^*). \end{aligned} \tag{21}$$

The first equation in (21) shows that the sequence $\{y^{k+1}\}$ of solutions to the regularized master problem (9) also converges to y^*. The 2nd and 3rd equations in (21) show that the sequence of lower bounds $\{\theta^k(y^{k+1})\}$ and objective values to the regularized master problem also converge to the optimal objective value of (3), respectively.

5 Computational experience with the algorithm

Section 5.1 discusses our MPI implementation and default parameter values. Section 5.2 presents detailed computational results with the algorithm. Section 5.3 discusses rate of convergence of decomposition algorithm. Sections 5.4 and 5.5 compare the solution times and memory requirements of our algorithm with the serial and OpenMP versions of CSDP, respectively. Section 5.6 discusses scalability of our parallel implementation.

5.1 Current implementation and default parameter values

We first discuss our implementation of Algorithm 1 to preprocess an existing SDP into an equivalent problem in the block-angular form. We borrowed a C++ implementation of the matrix completion code (Algorithm 3.1 in Fujisawa et al. [10]) by Mituhiro Fukuda to implement Steps 1-5 of Algorithm 1. Step 2 of this code calls heuristics such as the multilevel nested dissection algorithm in METIS-4.0.1 [22]; and the minimum degree, generalized nested dissection, multisection algorithms in Spooles-2.2 [32] to determine suitable orderings that minimize the fill-in in $A(E)$. Step 3 uses the ordering with the minimum fill-in from Step 2 to construct $G' = (V, E')$. Step 5 uses Algorithms 3.2 or 4.1 in Fujisawa et al. [10] to suitably process the maximal cliques (see Fujisawa et al. [10]). We indicate the choice of the algorithm along with the parameter values for our test instances in Table 1. We added Step 6 of Algorithm 1 to the code, so that it returns a block-angular SDP in SDPA format.

We have developed our parallel decomposition algorithm (Algorithm 2) in the C programming language that uses the Message Passing Interface (MPI) (see Pacheco [27]) for communication between processors. We implemented this algorithm on the distributed memory IBM Blade Center Linux Cluster (Henry2) [1] at North Carolina State University. There are 175 2.8GHz-3.2GHz nodes in the Henry2 cluster. Each node has two processors and 4GB of memory (2GB of memory per processor).

Our code takes a block-angular SDP that is stored in the SDPA format as input. Let p be the number of processors used in our computational experiments. Each processor is assigned a minimum of $\lfloor \frac{r}{p} \rfloor$ and a maximum of $\lceil \frac{r}{p} \rceil$ blocks. The assignment of blocks to processors is done with a view to balance the load among the various processors. It is done as follows: First, we sort the blocks based on their sizes in descending order. The first $p \times \lfloor \frac{r}{p} \rfloor$ elements of the sorted vector are rearranged in a $p \times \lfloor \frac{r}{p} \rfloor$ matrix, where the first p elements of the vector form the first column of the matrix and so on. Each element of an odd (even) column of this matrix is assigned to a processor from top to bottom (bottom to top). If $\lfloor \frac{r}{p} \rfloor$ is even (odd), the remaining $r - p \times \lfloor \frac{r}{p} \rfloor$ elements are assigned to the first (last) $r - p \times \lfloor \frac{r}{p} \rfloor$ processors. Each processor stores the data $\mathcal{A}_i$, C_i, and $\mathcal{B}_i$ for the assigned

[1] http://www.ncsu.edu/itd/hpc/Hardware/Hardware.php

blocks. The root processor (processor 0) also stores the right hand side vector b for the coupling constraints.

The master problem is solved in serial fashion on the root processor (processor 0). We used CPLEX 9.0 [8] to solve the QP master problem. Each processor including the root processor uses the serial version of CSDP 6.0 [4] to solve all the subproblems (smaller SDPs) that it has been assigned. After solving all these subproblems, each processor uses a flag (set to 1) to indicate to the root processor that it has columns to add to the master problem. If this is the case, the processor also collectively returns a number of columns that the root processor includes in the master problem. On the other hand, the algorithm terminates if none of the processors (including the root processor) adds a column to the master problem. In our current implementation, the root processor does not begin the solution of the master problem (in each iteration) until it has received the proposals from all the subproblems.

We have used appropriate MPI communication modes between the various processors during the course of the algorithm. In each iteration of algorithm, the root processor uses the `MPI_Bcast` collective communication mode to broadcast the dual multipliers y to all the other processors. All other communication between the processors is carried out using the basic `MPI_Send` and `MPI_Recv` commands, i.e., a processor executing a Send (Receive) will wait until the receiving processor executes a matching Receive (Send) operation (synchronous mode; see Pacheco [27]).

We also briefly mention the values of the parameters that are employed in our computational experiments: Our starting point y^0 is the vector of all zeros. Our default accuracy parameter $\epsilon = 10^{-2}$. We chose the parameter $m_L = 0.1$ in Algorithm 2 (as in Procedure 2.2 in Kiwiel [19]). Our starting weight was $u^0 = \frac{||g(y^0)||}{l}$, where $||g(y^0)||$ is the Euclidean norm of the subgradient vector computed at $y = y^0$, and $l > 1$ is an appropriate parameter. We include the choices for l for our test instances in Tables 2 and 3. We updated the weights during the course of the algorithm as in Section 4.2 of Helmberg & Kiwiel [15] (level modification). Some of our important parameters in this weight updating scheme are $m_R = 0.5$ and $u_{\min} = 10^{-10} \times u^0$.

We solve our QP master problem using the barrier algorithm in CPLEX 9.0 [8]; and our subproblems using CSDP 6.0 [4] to a default accuracy of 10^{-6} in every iteration. Let r denote the number of blocks in the preprocessed SDP. We set the maximum number of constraints in the quadratic master problem (9) to $c_{\max} = 10r$ for the first 6 instances and $c_{\max} = 5r$ for the last 19 instances in Table 1.

5.2 Detailed computational results

We have tested our algorithm on SDP relaxations of maxcut, stable set, and box constrained quadratic problems. Our 25 test instances are shown in Table 1.

A word on terminology: For all instances in Table 1, we use original SDP to denote the regular SDP relaxation for the maxcut, stable set, or box constrained quadratic problems that is mentioned in Section 3. We use preprocessed SDP to denote the block-angular SDP obtained by applying Algorithm 1 on the original SDP. The sizes of the original and preprocessed SDPs for the instances are mentioned in Table 1. In our computational experiments, we test CSDP 6.0 on both the original and the preprocessed SDPs in Table 1, while our decomposition algorithm (Algorithm 2) is applied on the preprocessed SDP.

The small SDPs (original SDPs for first 12 instances in Table 1) are taken from SDPLIB [5]. The large SDPs (original SDPs for last 13 instances in Table 1) are generated from the graphs in the Gset library [2]. The graphs chosen from Gset are the sparsest ones for a given number of vertices. We choose these graphs, since Algorithm 1 is able to generate preprocessed SDPs with the greatest reduction in the size of the largest semidefinite cone (for `thetaG77` in Table 1, the size of largest semidefinite cone in the preprocessed SDP has been reduced by a factor of 18). The columns in Table 1 represent the following:

1. Prob: Name of instance.
2. n: Size of the matrix in the original SDP.
3. m: Number of equality constraints in the original SDP.
4. n_p: Size of the largest semidefinite cone in the preprocessed SDP (Number of cones in the preprocessed SDP).
5. m_p: Number of equality constraints in the preprocessed SDP (Number of coupling constraints in the preprocessed SDP).
6. n_{total}: Size of overall block diagonal matrix in the preprocessed SDP.
7. Alg: Algorithm 3.2 or 4.1 from Fujisawa et al. [10] used in processing the maximal cliques in Step 5 of Algorithm 1.
8. ζ and η: Parameters in Algorithm 3.2 (see [10]).
9. α, β, and γ: Parameters in Algorithm 4.1 (see [10]).

A short discussion on the choice of the parameters in Algorithms 3.2 and 4.1 employed in Step 5 of Algorithm 1 to obtain the preprocessed SDPs is in order: Fujisawa et al. [10] use a statistical method based on the analysis of variance (ANOVA) to estimate the parameters α, β, γ, ζ, and η in Algorithms 3.2 and 4.1 (see Table 1 in [10]). Their analysis is based on the flop count of a typical primal-dual IPM in solving a given SDP. We continue to use their estimate for the various parameters, except that we vary ζ (default value of 0.095 in [10]) to a maximum value of 0.3 in our computational results. A larger value of ζ reduces

[2]http://www.stanford.edu/∼yyye/yyye/Gset/

Prob	n	m	n_p	m_p	n_{total}	Alg	ζ	η	α	β	γ
maxG11_1	800	800	216 (4)	1208 (360)	848	4.1	-	-	0.7	20	46
maxG11_2	800	800	122 (85)	3932 (2469)	1096	3.2	0.3	1.075	-	-	-
qpG11_1	1600	800	432 (4)	1256 (408)	1696	4.1	-	-	0.7	20	46
qpG11_2	1600	800	180 (61)	3808 (2528)	2560	3.2	0.2	1.075	-	-	-
maxG32_1	2000	2000	528 (8)	5310 (3138)	2172	4.1	-	-	0.7	20	46
maxG32_2	2000	2000	286 (22)	8690 (6318)	2500	3.2	0.15	1.075	-	-	-
maxG32_3	2000	2000	251 (66)	13550 (10729)	2821	3.2	0.2	1.075	-	-	-
maxG32_4	2000	2000	197 (97)	17036 (13867)	3169	3.2	0.3	1.075	-	-	-
thetaG11	801	2401	194 (23)	17036 (13867)	1019	3.2	0.2	1.075	-	-	-
maxG48_1	3000	3000	542 (64)	24580 (20650)	3930	3.2	0.2	1.075	-	-	-
maxG48_2	3000	3000	445 (157)	35151 (30280)	4871	3.2	0.3	1.075	-	-	-
qpG32	4000	2000	432 (97)	16402 (13313)	6178	3.2	0.2	1.075	-	-	-
maxG57	5000	5000	405 (516)	61595 (51883)	9712	3.2	0.3	1.075	-	-	-
thetaG32	2001	6001	212 (80)	22774 (16762)	3187	3.2	0.3	1.075	-	-	-
thetaG48	3001	9001	441 (102)	39468 (30632)	4586	3.2	0.3	1.075	-	-	-
thetaG57	5001	15001	434 (182)	75369 (60711)	9313	3.2	0.3	1.075	-	-	-
maxG62	7000	7000	515 (692)	100873 (87232)	13641	3.2	0.3	1.075	-	-	-
maxG65	8000	8000	656 (791)	101300 (85961)	15339	3.2	0.3	1.075	-	-	-
maxG66	9000	9000	633 (898)	134904 (117344)	17560	3.2	0.3	1.075	-	-	-
maxG67	10000	10000	903 (942)	131593 (112532)	19061	3.2	0.3	1.075	-	-	-
maxG77	14000	14000	916 (1395)	219118 (191591)	27527	3.2	0.3	1.075	-	-	-
thetaG62	7001	21001	506 (540)	118828 (98180)	13496	3.2	0.3	1.075	-	-	-
thetaG65	8001	24001	687 (555)	134199 (110726)	14974	3.2	0.3	1.075	-	-	-
thetaG67	10001	30001	853 (694)	201570 (172268)	19028	3.2	0.3	1.075	-	-	-
thetaG77	14001	42001	782 (933)	263318 (222229)	26315	3.2	0.3	1.075	-	-	-

Table 1: Our test instances

Prob	Opt val	p	l	UB	Time (h:m:s)	rel_acc	Serious steps	Null steps	Master (h:m:s)	Subproblem (h:m:s)
maxG11_1	629.16	4	3	629.54	40	6.0e-4	9	21	1	39
maxG11_2	629.16	4	3	629.48	58	5.0e-4	14	26	4	45
qpG11_1	2448.66	4	3	2450.81	1:55	8.7e-4	7	16	1	2:47
qpG11_2	2448.66	9	4	2450.36	1:06	6.9e-4	18	39	8	58
maxG32_1	1567.64	4	5	1568.47	26:37	5.3e-4	14	72	1:13	25:23
maxG32_2	1567.64	12	3	1568.47	10:00	5.3e-4	19	138	1:32	8:24
maxG32_3	1567.64	16	3	1568.43	16:53	5.0e-4	24	268	7:53	8:57
maxG32_4	1567.64	16	3	1568.49	18:42	5.4e-4	26	319	12:43	5:57
thetaG11	400.00	4	6	400.24	1:25	5.9e-4	14	14	2	1:23
maxG48_1	4500.00	16	3	4500.00	12:32	0.0	1	37	52	11:40
maxG48_2	4500.00	16	5	4500.00	14:04	0.0	1	58	3:22	10:37
qpG32	6226.55	16	5	6226.74	14:41	3e-5	24	100	2:06	12:33

Table 2: Computational results on small problems with default tolerance $\epsilon = 10^{-3}$

the size of the largest semidefinite cone but increases the number of blocks and constraints in the preprocessed SDP.

We describe our detailed computational results with the decomposition algorithm on the small instances in Table 2 and the large instances in Table 3, respectively. We solve the preprocessed SDPs for the small instances in Table 2 to an accuracy of 10^{-3} (see equation 11). The preprocessed SDPs for the large instances in Table 3 are solved to a default accuracy of $\epsilon = 10^{-2}$; this roughly translates to a relative accuracy (see equation (23)) of 5×10^{-3} for all these problems. The columns in Tables 2 and 3 represent the following:

1. Prob: Problem Name.

2. Opt val: Optimal objective value for SDP problem.

3. p: Number of processors used.

4. l: Factor used in determining the initial weight. The initial weight is

$$u^0 = \frac{||g(y^0)||}{l} \tag{22}$$

   where $g(y^0)$ is the subgradient computed at $y = y^0$.

5. UB: Objective value of best feasible solution obtained by decomposition algorithm.

6. Time (h:m:s): Total time taken by the decomposition algorithm to solve preprocessed SDP to a tolerance $\epsilon = 10^{-3}$ (Table 2) and $\epsilon = 10^{-2}$ (Table 3).

Prob	Opt val	p	l	UB	Time (h:m:s)	rel_acc	Serious steps	Null steps	Master (h:m:s)	Subproblem (h:m:s)
maxG57	3875.98	16	3	3903.62	17:31	7.1e-3	16	46	10:23	7:06
thetaG32	1000.00	16	2	1004.93	3:58	4.9e-3	19	48	1:21	2:36
thetaG48	1500.00	16	3	1505.49	25:17	3.6e-3	18	70	3:42	21:33
thetaG57	2500.00	16	5	2509.78	44:00	3.9e-3	21	83	15:35	28:22
maxG62	5410.91	16	3	5452.37	48:40	7.6e-3	17	57	31:58	16:32
maxG65	6185.04	16	3	6227.18	42:58	6.8e-3	16	41	17:26	25:24
maxG66	7097.21	16	3	7144.64	1:15:39	6.6e-3	17	54	45:28	30:01
maxG67	7708.93	8	3	7764.16	1:31:25	7.1e-3	16	42	28:32	1:01:30
maxG77	11097.67	16	3	11187.50	2:32:35	8.0e-3	18	52	1:18:34	1:13:41
thetaG62	3500.00	16	8	3511.57	1:46:40	3.3e-3	21	104	54:34	51:58
thetaG65	4000.00	16	8	4014.76	3:25:14	3.6e-3	20	95	48:35	2:36:31
thetaG67	5000.00	16	8	5024.48	6:19:05	4.8e-3	22	118	2:41:26	3:37:25
thetaG77	7000.00	16	10	7028.62	6:29:00	4.0e-3	22	119	3:42:30	2:46:05

Table 3: Computational results on large problems with default tolerance $\epsilon = 10^{-2}$

7. rel_acc: Relative accuracy given by

$$\text{rel\_acc} \quad = \quad \frac{\text{UB} - \text{Opt val}}{1 + |\text{Opt val}|}. \tag{23}$$

8. Serious steps: Number of serious steps performed by the algorithm.

9. Null steps: Number of null steps performed by the algorithm.

10. Master (h:m:s): Total time spent in solving the master problem.

11. Subproblem (h:m:s): Total time spent in solving the subproblems.

Tables 2 and 3 also give the number of null and serious steps performed by the algorithm in solving the preprocessed SDP. The sum of the serious and null steps gives the total number of iterations taken by the algorithm. The solution times mentioned in Tables 2 and 3 include the time taken by the decomposition algorithm to read the problem and solve it to default accuracy of $\epsilon = 10^{-3}$ and $\epsilon = 10^{-2}$, respectively. We also give the time spent in solving the master problem and subproblems. The time spent in solving the subproblems also includes the time spent in broadcasting the prices to the various subproblems and also the time spent in generating the columns to be included in the master problem. Time spent in solving the master problem includes the time spent in updating the master problem as well as in aggregation. We do not include the time taken by Algorithm 1 to preprocess the SDP; in any case, this time is negligible in comparison to the time taken by the decomposition algorithm to solve the problem. For the largest problem `thetaG77`, Algorithm 1 required

Prob	Iter	rel_acc	Time (h:m:s)	Iter	rel_acc	Time (h:m:s)	Iter	rel_acc	Time (h:m:s)
maxG67	50	9.7e-3	1:13:07	61	7.1e-3	1:31:25			
thetaG67	50	2.0e-2	1:51:28	100	7.1e-3	4:09:09	141	4.8e-3	6:19:05
maxG77	50	1.1e-2	1:36:29	71	8.0e-3	2:32:35			
thetaG77	50	1.5e-2	1:48:20	100	6.8e-3	4:07:48	142	4.0e-3	6:29:00

Table 4: Rate of convergence of decomposition algorithm

about 5 minutes to preprocess the problem. In fact, most of this time is spent in writing the preprocessed SDP in block-angular form in Step 6 of Algorithm 1.

For the smaller instances `maxG11`, `qpG11`, `maxG32`, and `maxG48` in Table 2, we provide computational results with different preprocessed SDP formulations. For instance, `maxG32` has 4 different formulations that are labeled as `maxG32_1`, `maxG32_2`, `maxG32_3` and `maxG32_4`, respectively. The parameters used in obtaining these formulations are given in Table 1. As one proceeds from `maxG32_1` to `maxG32_4`, the size of the largest semidefinite cone decreases, while the number of coupling constraints and blocks in the preprocessed SDP increase. We see that `maxG32_1` requires a larger solution time, with the subproblems accounting for a major share of the solution time. On the other hand, `maxG32_4` requires more iterations, with more time spent in solving the master problem than `maxG32_1`.

5.3 Rate of convergence of decomposition algorithm

We now discuss the rate of convergence of the decomposition algorithm on the 4 largest instances in Table 3. Table 4 presents the relative accuracy of the computed solution in the 50th, 100th, and the final iteration of the decomposition algorithm, and also the time taken to complete these iterations. We can see from Table 4 that we have to double the number of iterations (double the solution time) in order to reduce the relative accuracy of the computed solution by a factor of 2. Figure 2 shows the variation of the upper and lower bounds during one of the runs of the decomposition algorithm on the four largest instances `maxG67`, `thetaG67`, `maxG77`, and `thetaG77`, respectively. For all problems, we observe the tailing effect normally seen in decomposition algorithms for nonsmooth optimization, i.e., rapid progress with an accuracy level of $\epsilon = 10^{-1}$ attained within the first few iterations, and the $\epsilon = 10^{-2}$ accuracy level taking between 100-200 iterations. An accuracy level of $\epsilon = 10^{-3}$ on small problems typically takes between 200-1000 iterations.

5.4 Comparisons with serial version of CSDP on small instances

We solve the original SDPs for the small instances in Table 2 to 6 digits of accuracy with the serial version of CSDP 6.0 (see Borchers [4]) on the Henry2 cluster. We compare these solution times with the best solution times (see Table 2) for our decomposition algorithm

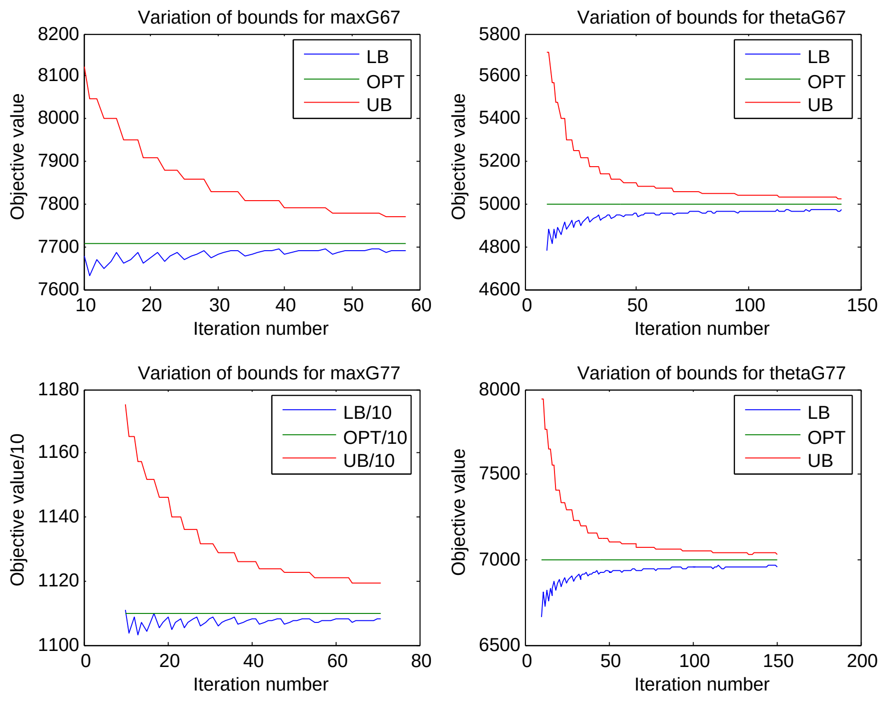

Figure 2: Variation of bounds for the 4 largest instances

Prob	Best time with MPI code on preprocessed SDPs – Table 2			Serial CSDP on original SDP		
	rel_acc	Time (Iter) (h:m:s)	Memory (GB)	dual_gap	Time (Iter) (h:m:s)	Memory (GB)
maxG11	6.0e-4	40 (30)	0.03	4.8e-7	14 (14)	0.06
qpG11	6.9e-4	1:06 (57)	0.07	8.7e-7	1:57 (16)	0.22
maxG32	5.3e-4	10:00 (157)	0.17	6.5e-7	3:54 (15)	0.37
thetaG11	5.9e-4	1:25 (28)	0.04	3.9e-7	55 (21)	0.1
maxG48	0.0	12:32 (38)	0.32	6.5e-7	9:47 (13)	0.83
qpG32	3.0e-5	14:41 (125)	0.28	7.9e-7	26:19 (16)	1.38

Table 5: Decomposition algorithm vs serial version of CSDP on small instances

on these instances. The results are summarized in Table 5, where we also compare the total memory taken by the two codes. The entry dual_gap in the table is the normalized duality gap between the primal and the dual objective values (see Borchers [4]). The serial version of CSDP is slightly faster but requires more memory than our decomposition algorithm on these instances.

5.5 Comparisons with OpenMP version of CSDP on large instances

We compare our decomposition algorithm with the 64 bit OpenMP version of CSDP 6.0 (see Borchers & Young [6]) on the large instances in Table 3. We installed the OpenMP version of CSDP 6.0 on the IBM p575, Power5 shared memory system [3] at NC State University. Each node on the Power5 system has eight 1.9GHz single core Power5 processors, and 32 GB of memory (much larger than Henry2's 4GB of memory per dual node). Each of the 8 processors in the Power5 shared memory system has access to the entire 32GB of shared memory.

The memory requirements of CSDP to solve an SDP of size n with m constraints is roughly $8(m^2 + 11n^2)$ bytes (see Borchers & Young [6]). The preprocessed SDP for instance `maxG57` has 61595 constraints (see Table 1). In fact, CSDP requires about 30.3GB of memory to store the Schur complement matrix for this SDP alone! As a result, CSDP runs out of memory on all the preprocessed SDP instances in Table 3 and we can only make comparisons with CSDP runs on original SDPs. We solve the original SDPs in Table 3 with the OpenMP version of CSDP 6.0 to 6 digits of accuracy with 1, 2, 4, and 8 processors, respectively. We compare the solution times, number of iterations, and the memory taken by CSDP with that of the MPI code on the preprocessed SDP with 1, 2, 4, 8, and 16 processors, respectively. The results with our MPI code are given in Table 6, while those with the OpenMP CSDP code are given in Table 7. The iteration and memory columns

[3]http://www.ncsu.edu/itd/hpc/Hardware/Hardware.php

Prob	rel_acc	Time (h:m:s)					Iter	Memory
		p=1	p=2	p=4	p=8	p=16		(GB)
thetaG32	5.0e-3	19:44	14:26	8:40	5:35	3:58	68	0.19
thetaG48	3.6e-3	56:39	54:25	48:04	39:45	25:17	89	0.32
maxG57	7.1e-3	42:36	37:36	32:18	29:01	17:31	63	0.51
thetaG57	3.9e-3	2:42:35	1:29:30	1:18:56	47:07	44:00	103	0.52
maxG62	7.7e-3	1:48:54	1:39:15	1:21:34	1:02:14	48:40	75	0.78
maxG65	6.8e-3	2:54:56	1:39:26	1:25:17	1:06:02	42:58	58	0.70
maxG66	6.9e-3	3:31:45	2:43:22	2:08:31	1:31:34	1:15:39	72	0.92
thetaG62	3.3e-3	4:28:24	3:08:32	2:44:23	2:22:54	1:46:40	126	0.89
maxG67	7.1e-3	3:16:40	2:08:40	1:31:25	1:30:21	1:31:25	61	0.98
thetaG65	3.6e-3	7:03:48	4:58:38	4:19:52	3:59:31	3:25:14	116	0.92
thetaG67	4.8e-3	-	11:13:25	9:43:20	8:25:47	6:19:05	141	1.5
maxG77	8.0e-3	-	4:21:11	3:35:31	3:02:44	2:32:35	71	1.43
thetaG77	4e-3	-	14:04:37	11:50:07	11:38:05	6:29:00	142	1.49

Table 6: MPI code on preprocessed SDP - Henry2 distributed memory system (p=1,2,4,8,16)

in Table 6 contain the number of iterations and the maximum memory taken by the MPI code with $p = 16$ processors. It must be mentioned that the number of iterations and the memory taken by our decomposition algorithm are practically constant as one increases the number of processors from 1 to 16. It is clear that CSDP requires a lot more memory than our MPI code and these memory requirements increase dramatically as the problem size increases. In fact, CSDP is only able to solve these problems due to the extra memory that is available. However, CSDP still runs out of memory on the largest problem thetaG77, where it requires about 20 times the memory taken by the MPI code. We wish to point out that the largest SDP solved in Borchers & Young [6] requires about 24.3GB of memory.

Comparing the solution times for the two algorithms with 1 processor (serial run) in Tables 6 and 7, we see that our decomposition algorithm is faster than CSDP on problems `maxG57`, `thetaG57`, `maxG62`, `maxG65`, `maxG66`, `thetaG62`, `maxG67`, and `thetaG65`. Moreover, our algorithm is also considerably faster than CSDP as the problem size increases. For instance on problem `maxG67`, our algorithm is more than twice as fast as CSDP. On the other hand, the OpenMP version of CSDP achieves a very good scalability on these problems as one increases the number of processors. This is due to the fact that each processor on the Power5 system has access to the entire 32GB of memory that eliminates any communication time between processors.

If we compare the solution times with 8 processors, we see that our solution times are competitive with that of CSDP, especially on the maxcut instances. For the larger maxcut

Prob	dual_gap	Time (h:m:s)				Iter	Memory
		p=1	p=2	p=4	p=8		(GB)
thetaG32	4e-7	12:01	6:54	4:07	2:43	23	0.63
thetaG48	5e-7	39:42	21:38	12:51	8:09	24	1.39
maxG57	8.5e-7	1:00:00	32:28	21:41	13:55	16	2.31
thetaG57	6.4e-7	2:44:10	1:29:17	52:37	36:01	23	3.84
maxG62	6.3e-7	2:32:10	1:26:08	52:48	36:01	16	4.51
maxG65	3e-7	3:58:30	2:08:26	1:18:51	51:11	18	5.88
maxG66	4e-7	5:27:01	2:59:38	1:51:10	1:12:39	17	7.43
thetaG62	7.6e-7	6:33:02	4:42:50	2:05:16	1:20:50	21	7.51
maxG67	2.5e-7	7:45:47	4:13:35	2:34:42	1:42:05	18	9.18
thetaG65	1.2e-7	10:39:10	5:50:43	3:23:05	2:08:55	23	9.81
thetaG67	1.6e-7	22:32:58	12:09:56	6:57:47	4:23:10	26	15.3
maxG77	7.3e-7	20:01:32	Failed	Failed	4:02:15	17	17.97
thetaG77	-	MM	MM	MM	MM	-	31.2

Table 7: OpenMP CSDP on original SDP - Power5 shared memory system (p=1,2,4,8)

instances `maxG67` and `maxG77`, our solution times are better than the OpenMP version of CSDP. However, the OpenMP version of CSDP achieves better solution times on the theta instances, where the decomposition algorithm requires more than 100 iterations to solve these problems to the default tolerance. We need at least 2 processors to solve the three largest instances: `thetaG67`, `maxG77`, and `thetaG77`, where each processor has only 2GB of memory.

5.6 Scalability

We now investigate the speedups attained by our parallel algorithm as one increases the number of processors. We choose the 12 largest problems from Table 1 for this test. We use the heuristic described in Section 5.1 to distribute the blocks among processors during the various runs. The results are shown in Table 8, where the columns denote the number of processors used. Moreover, there are two rows corresponding to each problem. The first row indicates the total time and the number of iterations needed by the algorithm to solve the problems in Tables 2 and 3. The 2nd row indicates the total time spent in solving the subproblems and the number of null steps. We see from Table 8 that our algorithm takes different number of iterations to solve a problem, as the number of processors is changed. This can be attributed to rounding errors in the algorithm. We define the speed up factor

Prob	p=1	p=2	p=4	p=8	p=16
thetaG32	19:44 (65)	14:26 (70)	8:40 (63)	5:35 (70)	3:58 (68)
	18:28 (46)	13:16 (51)	7:32 (44)	4:25 (52)	2:36 (48)
thetaG48	56:39 (87)	54:25 (85)	48:04 (90)	39:45 (88)	25:17 (89)
	53:59 (66)	51:33 (66)	43:38 (69)	36:34 (67)	21:33 (70)
maxG57	42:36 (65)	37:36 (65)	32:18 (64)	29:01 (71)	17:31(63)
	31:41 (48)	27:29 (48)	20:35 (47)	17:10 (54)	7:06 (46)
thetaG57	2:42:35 (108)	1:29:30 (104)	1:18:56 (105)	47:07 (94)	44:00 (103)
	2:14:04 (87)	1:13:46 (82)	1:01:42 (84)	33:26 (72)	28:22 (82)
maxG62	1:48:54 (70)	1:39:15 (73)	1:21:34 (72)	1:02:14 (74)	48:40 (75)
	1:17:25 (52)	1:06:28 (55)	49:50 (54)	32:16 (56)	16:32 (57)
maxG65	2:54:56 (60)	1:39:26 (60)	1:25:17 (58)	1:06:02 (59)	42:58 (58)
	2:13:10 (42)	1:16:47 (42)	1:03:23 (41)	48:06 (41)	25:24 (41)
maxG66	3:31:45 (72)	2:43:22 (70)	2:08:31 (70)	1:31:34 (71)	1:15:39 (72)
	2:24:26 (54)	1:53:17 (53)	1:26:51 (53)	50:56 (53)	30:01 (54)
thetaG62	4:28:24 (127)	3:08:32 (113)	2:44:23 (120)	2:22:54 (138)	1:46:40 (126)
	3:33:20 (105)	2:26:44 (91)	2:02:30 (98)	1:28:26 (116)	51:58 (104)
maxG67	3:16:40 (61)	2:08:40 (61)	1:31:25 (59)	1:30:21 (60)	1:31:25 (61)
	2:33:09 (43)	1:40:54 (43)	1:02:30 (42)	1:04:08 (43)	1:03:08 (43)
thetaG65	7:03:48 (125)	4:58:38 (123)	4:19:52 (116)	3:59:31 (118)	3:25:14 (116)
	5:45:39 (103)	4:09:15 (102)	3:33:11 (95)	3:12:15 (97)	2:36:31 (95)
thetaG67	-	11:13:25 (161)	9:43:20 (152)	8:25:47 (145)	6:19:05 (141)
	-	8:01:01 (138)	6:47:16 (128)	5:34:20 (122)	3:37:25 (118)
maxG77	-	4:21:11 (72)	3:35:31 (71)	3:02:44 (69)	2:32:35 (71)
	-	2:51:05 (53)	2:16:23 (52)	1:49:42 (50)	1:13:41 (52)
thetaG77	-	14:04:37 (154)	11:50:07 (149)	11:38:05 (154)	6:29:00 (142)
	-	10:00:40 (131)	8:04:48 (126)	6:29:19 (131)	2:46:05 (119)

Table 8: Scalability of decomposition algorithm on selected problems on up to 16 processors

$S(p)$ as follows:

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

We plot the speed ups attained by our decomposition algorithm on `thetaG32`, `maxG57` (two smallest instances in Table 8); `maxG66` and `thetaG62` (two medium instances in Table 8); and `thetaG67` and `maxG77` (two largest instances in Table 8) with $p = 1, 2, 4, 6, 8, 10, 12, 14, 16$ processors in Figure 3. For the smaller problems `thetaG32` and `maxG57`, we obtain speed ups of 5 and 2.5 respectively as the number of processors is increased from 1 to 16. For the medium sized problems `maxG66` and `thetaG62`, we obtain speed ups of 3 and 2.5 respectively as the number of processors is increased from 1 to 16. For the two largest problems `thetaG67` and `maxG77`, we obtain speed ups only close to 2 as the number of processors is increased from 2 to 16. The degradation in the speed up for larger problems is primarily due to communication costs between the processors. Also, it might be possible to improve these speed ups by a better distribution of blocks among the various processors. This is a topic for future investigation. For some problems like `maxG67`, our decomposition algorithm takes about the same time to solve the problem with 4, 8, and 16 processors.

6 Conclusions

We have presented a two stage interior point decomposition algorithm for solving large SDP relaxations of sparse maxcut, stable set, and box constrained quadratic problems. There are 3 main ingredients in our algorithm: a) a preprocessing algorithm (Algorithm 1) based on the matrix completion scheme of Fukuda et al. [11] that processes an existing SDP into an equivalent problem in the block-angular form, b) a regularized decomposition algorithm (Algorithm 2) that exploits the special structure of block-angular SDPs, and c) a parallel implementation of the decomposition algorithm on a distributed cluster of workstations.

We demonstrate via computational experiments that our algorithm: a) solves large SDPs to 2-3 digits of accuracy, where IPMs such as CSDP run out of memory, b) returns competitive solution times with the OpenMP version of CSDP on large instances, and c) attains a good parallel scalability. Comparing our results with Fujisawa et al. [10], we also show that a suitable modification of the matrix completion scheme can be used in the solution of larger SDPs than was previously possible.

We are presently working on a hybrid (OpenMP-MPI) version of the decomposition algorithm, where each subproblem is solved in parallel with the OpenMP version of CSDP. Warm-starting the subproblems is also an important issue that needs to be addressed.

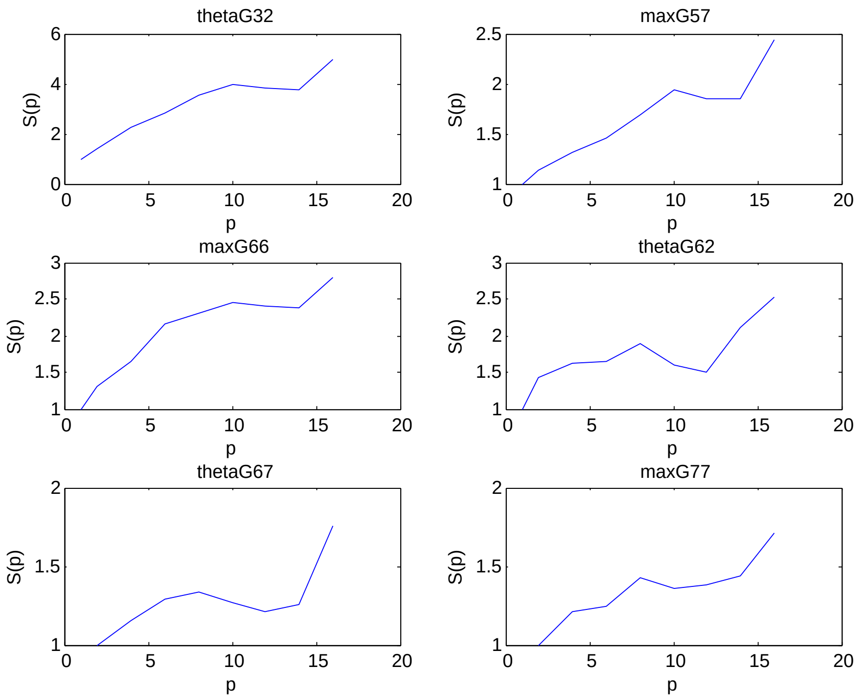

Figure 3: Speed ups attained with $p = 1, 2, 4, 6, 8, 10, 12, 14, 16$ processors

7 Acknowledgments

The author would like to thank Mituhiro Fukuda for help with the matrix completion code, Gary Howell at NC State, and the reviewers for comments that improved the presentation of the paper.

References

[1] S. Benson, Y. Ye, and X. Zhang, *Solving large scale sparse semidefinite programs for combinatorial optimization*, SIAM Journal on Optimization, 10(2000), pp. 443-461.

[2] D.P. Bertsekas, *Nonlinear Programming*, 2nd edition, Athena Scientific, Belmont, Massachusetts, 1999.

[3] J.R.S. Blair and B. Peyton, *An introduction to chordal graphs and clique trees*, in Graph Theory and Sparse Matrix Computation, edited by A. George, J.R. Gilbert, and J.W.H. Liu, Springer Verlag, New York, 1993, pp. 1-29.

[4] B. Borchers, *CSDP, a C library for semidefinite programming*, Optimization Methods and Software, 11 & 12(1999), pp. 613-623. Available at `https://projects.coin-or.org/Csdp/`.

[5] B. Borchers, *SDPLIB 1.2, A library of semidefinite programming test problems*, Optimization Methods and Software, 11, 1999, pp. 683-690. Available at `http://infohost.nmt.edu/~sdplib/`.

[6] B. Borchers and J. Young, *Implementation of a primal-dual method for SDP on a shared memory parallel architecture*, Computational Optimization and Applications, 37(2007), pp. 355-369.

[7] S. Burer and R.D.C. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low rank factorization*, Mathematical Programming, 95(2003), pp. 329-357.

[8] *ILOG CPLEX: High Performance software for mathematical programming*, `http://www.ilog.com/products/cplex/`.

[9] E. de Klerk, *Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications*, Kluwer Academic Publishers, Dordrecht, 2002.

[10] K. Fujisawa, M. Fukuda, and K. Nakata, *Preprocessing sparse semidefinite programs via matrix completion*, Optimization Methods and Software, 21(2006), pp. 17-39.

[11] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, *Exploiting sparsity in semidefinite programming via matrix completion I: General framework*, SIAM Journal on Optimization, 11(2000), pp. 647-674.

[12] M. Goemans and D.P. Williamson, *Improved approximation algorithms for max cut and satisfiability problems using semidefinite programming*, Journal of A.C.M 42 (1995), pp. 1115-1145.

[13] R. Grone, C.R. Johnson, E.M. Sá, and H. Wolkowicz, *Positive definite completions of partial hermitian matrices*, Linear Algebra and its Applications, 58(1984), pp. 109-124.

[14] C. Helmberg, *Numerical evaluation of spectral bundle method*, Mathematical Programming, 95(2003), pp. 381-406.

[15] C. Helmberg and K.C. Kiwiel, *A spectral bundle method with bounds*, Mathematical Programming, 93(2002), pp. 173-194.

[16] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, SIAM Journal on Optimization, 10(2000), pp. 673-696.

[17] J.B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms*, Vol II, Springer Verlag, Berlin Heidelberg, 1993.

[18] K.C. Kiwiel, *An aggregate subgradient method for nonsmooth convex minimization*, Mathematical Programming, 27(1983), pp. 320-341.

[19] K.C. Kiwiel, *Proximity control in bundle methods for convex nondifferentiable optimization*, Mathematical Programming, 46(1990), pp. 105-122.

[20] C. Langbort, R. D'Andrea, L. Xiao, and S. Boyd, *A decomposition approach to distributed analysis of networked systems*, Proceedings of the 43rd IEEE Conference on Decision and Control, pp. 3980-3985.

[21] S. Mehrotra and M.G. Özevin, *Decomposition based interior point methods for two-stage stochastic semidefinite programming*, SIAM Journal on Optimization, 18(2007), pp. 206-222.

[22] G. Karypis and V. Kumar, *METIS - A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0*, Department of Computer Science, University of Minnesota, 1998. Available at `http://glaros.dtc.umn.edu/gkhome/views/metis`.

[23] R.D.C. Monteiro, *First and second order methods for semidefinite programming*, Mathematical Programming, 97(2003), pp. 209-244.

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

[25] M.V. Nayakkankuppam, *Solving large-scale semidefinite programs in parallel*, Mathematical Programming, 109(2007), pp. 477-504.

[26] I. Nowak, *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*, Birkhäuser, 2005.

[27] P.S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, 1997.

[28] A. Ruszczyński, *Nonlinear Optimization*, Princeton University Press, 2006.

[29] A. Ruszczyński and A. Shapiro, editors, *Stochastic Programming*, Volume 10, Handbooks in Operations Research and Management Science, Elsevier, 2003.

[30] J.F. Sturm, *Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 & 12(1999), pp. 625-653. Available at `http://sedumi.mcmaster.ca/`.

[31] R.H. Tütüncü, K.C. Toh, and M.J. Todd, *Solving semidefinite-quadratic-linear programs using SDPT3*, Mathematical Programming, 95(2003), pp. 189-217. Available at `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.

[32] SPOOLES 2.2: Sparse Object Oriented Linear Equations Solver. Available at `http://www.netlib.org/linalg/spooles/spooles.2.2.html`.

[33] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, *Sums of squares and semidefinite programming for polynomial optimization problems with structured sparsity*, SIAM Journal of Optimization, 17(2006), pp. 218-242.

[34] M. Yamashita, K. Fujisawa, and M. Kojima, *SDPARA: SemiDefinite Programming Algorithm PARAllel version*, Parallel Computing, 29(2003), pp. 1053-1067. Available at `http://grid.r.dendai.ac.jp/sdpa/sdpa.6.00/sdpara.index.html`.