A Nonlinear Programming Algorithm for Solving Semidefinite Programs via Low-rank Factorization[*]

Samuel Burer[†] Renato D.C. Monteiro[‡]

March 9, 2001

Abstract

In this paper, we present a nonlinear programming algorithm for solving semidefinite programs (SDPs) in standard form. The algorithm's distinguishing feature is a change of variables that replaces the symmetric, positive semidefinite variable X of the SDP with a rectangular variable R according to the factorization $X = RR^T$. The rank of the factorization, i.e., the number of columns of R, is chosen minimally so as to enhance computational speed while maintaining equivalence with the SDP. Fundamental results concerning the convergence of the algorithm are derived, and encouraging computational results on some large-scale test problems are also presented.

Keywords: semidefinite programming, low-rank factorization, nonlinear programming, augmented Lagrangian, limited memory BFGS.

1 Introduction

In the past few years, the topic of semidefinite programming, or SDP, has received considerable attention in the optimization community, where interest in SDP has included the investigation of theoretically efficient algorithms, the development of practical implementation codes, and the exploration of numerous applications. In terms of applications, some of the most intriguing arise in combinatorial optimization where SDPs serve as tractable, convex relaxations of NP-Hard problems. The progress in this area, however, has been somewhat slow due to the fact that, in practice, the theoretically efficient algorithms developed for SDP are actually quite time- and memory-intensive, a fact that is especially true for SDP relaxations of large-scale combinatorial optimization problems. Attempting to address these issues, a recent trend in SDP has been the development of practically efficient algorithms that are less likely to have strong theoretical guarantees. The present paper follows this trend by introducing a new, experimental nonlinear programming algorithm for SDP that exhibits strong practical performance.

[*]This research was supported in part by the National Science Foundation under grants CCR-9902010 and INT-9910084.

[†]School of Mathematics, Georgia Tech, Atlanta, GA 30332, USA (burer@math.gatech.edu).

[‡]School of ISyE, Georgia Tech, Atlanta, GA 30332, USA (monteiro@isye.gatech.edu).

In straightforward terms, semidefinite programming is a generalization of linear programming (LP) in which a linear function of a symmetric matrix variable X is minimized over an affine subspace of real symmetric matrices subject to the constraint that X be positive semidefinite. SDP shares many features of LP, including a large number of applications, a rich duality theory, and the ability to be solved (more precisely, approximated) in polynomial time. For a nice survey of SDP, we refer the reader to [18].

The most successful polynomial-time algorithms for LP have been the class of interior-point methods, which have been shown efficient in both theory and practice. With the advent of SDP, the interior-point algorithms for LP were extended to solve SDP in polynomial-time, and on small- to medium-scale problems, these interior-point algorithms have proven to be very robust, obtaining highly accurate optimal solutions in short amounts of time. Their performance on sparse, large-scale problems, however, has not been as impressive, the main reason being that it is difficult to preserve sparsity when computing the second-order search directions common to these types of interior-point algorithms. For the algorithms and issues surrounding the classical, second-order interior-point methods for SDP, see [18], and for a selection of papers which deal with sparsity in these types of interior-point methods, see [3, 10].

Recognizing the practical disadvantages of the classical interior-point methods, several researchers have proposed alternative approaches for solving SDPs. Common to each of these new approaches is an attempt to exploit sparsity more effectively in large-scale SDPs by relying only on first-order, gradient-based information. In [12], Helmberg and Rendl have introduced a first-order bundle method to solve a special class of SDP problems in which the trace of the primal matrix X is fixed. For the special case of the graph-theoretic maximum cut SDP relaxation, Homer and Peinado have shown in [13] how the change of variables $X = VV^T$, where V is a real square matrix having the same size as X, allows one to recast the SDP as an unconstrained optimization problem for which any standard nonlinear method—in particular, a first-order method—can be used. Burer and Monteiro [4] improved upon the idea of Homer and Peinado by simply noting that, without loss of generality, V can be required to be lower triangular in accordance with the Cholesky factorization. Then, in a series of papers [6, 7, 5], Burer, Monteiro, and Zhang showed how one could apply the idea of Cholesky factorization in the dual SDP space to transform any SDP into a nonlinear optimization problem over a simple feasible set. They also provided a globally convergent, first-order log barrier algorithm to solve SDPs via this method, one of the key features being the preservation of sparsity. Most recently, Fukuda and Kojima [9] have introduced an interior-point technique based on Lagrangian duality which solves the class of SDPs studied in [12] and allows the use of first-order methods in the unrestricted space of Lagrange multipliers.

The current paper follows the path laid by these alternative methods and is specifically motivated by [13, 4], that is, we consider the use of first-order methods for solving the nonlinear reformulation of an SDP obtained by replacing the positive semidefinite variable with an appropriate factorization. We work with the standard-form primal SDP

$$\min\{C \bullet X : A_i \bullet X = b_i, \; i = 1, \ldots, m, \; X \succeq 0\}, \tag{1}$$

where the data matrices C and $\{A_i\}_{i=1}^m$ are $n \times n$ real symmetric matrices, the data vector b is m-dimensional, the operator $\bullet$ denotes the inner product of matrices, and the $n \times n$ matrix variable X is required to be symmetric, positive semidefinite as indicated by the constraint $X \succeq 0$. Generally speaking, the constraint $X \succeq 0$ is the most challenging aspect of solving (1) since the objective function and constraints are only linear in X. Hoping simply to circumvent this difficult constraint, we introduce the change of variables $X = VV^T$ where V is a real, $n \times n$ matrix (that is not required to be symmetric). In terms of the new variable V, the resulting nonlinear program

$$\min\{C \bullet (VV^T) : A_i \bullet (VV^T) = b_i,\ i = 1, \ldots, m\} \tag{2}$$

is easily seen to be equivalent to (1) since every $X \succeq 0$ can be factored as VV^T for some V. Since the positive semidefiniteness constraint has been eliminated, (2) has a significant advantage over (1), but this benefit has a corresponding cost: the objective function and constraints are no longer linear—but instead quadratic and in general nonconvex.

Is it practical to optimize (2) in place of (1)? The answer is certainly not an immediate "yes" as there are several important questions that should be addressed:

Q1 The number of variables in V is n^2. Can this large number of variables be managed efficiently?

Q2 What optimization method is best suited for (2)? In particular, can the optimization method exploit sparsity in the problem data?

Q2 Since (2) is a nonconvex programming problem, can we even expect to find a global solution in practice?

To answer Q1, we appeal to a theorem that posits the existence of an optimal solution X^* of (1) having rank r satisfying the inequality $r(r+1)/2 \le m$. In terms of the reformulation (2), the existence of X^* implies the existence of some V^* satisfying $X^* = V^*(V^*)^T$ and having its last $n - r$ columns equal to zero. The idea to manage the n^2 variables of V is then simply to set the last $n - \bar{r}$ columns of V to zero, where $\bar{r}$ is taken large enough so as to not eliminate all optimal solutions. In other words, we ignore the last $n - \bar{r}$ columns in the optimization. As a consequence, the resulting optimization is equivalent to the original SDP while having far fewer variables. In answering Q2, we develop an effective limited memory BFGS augmented Lagrangian algorithm for solving (2) whose major computations require computer time and space that are directly proportional to the number of nonzeros in the data matrices C and $\{A_i\}_{i=1}^m$. For Q3, we present computational results which show that the method finds optimal solutions to (2) quite reliably, and although we are able to derive some amount of theoretical justification for this, our belief that the method is not strongly affected by the inherent nonconvexity of (2) is largely experimental. Finally, after positively addressing these three questions, the primary conclusion of this paper is that optimizing (2) in place of (1) is indeed practical, especially for large, sparse SDPs.

The paper is organized as follows. In Section 2, we discuss in detail the standard form SDP (1) as well as its nonlinear reformulation (2). In particular, we analyze optimality conditions and the consequences of the low-rank factorization theorem mentioned above. Then in Section 3, we describe our optimization technique for (2), focusing in particular on how to exploit sparsity in the data. In Section 4, we discuss and demonstrate an implementation of the proposed algorithm on two classes of large-scale SDPs. We compare our method with one of the classical interior-point methods as well as with the algorithm of Helmberg and Rendl [12] and conclude that our method outperforms both of the other methods in terms of time and solution quality. Lastly, in Section 5, we conclude the paper with a few final remarks and ideas for future research.

1.1 Notation and terminology

We use $\Re$, $\Re^p$, and $\Re^{p\times q}$ to denote the space of real numbers, real p-dimensional column vectors, and real $p \times q$ matrices, respectively. We use $\|\cdot\|$ to denote the Euclidean norm for vectors. By $\mathcal{S}^p$ we denote the space of real $p \times p$ symmetric matrices, and we define $\mathcal{S}^p_+$ and $\mathcal{S}^p_{++}$ to be the subsets of $\mathcal{S}^p$ consisting of the positive semidefinite and positive definite matrices, respectively. For a matrix $A \in \mathcal{S}^p$, we write $A \succeq 0$ and $A \succ 0$ to indicate that $A \in \mathcal{S}^p_+$ and $A \in \mathcal{S}^p_{++}$, respectively. We let trace(A) denote the trace of a matrix $A \in \Re^{n\times n}$, i.e. the sum of the diagonal elements of A. For $A, B \in \Re^{p\times q}$, we define $A \bullet B \equiv \text{trace}(A^T B)$. Let Diag : $\Re^p \to \Re^{p\times p}$ be the operator defined by Diag$(u) = U$, where U is the diagonal matrix having $U_{ii} = u_i$ for all $i = 1, \ldots, p$, and let diag : $\Re^{p\times p} \to \Re^p$ be defined as diag$(U) = u$, where $u_i = U_{ii}$ for all $i = 1, \ldots, p$. For a matrix A, we let $A_{i\cdot}$ and A_{ij} denote the i-th row and the (i, j)-th entry of A, respectively.

2 The Nonlinear Formulation

In this section, we explore the standard-form primal SDP (1) and its nonlinear formulation (2) in more detail . We first highlight some of the most important features of semidefinite programming and state our assumptions on (1). We then consider the consequences of the "low-rank" theorem mentioned in the introduction and finally derive various optimality conditions for (2).

2.1 The SDP problem and its low-rank reformulation

As stated in the introduction, the standard form primal SDP (1) is specified by data $C \in \mathcal{S}^n$, $\{A_i\}_{i=1}^m \subset \mathcal{S}^n$, and $b \in \Re^m$. Its variable X is an $n \times n$ symmetric matrix that is required to be positive semidefinite, that is, $X \in \mathcal{S}^n_+$ or $X \succeq 0$. We assume that the constraint matrices $\{A_i\}_{i=1}^m$ are linearly independent, and since $\mathcal{S}^n$ has dimension $n(n + 1)/2$, this means in particular that $m \le n(n + 1)/2$.

Associated with the primal SDP is the following dual SDP, with variables $(S, y) \in \mathcal{S}^n_+ \times \Re^m$:

$$\max \left\{ b^T y : S = C - \sum_{i=1}^m y_i A_i,\ S \succeq 0 \right\}. \tag{3}$$

We assume that (1) and (3) have nonempty optimal solution sets with zero duality gap, that is, we assume the existence of feasible solutions X^* and (S^*, y^*) such that $C \bullet X^* = b^T y^*$. The following fundamental proposition (whose proof can be found for example in Corollary 2.1 of [16]) will be useful to us in our analysis of the nonlinear reformulation of the primal SDP.

Proposition 2.1 *Feasible solutions X and (S, y) are simultaneously optimal if and only if $X \bullet S = 0$, or equivalently, $XS = SX = 0$.*

Since every $X \in \mathcal{S}^n_+$ can be factored as $X = VV^T$ for some $V \in \Re^{n \times n}$, the primal SDP (1) can be reformulated as the nonlinear program (2) in terms of the unrestricted variable V. Note that variations of (2) can be obtained by requiring structure on the factorization $X = VV^T$. For example, we could impose the requirement that V be lower triangular with nonnegative diagonal elements, in which case the factorization would represent the Cholesky factorization. Another possibility is to require that V be symmetric positive semidefinite making V the matrix square root of X. In fact, any type of factorization that is valid for all feasible X can be used in place of the generic factorization to reformulate the SDP.

A slightly different perspective on reformulating the SDP through factorization is the following: instead of using a factorization that is valid for all feasible solutions, use one that is valid for some or all optimal solutions. Such a reformulation will clearly have an optimal solution set which is—via the factorization—a subset of the SDP optimal solution set. Pursuing this idea, consider the following theorem, which was proven concurrently in Barvinok [2] and Pataki [17]:

Theorem 2.2 *There exists an optimal solution X^* of (1) with rank r satisfying the inequality $r(r+1)/2 \le m$.*

Since a matrix $X \in \mathcal{S}^n_+$ with rank r can be factored as $X = VV^T$ for some $V \in \Re^{n \times n}$ having its last $n - r$ columns equal to zero, we can use the above theorem to impose a structure on the factorization $X = VV^T$ that is valid for some of the SDP optimal solutions (but not valid for all feasible solutions). In particular, if we define

$$r^* = \min\{\text{rank}(X^*) : X^* \text{ is optimal for (1) }\}, \tag{4}$$

$$\bar{r} = \max\{r \ge 0 : r(r+1)/2 \le m\}, \tag{5}$$

then $r^* \le \bar{r} \le n$, and so the nonlinear program

$$\min\{C \bullet (VV^T) : A_i \bullet (VV^T) = b_i,\ i = 1, \ldots, m,\ V_{\cdot j} = 0,\ j = \bar{r} + 1, \ldots, n\} \tag{6}$$

is equivalent to the SDP (1) in the sense of the previous paragraph. The advantage of (6) over (2) is clear: the number $n\bar{r}$ of variables in (6) can be (and is typically) much smaller than the number n^2 in (2).

We will find it useful to discuss the above ideas in a slightly more general context and using slightly different notation. For this, we introduce the following nonlinear program, dependent upon a positive integer $r \le n$:

$$(N_r) \qquad \min\{C \bullet (RR^T) : A_i \bullet (RR^T) = b_i,\ i = 1, \ldots, m,\ R \in \Re^{n \times r}\}.$$

Note that the distinguishing feature of (N_r) is the rectangular shape of the matrix R, which has n rows but only r columns. Note also that (N_n) is equivalent to (2) and $(N_{\bar{r}})$ is equivalent to (6).

2.2 Optimality conditions

We wish to analyze the optimality conditions of the nonlinear program (N_r) for a fixed r, and so we define the usual Lagrangian function

$$L(R, y) = C \bullet (RR^T) - \sum_{i=1}^{m} y_i (A_i \bullet (RR^T) - b_i), \tag{7}$$

where $y \in \Re^m$ is the vector of unrestricted Lagrange multipliers for the equality constraints of (N_r). Introducing the auxiliary variable $S \in \mathcal{S}^n$ defined by

$$S = C - \sum_{i=1}^{m} y_i A_i, \tag{8}$$

the Lagrangian can be rewritten more simply as

$$L(R, y) = S \bullet (RR^T) + b^T y.$$

It is important to the note the relationship of the dual SDP with the Lagrange multipliers y and the auxiliary variable S. Comparing (8) and (3), we see that, in both cases, the relationship between y and S is the same. Moreover, in the case of the Lagrangian, if S happens to be positive semidefinite, then (S, y) constitutes a feasible solution of the dual SDP.

Consider the following easily derived formulas:

$$\begin{aligned} \nabla_R \left(A_i \bullet (RR^T) - b_i \right) &= 2\, A_i R, \\ \nabla_R L(R, y) &= 2\, SR, \\ L''_{RR}(R, y)[D, D] &= 2\, S \bullet (DD^T) \text{ for } D \in \Re^{n \times r}. \end{aligned} \tag{9}$$

Combining these formulas with standard results from nonlinear programming, we have the following proposition concerning the local minimizers of (N_r).

Proposition 2.3 *Let R^* be a local minimum of (N_r), and suppose that R^* is a regular point, i.e., the gradients $\{A_i R\}_{i=1}^m$ of the constraints at R^* are linearly independent. Then there exists a unique Lagrange multiplier vector $y^* \in \Re^m$ and corresponding S^* such that*

$$S^* R^* = 0. \tag{10}$$

Moreover, the inequality

$$S^* \bullet (DD^T) \geq 0 \tag{11}$$

holds for all matrices $D \in \Re^{n \times r}$ satisfying

$$A_i R^* \bullet D = 0 \quad \forall \; i = 1, \ldots, m. \tag{12}$$

Proposition 2.3 states the first- and second-order necessary conditions for a feasible point R to be a local minimizer of (N_r). It is also possible to formulate the standard sufficient conditions for R being a strict local minimum. It is easy to see, however, that (N_r) has no strict local minima. To see this, let R be any feasible point, and let $Q \in \Re^{r \times r}$ be an arbitrary orthogonal matrix. Then the point RQ is also feasible, and its objective value equals that of R, as the following equations demonstrate:

$$A_i \bullet ((RQ)(RQ)^T) = A_i \bullet (RQQ^T R^T) = A_i \bullet (RR^T) = b_i,$$
$$C \bullet ((RQ)(RQ)^T) = C \bullet (RQQ^T R^T) = C \bullet (RR^T).$$

Since Q can be chosen so that RQ is arbitrarily close to R, it follows that R cannot be a strict local minimum.

Even though the standard sufficiency conditions are not relevant in the current context, there are some interesting sufficient conditions for a feasible point R of (N_r) to yield an optimal SDP solution $X = RR^T$. Said differently, there are conditions—irrespective of the integer r—which guarantee that a feasible point R is an optimal solution of (N_n), or equivalently of (2), when R is included in the feasible set of (N_n) in the obvious way. These conditions are stated in the following two propositions.

Proposition 2.4 *Let R^* be a stationary point of (N_r), i.e., there exists $y^* \in \Re^m$ such that $\nabla_R L(R^*, y^*) = 0$. If the associated matrix $S^* \in \mathcal{S}^n$ is positive semidefinite, then $X^* = R^*(R^*)^T$ and (S^*, y^*) are optimal for (1) and (3), respectively.*

Proof. First note that X^* is feasible for (1) since R^* is feasible for (N_r) and that S^* is feasible for (3) since it is related to y^* by (8) and is also positive semidefinite. Second, the condition $\nabla_R L(R^*, y^*) = 0$ can be rewritten as the equation $S^* R^* = 0$ according to (9). From this we easily that $S^* R^* (R^*)^T = S^* X^* = 0$. The claim of the proposition now follows by combining the facts just derived with Proposition 2.1. ■

Proposition 2.5 *Let $r < n$, and suppose that $R^* \in \Re^{n \times r}$ satisfies the hypotheses of Proposition 2.3 for (N_r) with associated (S^*, y^*). Let $\hat{R}$ be the injection of R^* into $\Re^{n \times (r+1)}$. If $\hat{R}$ is a local minimum of (N_{r+1}), then $X^* = R^*(R^*)^T$ and (S^*, y^*) are optimal for (1) and (3), respectively.*

Proof. The linear independence of $\{A_i R^*\}_{i=1}^m$ implies the linear independence of $\{A_i \hat{R}\}_{i=1}^m$ since $A_i \hat{R}$ is simply the injection of $A_i R^*$ into $\Re^{n \times (r+1)}$. Then, since $\hat{R}$ is a local minimum of (N_{r+1}), $\hat{R}$ satisfies the hypotheses of Proposition 2.3 for (N_{r+1}), and so there exists a unique $\hat{y} \in \Re^m$ and an associated $\hat{S}$ that fulfill the conclusions of Proposition 2.3. Using the same proposition as it applies to R^* and (N_r), we see that $S^* R^* = 0$. Since $\hat{R}$ is simply R^* with an additional zero column appended, we see that $S^* \hat{R} = 0$. Thus, using the uniqueness of $\hat{y}$, we conclude that $\hat{y} = y^*$ and $\hat{S} = S^*$.

Since the $(r+1)$-st column of $\hat{R}$ is zero, we have that the $(r+1)$-st column of $A_i \hat{R}$ is zero for all $i = 1, \ldots, m$. It follows that any matrix $D \in \Re^{n \times (r+1)}$ having its first r columns equal to zero satisfies $A_i \hat{R} \bullet D = 0$ for all i, and from Proposition 2.3, we have that $\hat{S} \bullet (DD^T) \geq 0$ for all such D. In particular, let $d \in \Re^n$ be an arbitrary vector, and consider the matrix $D \in \Re^{n \times (r+1)}$ formed by replacing the $(r+1)$-st column of the zero matrix with d. We then have

$$\hat{S} \bullet (DD^T) = \hat{S} \bullet (dd^T) = d^T \hat{S} d \geq 0.$$

Since d is arbitrary, the above inequality proves that $\hat{S}$ is positive semidefinite.

So $S^* = \hat{S}$ is positive semidefinite. Thus, the conclusion of the proposition follows from Proposition 2.4. ∎

Proposition 2.4 has a special significance when we consider how to solve the primal SDP practically using the reformulations (N_r). For example, suppose we attempt to solve the formulation (N_r) using an algorithm that computes a local minimum by explicitly computing the Lagrange multipliers y. Once a local minimum is obtained, if the current S is positive semidefinite, we can conclude by Proposition 2.4 that the current $X = RR^T$ and (S, y) are optimal primal-dual SDP solutions. It is important to note, however, that the choice of r must be large enough so that at least one SDP optimal solution X has rank r, that is, r must be at least as big as r^* defined in (4). Otherwise, (N_r) would not be equivalent to the SDP (1), implying that S could never be positive semidefinite at a stationary point.

Proposition 2.5 also presents an intriguing algorithmic possibility. Suppose that, before solving (1), we know the exact value of r^*. Then it is best to solve (N_r) for $r = r^*$. More specifically, if we choose $r < r^*$, then (N_r) is not equivalent to (1), and if we choose $r > r^*$, then we are solving a larger program than necessary. Of course, since we do not know the value of r^* ahead of time, Theorem 2.2 guarantees that by solving (N_r) for $r = \bar{r} \geq r^*$, we will solve the SDP. Nonetheless, if $\bar{r}$ happens to be much bigger than r^*, we would be solving a much larger program than necessary. Proposition 2.5, however, suggests a scheme to solve the SDP which avoids solving (N_r) for $r > r^*$:

0. Choose r small and compute a local minimum R of (N_r).
1. Use an optimization technique either (a) to determine that the injection $\hat{R}$ of R into $\Re^{n\times(r+1)}$ is a local minimum of (N_{r+1}) or (b) to compute a better local minimum $\tilde{R}$ of (N_{r+1}).
2. If (a) holds, then $X = RR^T$ is SDP optimal by Proposition 2.5; otherwise, repeat step 1 with $R = \tilde{R}$ and $r = r + 1$.

We remark that in step 1, if $\hat{R}$ is not a local minimum, then it is a saddle point and hence can be escaped from. The goal of this scheme is to solve the SDP when (N_r) first is equivalent to (1), i.e., when r first equals r^*. There are, of course, some theoretical and computational considerations with such a scheme (for example, the linear independence assumptions of Proposition 2.5 and the optimization technique of step 1), but in Section 4, we show that an adaptation of this scheme allows us to solve the SDP much faster than just solving $(N_{\bar{r}})$ directly.

3 The Optimization Method

In this section, we describe a practical algorithm for obtaining a local minimizer of the nonlinear program (N_r). The key features of the algorithm are its ability to handle the nonconvex equality constraints of (N_r) and its exploitation of sparsity in the problem data via first-order search directions.

3.1 The augmented Lagrangian algorithm

As mentioned in the introduction, the price we pay for the elimination of the difficult constraint $X \succeq 0$ in (1) via the factorization $X = RR^T$ (or $X = VV^T$) is the introduction of the difficult constraints $A_i \bullet (RR^T) = b_i$. Since we assume no structure on $A_i \in \mathcal{S}^n$, these constraints are in general nonconvex. Hence, any optimization method we choose for solving (N_r) must address these difficult constraints.

The nonlinear programming method that we believe is a good candidate for solving (N_r) is the augmented Lagrangian method (also called the method of multipliers). The basic idea behind the method is the idea of penalization, i.e., the method ignores the constraints all together and instead optimizes an objective which includes additional terms that penalize infeasible points. Penalization alone, however, can lead to ill-conditioning in the optimization, and so a feature of the augmented Lagrangian method is to introduce explicit Lagrange multipliers y_i, one for each constraint, that help to balance the ill-conditioning induced by the penalization. In addition to mitigating the penalization, the multipliers also can serve as a test for optimality (as discussed at the end of Section 2) and as a direct connection to the dual problem (3).

In the following description of the augmented Lagrangian algorithm for solving (N_r), we assume that r has been chosen so that (N_r) is feasible.

A key component of the augmented Lagrangian algorithm is the following function, called the augmented Lagrangian function:

$$\mathcal{L}(R, y, \sigma) = C \bullet (RR^T) - \sum_{i=1}^{m} y_i (A_i \bullet (RR^T) - b_i) + \frac{\sigma}{2} \sum_{i=1}^{m} (A_i \bullet (RR^T) - b_i)^2, \qquad (13)$$

where the variables $R \in \Re^{n \times r}$ and $y \in \Re^m$ are unrestricted and the parameter $\sigma \in \Re$ is positive. Comparing (13) with (7), we see that the augmented Lagrangian function $\mathcal{L}$ differs from the usual Lagrangian L only in the addition of the term involving σ. This term measures the Euclidean norm of the infeasibility of R with respect to (N_r) and is scaled by the real number σ. As such, σ is called the penalty parameter. The motivation for using the augmented Lagrangian algorithm is that, for an appropriate, fixed choice (y^*, σ_*), an optimal solution R^* of (N_r) can be found by simply optimizing the function $\mathcal{L}(\cdot, y^*, \sigma_*)$ with respect to R (see for example [8] for more details).

Of course, the trick is to determine (y^*, σ_*), and the augmented Lagrangian algorithm attempts to do so by forming a sequence $\{(y^k, \sigma_k)\}_{k \ge 0}$ that converges to (y^*, σ_*). This is done by minimizing $\mathcal{L}(\cdot, y^k, \sigma_k)$ with respect to R in order to find its optimal solution R^k and then using (R^k, y^k, σ_k) to determine a new pair (y^{k+1}, σ_{k+1}), from which the process is continued. The exact method by which (y^k, σ_k) is updated to (y^{k+1}, σ_{k+1}) is an important theoretical and practical detail of the algorithm. In implementations, the update rule is typically given as follows: parameters $\gamma > 1$ and $\eta < 1$ are given and an auxiliary scalar v_k is introduced; then

(i) compute $v = \sum_{i=1}^{m} (A_i \bullet (R^k (R^k)^T) - b_i)^2$;

(ii) if $v < \eta\, v_k$, then set

$$\begin{aligned} y_i^{k+1} &= y_i^k - \sigma_k(A_i \bullet (R^k(R^k)^T) - b_i) \text{ for all } i, \\ \sigma_{k+1} &= \sigma_k, \\ v_{k+1} &= v; \end{aligned}$$

(iii) otherwise, set

$$\begin{aligned} y_i^{k+1} &= y_i^k \text{ for all } i, \\ \sigma_{k+1} &= \gamma\, \sigma_k, \\ v_{k+1} &= v_k. \end{aligned}$$

In words, the quantity v represents the infeasibility of R^k, and the quantity v_k is the best infeasibility obtained by some point R during prior iterations of the algorithm. The hope is that v is smaller than $\eta\, v_k$, meaning that R^k has obtained a new best infeasibility. If v is in fact smaller than $\eta\, v_k$, then item (ii) details how to update the Lagrange multipliers and the penalty parameter (which stays the same), and then v_{k+1} carries the new best infeasibility v. If, on the other hand, $v \geq \eta\, v_k$, then item (iii) increases the penalty parameter by a factor of γ and keeps the other parameters fixed, all with the goal of reducing the infeasibility to the target level $\eta\, v_k$. We remark that typical choices for γ and η are 10 and 1/4, respectively. In addition, some methods choose to update the parameters γ and η dynamically during the course of the algorithm. This is actually the approach that we take in our numerical experiments (see Section 4 for more details).

Under reasonable assumptions, the sequence $\{(y^k, \sigma_k)\}_{k\geq 0}$ converges to $\{(y^*, \sigma_*)\}$, and the sequence $\{R^k\}_{k\geq 0}$ also converges to R^*. However, since obtaining the exact optimal solution R^k of $\mathcal{L}(\cdot, y^k, \sigma_k)$ is unlikely in a practical implementation, if the sequence $\{R^k\}_{k\geq 0}$ is instead replaced with a sequence of stationary points $\{\bar{R}^k\}_{k\geq 0}$ (or even approximate stationary points), then it can be proven that $\{\bar{R}^k\}_{k\geq 0}$ will converge to a stationary point $\bar{R}^*$ instead (see for example [8]). In practice, we could probably expect $\bar{R}^*$ to be not just a stationary point, but rather a local minimum of (N_r). In fact, the computational results that we present in the next section demonstrate that $\bar{R}^*$ is likely to be a global solution of (N_r).

The method used to perform the unconstrained minimization of $\mathcal{L}(\cdot, y^k, \sigma_k)$ with respect to R naturally plays a critical role in the overall efficiency of the augmented Lagrangian algorithm. For this, we have chosen a first-order, limited memory BFGS approach that employs a strong Wolfe-Powell line search (see for example [8]), and the number of limited memory BFGS updates that we store is three. We prefer a gradient-based algorithm because the function and gradient evaluations of $\mathcal{L}(\cdot, y^k, \sigma_k)$ can be performed very quickly, especially when r is small and the data matrices are very sparse (as detailed in the next subsection). In addition, it is not difficult to see that computing and factoring the Hessian of $\mathcal{L}(\cdot, y^k, \sigma_k)$ (as in the application of Newton's method) would consume large amounts of space and time.

3.2 The function and gradient evaluations

Since one of the stated goals of our optimization of (N_r) is the ability to exploit sparsity and problem structure, we now wish to discuss briefly how the augmented Lagrangian algorithm can do exactly this. The main computational work of the algorithm lies in the solution of the subproblems $\min\{\mathcal{L}(R, y^k, \sigma_k) : R \in \Re^{n \times r}\}$, and the limited memory BFGS algorithm that we have chosen to perform this unconstrained minimization uses function and gradient evaluations as its main computational engine. Hence, we focus our attention on the work involved in computing the function value and gradient (with respect to R) of the augmented Lagrangian function $\mathcal{L}(\cdot, y, \sigma)$, where $y \in \Re^m$ and $\sigma > 0$ are fixed.

From (13), it is not difficult to see that the main work involved in evaluating $\mathcal{L}(R, y, \sigma)$ is in the computation of the quantities $C \bullet (RR^T)$ and $A_i \bullet (RR^T)$ for $i = 1, \ldots, m$. In particular, once these $m+1$ quantities are computed, the function value can easily be obtained in roughly $\mathcal{O}(m)$ additional flops. So how can these dot products be performed efficiently? We will consider two cases that arise in the example SDPs of the section concerning computational results.

First, suppose all of the data matrices are sparse but have arbitrary structure. Letting W represent any one of the data matrices, we have

$$W \bullet (RR^T) = \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} [RR^T]_{ij} = \sum_{W_{ij} \neq 0} W_{ij} \sum_{k=1}^{r} R_{ik} R_{jk}. \tag{14}$$

The final summation clearly shows that the work needed to compute $W \bullet (RR^T)$ is proportional to the number of nonzeros of W times the work involved in computing the dot product of the i-th and j-th rows of R, i.e., the work involved is $\mathcal{O}(|W|r)$. Hence, the augmented Lagrangian function value can be computed in $\mathcal{O}((|C| + |A_1| + \cdots + |A_m|)r)$ flops. This quantity, however, represents some redundant computations. In particular, if two or more of the data matrices share a nonzero position (i, j), then the dot product between the i-th and j-th rows of R will have been computed more than once. To fix this, we first compute and store $[RR^T]_{ij}$ for any (i, j)-th entry that is nonzero for some data matrix. Then, for each data matrix W, $W \bullet (RR^T)$ can be computed in $\mathcal{O}(|W|)$ additional flops. Since, the nonzero pattern of the matrix S defined by (8) is precisely the combined nonzero patterns of the data matrices, we have the following proposition.

Proposition 3.1 *When the data matrices are arbitrary, the time required to evaluate the augmented Lagrangian function $\mathcal{L}(R, y, \sigma)$ is $\mathcal{O}(|S|r + |C| + |A_1| + \cdots + |A_m|)$.*

Second, suppose each of the data matrices is a rank-one matrix, i.e., each data matrix W equals ww^T for some $w \in \Re^n$. In this case, W may not be sparse but is nonetheless very structured. Again, the main work in evaluating the function is in computing $W \bullet (RR^T)$, which can arranged as

$$W \bullet (RR^T) = (ww^T) \bullet (RR^T) = \|R^T w\|^2.$$

Since $\|R^T w\|^2$ can be calculated in $\mathcal{O}(nr)$ flops, we have the following proposition.

Proposition 3.2 *When the data matrices have rank equal to one, the time required to evaluate the augmented Lagrangian function $\mathcal{L}(R, y, \sigma)$ is $\mathcal{O}(mnr)$.*

A couple of remarks concerning the two above propositions are in order. First, the two propositions can be joined in a straightforward way to handle the case when some of the data matrices are sparse and arbitrary while others are rank-one. Second, it can be easily seen that the space requirements of the function evaluation beyond that needed for the data matrices and the point R are on the order of $\mathcal{O}(|S|)$ for arbitrary data matrices and $\mathcal{O}(mr)$ for rank-one matrices (if the r-vectors $R^T w$ are stored as will be convenient for the gradient computation).

Now turning to the gradient evaluation, we first point out that the formula for the gradient of $\mathcal{L}(R, y, \sigma)$ with respect to R is

$$\nabla_R \mathcal{L}(R, y, \sigma) = 2\,CR - 2\sum_{i=1}^{m} y_i A_i R + 2\,\sigma \sum_{i=1}^{m} (A_i \bullet (RR^T) - b_i) A_i R = 2\,\tilde{S} R, \tag{15}$$

where, if we let $\tilde{y}_i = y_i - \sigma(A_i \bullet (RR^T) - b_i)$,

$$\begin{aligned} \tilde{S} &= C - \sum_{i=1}^{m} y_i A_i + \sigma \sum_{i=1}^{m} (A_i \bullet (RR^T) - b_i) A_i \\ &= C - \sum_{i=1}^{m} \left[y_i - \sigma(A_i \bullet (RR^T) - b_i) \right] A_i \\ &= C - \sum_{i=1}^{m} \tilde{y}_i A_i. \end{aligned}$$

Notice that $\tilde{y}_i$ can be computed directly if we make the reasonable assumption that the function value at R has already been computed.

In the case that the data matrices have arbitrary structure, a reasonable approach to compute the gradient is simply to form $\tilde{S}$ and compute the gradient according to (15), making sure to take advantage of the sparsity that $\tilde{S}$ inherits from the data matrices. This procedure of forming $\tilde{S}$ and then computing the matrix product can easily be seen to cost $\mathcal{O}(|S|r + |C| + |A_1| + \cdots + |A_m|)$ flops. In the case that the data matrices each have rank one, for each data matrix W, WR can be computed in $\mathcal{O}(n^2)$ flops according to the formula $WR = w(w^T R)$, assuming that $R^T w$ has been stored from the function evaluation. Because of this, the most practical way to form the gradient is to first compute WR for each data matrix W, and then to combine the resulting matrices according to the linear combination of matrices defining $\tilde{S}$. Such a procedure gives an overall flop count of $\mathcal{O}(mn^2 + mnr)$. Putting both cases together, we have the following proposition.

Proposition 3.3 *When the data matrices are arbitrary, the time required to evaluate the gradient of the augmented Lagrangian function $\mathcal{L}(R, y, \sigma)$ with respect to R is $\mathcal{O}(|S|r + |C| + |A_1| + \cdots + |A_m|)$. When the data matrices have rank equal to one, the time is $\mathcal{O}(mnr + mn^2)$.*

Once again, we remark that the space requirements are not excessive and, in particular, can be managed to be $\mathcal{O}(|S| + nr)$ in the case of arbitrary matrices and $\mathcal{O}(nr)$ in the case of rank-one matrices.

3.2.1 How the gradient can preserve zeros in the iterates

We now wish to draw attention to a characteristic of the gradient which can potentially have undesirable effects for the augmented Lagrangian algorithm but can be handled so as to cause little trouble. The problem arises from the following observation: the pattern of zeros in R and $\tilde{S}$ can propagate a pattern of zeros in the gradient $\mathcal{L}(R, y, \sigma) = 2\,\tilde{S}R$, which can in turn propagate a pattern of zeros in points $\tilde{R}$ obtained by performing a gradient-based line search from R.

For example, suppose the augmented Lagrangian algorithm for (N_r) is initialized with a point $R \in \Re^{n\times r}$ having its final column equal to zero. Then the gradient $2\,\tilde{S}R$ will also have its final column equal to zero. Hence, any point $\tilde{R}$ obtained from R by performing a line search along the gradient will also have a zero final column. Continuing the algorithm in this way, it is not difficult to see that the limited memory BFGS algorithm, which uses combinations of gradients to compute its search directions, will also ensure that the final column of each ensuing iterate is zero. Thus, though the algorithm was meant to solve (N_r), it is in fact solving (N_{r-1}).

The net effect of this zero-preserving characteristic of the gradient is that we may be imposing an unintended structure on the iterates, which can in turn mean that we are solving a restriction of (N_r). A practical way to alleviate this problem is simply to initialize the algorithm with an R which has no zeros at all.

4 Computational Results

In this section, we describe our computational experiences with the low-rank augmented Lagrangian approach. For comparison, we also present computational results from the spectral bundle method of Helmberg and Rendl [12] as well as the dual-scaling interior-point method of Benson, Ye, and Zhang [3]. The implementation of our method was written in ANSI C, and all experiments for each code were performed on an SGI Origin2000 with 16 300MHz R12000 processors and 10 Gigabytes of RAM. We stress, however, that none of the three codes is parallel, that is, each code uses only one processor.

4.1 The Lovász theta SDP

The first set of computational results that we present are for the Lovász theta SDP, which was introduced by L. Lovász in the seminal paper [15]. Given a simple, undirected graph $G = (V, E)$, the Lovász theta number ϑ of G is defined as the negative of the optimal value of the Lovász theta SDP

$$\min\left\{-(ee^T)\bullet X : \mathrm{trace}(X) = 1,\ X_{ij} = 0\ \forall\ (i,j) \in E,\ X \succeq 0\right\}, \tag{16}$$

where $e \in \Re^{|V|}$ is the vector of all ones and X has dimension $|V| \times |V|$. (Note that the Lovász theta SDP is usually stated as a maximization, implying that ϑ is simply the optimal value, but we have chosen to state the SDP as a minimization in standard form.) One of the most interesting properties of ϑ is that it satisfies the inequality $\alpha = \omega(\bar{G}) \le \vartheta \le \chi(\bar{G})$, where α, $\omega(\bar{G})$, and $\chi(\bar{G})$ are the stability number of G, the maximum clique number of the complement graph $\bar{G}$, and the chromatic number of $\bar{G}$, respectively. In this regard, ϑ is a polynomial-time computable number which lies between two numbers that are NP-Hard to compute.

In terms of the semidefinite program (1), $n = |V|$, $m = |E| + 1$, $C = -ee^T$, the set $\{A_i\}_{i=1}^m$ consists of the identity matrix I as well as $|E|$ matrices of the form $e_ie_j^T + e_je_i^T$

(where $e_k \in \Re^n$ has a 1 in position k and 0 elsewhere), and the vector $b \in \Re^m$ has one entry equal to 1 and all other $|E|$ entries 0. In the computations, we will treat C as a rank-one matrix while all other data matrices we will be handled as arbitrary, sparse matrices. Hence, we can tailor Propositions 3.1–3.3 to the case at hand, obtaining the following proposition.

Proposition 4.1 *For the Lovász theta SDP, the function and gradient evaluations of the augmented Lagrangian $\mathcal{L}(R, y, \sigma)$ of (N_r) can be performed in $\mathcal{O}(|E|r + nr)$ and $\mathcal{O}(|E|r + nr + n^2)$ flops, respectively.*

To optimize the Lovász SDP, we apply the augmented Lagrangian algorithm as stated in Section 3 to the problem $(N_{\bar{r}})$, where $\bar{r}$ is defined by (5). (In this case, the approximate value of $\bar{r}$ is $\sqrt{2(|E| + 1)}$.) In particular, we do not employ the idea of dynamically increasing the rank as described at the end of Section 2. There are two reasons for this. Firstly, we cannot guarantee that the regularity assumption of Proposition 2.5 holds at an arbitrary stationary point R of (N_r) obtained by the algorithm, and as such, the theoretical base for dynamically changing the rank is not strong. Secondly, our computational experience shows that the problem (N_r) corresponding to the Lovász SDP is difficult to solve for small values of r. In particular, the convergence of our method on problems with $r \ll \bar{r}$ is not very strong, while the convergence for $r = \bar{r}$ is quite good. As a result, we find computationally that the potential benefits of using (N_r) for small r are not realized due to slow convergence.

In addition, we have chosen not to test optimality by checking the positive semidefiniteness of S during the algorithm. The reason for this is two-fold: firstly, we desired to test our own stopping criterion (described below); and secondly, an efficient routine for testing positive semidefiniteness was not readily available. In particular, we remark that the Cholesky factorization, a typical way to test $S \succeq 0$, can still require a large amount of computational work if the nonzero fill-in of the Cholesky factor is large.

In our numerical tests for the Lovász theta SDP, we found it very important to monitor the increase of the penalty parameter σ. In early testing, we found that σ had a tendency to increase to a high level which resulted in ill-conditioning of the augmented Lagrangian function and consequently poor overall performance of the algorithm. A related observation was that the progress of the Lagrange multipliers y towards the optimal y^* was extremely important for the accuracy of the method. In fact, we found it necessary to allow the updating of y as often as possible. So, in order to deemphasize the penalty parameter while emphasizing the multipliers, we chose the penalty update factor γ to be $\sqrt{10}$ (as opposed to the usual value of 10), and we dynamically updated the infeasibility reduction parameter η so as to allow the updating of y while still requiring a moderate amount of reduction in the infeasibility.

A fundamental property of (16) is that its optimal value lies between $-n$ and -1. With this in mind, we would like to initialize the augmented Lagrangian algorithm with a specific (R, y) so that $X = RR^T$ is primal-feasible with objective value -1 and so that (S, y) is dual-feasible with objective value $-n$. For the primal, the matrix $R \in \Re^{n \times \bar{r}}$ having $1/\sqrt{\bar{r}}$ in each diagonal position and zeros elsewhere would certainly suffice, but this matrix is not an appropriate choice since it has a large pattern of zeros that can cause problems for the algorithm according to Section 3.2.1. So we alter the suggested matrix by perturbing it by a dense matrix of small norm. In particular, the initial $R^0 \in \Re^{n \times \bar{r}}$ that we choose is defined by

$$R^0_{ij} = \begin{cases} (1/\bar{r})^{1/2} + (1/nm)^{1/2} & \text{if } i = j, \\ (1/nm)^{1/2} & \text{otherwise.} \end{cases}$$

For the initial choice of (S, y), we first note that the dual SDP of (16) can be written as follows:

$$\max \left\{ y_0 : S = -ee^T - y_0 I - \sum_{(i,j)\in E} y_{ij}(e_i e_j^T + e_j e_i^T),\ S \succeq 0 \right\}.$$

Here, we have departed from our usual notation of denoting the Lagrange multipliers by a vector $y \in \Re^m$. Instead, we let y_0 denote the multiplier for the trace constraint and y_{ij} denote the multiplier for the constraint on edge (i, j). We choose the components $y^0_0 = -n$ and $y^0_{ij} = -1$ to comprise our initial Lagrange multiplier y^0, and S^0 is then given by (8). From this it can be seen that the objective value at (S^0, y^0) is $-n$, and moreover, if we make the assumption that G has no isolated nodes (a reasonable assumption in the context of computing the Lovász theta number of graph), then it is easy to see that S^0 is diagonally dominant with each diagonal entry equal to $n-1$ and each off-diagonal entry equal to either -1 or 0. Hence S^0 is positive definite, and so (S^0, y^0) constitutes a feasible solution of the dual SDP having objective value $-n$, as desired.

As stated in Section 3, the augmented Lagrangian algorithm does not have a stopping criterion. In the computational results below, the stopping criterion we have chosen is as follows: the algorithm is terminated once points R^* and (y^*, σ_*) are obtained such that

$$\frac{|\mathcal{L}(R^*, y^*, \sigma_*) - C \bullet (R^*(R^*)^T)|}{\max(|L(R^*, y^*)|, 1)} < 10^{-5}.$$

In other words, the algorithm stops when the relative difference between the augmented Lagrangian function and the regular objective function is sufficiently small.

In the following discussion, the low-rank augmented Lagrangian algorithm will be referred to as LR, and we will compare this method with two other methods—the spectral bundle method (SBmethod version 1.1.1) of Helmberg, Rendl and Kiwiel and the dual-scaling interior-point method (DSDP version 3.2) of Benson, Ye, and Zhang. The spectral bundle method, or simply "SB," is a dual ascent method, that is, it maximizes the dual of (16) by computing a sequence of feasible points whose objective values monotonically approach the optimal value. The dual-scaling algorithm, or simply "DSDP," on the other hand, maintains both primal and dual feasibility and obtains optimality by forcing the primal and dual objective values to converge to one another. Both SB and DSDP are run with their default parameters.

Firstly, we compare the four methods on a set of six graphs which were used in the Seventh DIMACS Implementation Challenge on Semidefinite and Related Optimization Problems [1]; their characteristics are listed in Table 1. In the table, we give the graph name, the number of vertices, the number of edges, and the edge density of the graph, which is also the nonzero density of the dual matrix S. In addition, the last column of Table 1 gives the value of $\bar{r}$ for each graph.

In Table 2, we list the objective values obtained by each method on each of the six problems. Note that there are two columns for DSDP: DSDP-p gives DSDP's primal objective

Table 1: The Hamming Theta Graphs

graph	$\lvert V\rvert$	$\lvert E\rvert$	dens %	rank $\bar{r}$
hamming-9-8	120	1680	23.53	58
hamming-10-2	70	560	23.19	34
hamming-11-2	171	5100	35.09	102
hamming-7-5-6	300	33917	75.62	261
hamming-8-3-4	200	5970	30.00	110
hamming-9-5-6	200	6032	30.31	110

Table 2: Objective Values for the Hamming Theta Graphs

graph	LR	SB	DSDP-p	DSDP-d
hamming-9-8	−224.000	−224.000	−223.795	−224.005
hamming-10-2	−102.400	−102.400	n/a	n/a
hamming-11-2	−170.665	−170.667	n/a	n/a
hamming-7-5-6	−42.667	−42.667	−42.642	−42.669
hamming-8-3-4	−25.600	−25.600	n/a	n/a
hamming-9-5-6	−85.333	−85.333	n/a	n/a

value, and DSDP-d gives its dual objective value. Note also that DSDP was unable to run efficiently on four of the six graphs, and so those results are not available as indicated by the symbol "n/a." Table 3 helps to interpret the objective values by providing estimates of the accuracy of each method. The first column gives the Euclidean norm of the final infeasibility obtained by the augmented Lagrangian algorithm and in particular shows that LR was able to obtain points that were very nearly feasible. The second column compares the objective values found by LR and SB in a relative sense, i.e., the absolute value of the difference of the two numbers is divided by the larger (in absolute value) of the two numbers. Since LR is a primal method while SB is a dual method, the numbers indicate that each of the two methods computed a highly accurate optimal value. Finally, the last column compares DSDP-p with DSDP-d in a similar way, and the numbers in this column indicate that DSDP also obtained an accurate optimal value.

Finally, in Table 4, we give the times (in seconds) taken by each method on each problem, and in the final row, we give the total time for each method. The table shows that SB outperformed both LR and DSDP on this class of graphs. It should be noted, however, that these "hamming" graphs have a particular structure that allows SB to solve each of them in one iteration. This type of performance is actually quite atypical as demonstrated by Tables 5 through 7 (see next paragraph).

In Tables 5 through 7, we give computational results for LR and SB on an additional set of 25 graphs that come from two sources: the so-called Gset collection of graphs introduced by Helmberg and Rendl in [12] and the Second DIMACS Implementation Challenge on the maximum clique (or maximum stable set) problem [14]. The data in the tables is organized

Table 3: Accuracies for the Hamming Theta Graphs

graph	LR feas	LR-SB	DSDP
hamming-9-8	4.8e-05	4.0e-07	9.4e-04
hamming-10-2	4.3e-05	2.7e-06	n/a
hamming-11-2	1.7e-04	1.1e-05	n/a
hamming-7-5-6	2.1e-05	2.3e-08	6.3e-04
hamming-8-3-4	4.9e-05	3.1e-07	n/a
hamming-9-5-6	4.1e-05	4.7e-07	n/a

Table 4: Times in Seconds for the Hamming Theta Graphs

Note: for each graph, SB took an atypical one iteration.

graph	LR	SB	DSDP
hamming-9-8	17.0	0.8	250.5
hamming-10-2	978.4	50.7	n/a
hamming-11-2	3420.2	159.2	n/a
hamming-7-5-6	3.5	0.3	59.3
hamming-8-3-4	223.2	9.9	n/a
hamming-9-5-6	454.6	4.9	n/a
totals	5096.9	225.8	n/a

Table 5: The Gset and Second DIMACS Theta Graphs

graph	$\lvert V\rvert$	$\lvert E\rvert$	dens %	rank $\bar{r}$
G43	1000	9990	2.00	142
G44	1000	9990	2.00	142
G45	1000	9990	2.00	142
G46	1000	9990	2.00	142
G47	1000	9990	2.00	142
G48	3000	6000	0.13	110
G49	3000	6000	0.13	110
G50	3000	6000	0.13	110
G51	1000	5909	1.18	109
G52	1000	5916	1.18	109
G53	1000	5914	1.18	109
G54	1000	5916	1.18	109
MANN-a27.co	378	702	0.99	38
brock200-1.co	200	5066	25.46	101
brock200-4.co	200	6811	34.23	117
brock400-1.co	400	20077	25.16	201
c-fat200-1.co	200	18366	92.29	192
hamming6-4.co	64	1312	65.08	52
hamming8-4.co	256	11776	36.08	154
johnson16-2-4.co	1024	23040	4.40	215
johnson8-4-4.co	2048	56320	2.69	336
keller4.co	128	1792	22.05	60
p-hat300-1.co	256	16128	49.41	180
san200-0.7-1.co	512	53760	41.10	328
sanr200-0.7.co	512	2304	1.76	68

in a similar way as for Tables 1–4. Regarding Table 7, note that each method was given an upper limit of ten hours (or 36,000 seconds) for computation time on a single instance. The tables support the following conclusion: each method is accurate but LR outperforms SB by a large margin in terms of time.

4.2 The maximum cut SDP relaxation

The maximum cut problem on a simple, undirected, edge-weighted graph $G = (V, E, W)$ is the problem of partitioning the vertices into two sets V_1 and V_2 so that the total weight of all edges crossing between V_1 and V_2 is maximized. The maximum cut problem is often referred to simply as "maxcut." We assume that $V = \{1, \ldots, n\}$ and that W is an $n \times n$ symmetric matrix such that entry w_{ij} is the weight on edge (i, j), where $w_{ij} = 0$ if $(i, j) \notin E$. The following SDP relaxation of the maxcut problem was introduced by Goemans and

Table 6: Objective Values and Accuracies for the Gset and Second DIMACS Theta Graphs

graph	LR	SB	LR feas	LR-SB
G43	−280.576	−280.629	3.1e-05	1.9e-04
G44	−280.485	−280.588	6.5e-06	3.7e-04
G45	−280.142	−280.190	1.7e-06	1.7e-04
G46	−279.782	−279.843	1.2e-04	2.2e-04
G47	−281.858	−281.899	8.5e-07	1.5e-04
G48	−1499.973	−1500.000	4.1e-05	1.8e-05
G49	−1499.982	−1500.000	9.3e-06	1.2e-05
G50	−1494.050	−1497.037	2.5e-05	2.0e-03
G51	−349.000	−349.023	7.2e-06	6.7e-05
G52	−348.387	−348.515	1.8e-06	3.7e-04
G53	−348.348	−348.386	2.1e-06	1.1e-04
G54	−341.000	−341.014	6.0e-06	4.0e-05
MANN-a27.co	−132.753	−132.764	2.9e-05	8.8e-05
brock200-1.co	−27.454	−27.459	8.0e-05	1.6e-04
brock200-4.co	−21.290	−21.296	1.5e-04	2.5e-04
brock400-1.co	−39.652	−39.710	1.4e-04	1.5e-03
c-fat200-1.co	−12.000	−12.003	1.4e-05	2.8e-04
hamming6-4.co	−5.333	−5.333	4.0e-05	2.7e-05
hamming8-4.co	−16.000	−16.001	2.6e-04	8.5e-05
johnson16-2-4.co	−8.000	−8.000	8.7e-05	3.3e-06
johnson8-4-4.co	−14.000	−14.000	3.6e-05	7.1e-08
keller4.co	−14.005	−14.013	2.3e-05	6.3e-04
p-hat300-1.co	−10.068	−10.109	1.3e-05	4.1e-03
san200-0.7-1.co	−30.000	−30.000	1.3e-05	4.4e-06
sanr200-0.7.co	−23.810	−23.838	5.4e-05	1.2e-03

Table 7: Times in Seconds for the Gset and Second DIMACS Theta Graphs

graph	LR	SB
G43	980.1	36000.0
G44	961.3	36000.1
G45	899.8	36000.6
G46	917.6	36000.3
G47	884.1	36000.1
G48	338.9	1.6
G49	405.7	1.6
G50	4255.4	3.2
G51	3125.1	5682.6
G52	7023.2	3399.7
G53	7530.6	27718.2
G54	2402.0	1313.4
MANN-a27.co	30.7	1689.4
brock200-1.co	259.1	36000.2
brock200-4.co	791.6	36000.2
brock400-1.co	2027.6	36000.2
c-fat200-1.co	741.7	36000.2
hamming6-4.co	1.2	7589.6
hamming8-4.co	73.2	36000.0
johnson16-2-4.co	3.6	3.0
johnson8-4-4.co	1.7	0.2
keller4.co	153.2	36000.0
p-hat300-1.co	8052.4	36000.4
san200-0.7-1.co	17.4	34.6
sanr200-0.7.co	396.1	36000.2
totals	42273.3	515439.6

Williamson in [11]:

$$\min \left\{ \frac{1}{4} [W - \mathrm{Diag}(We)] \bullet X : \mathrm{diag}(X) = e,\ X \succeq 0 \right\}. \tag{17}$$

Here, $e \in \Re^n$ is the vector of all ones. (Note also that we state the maxcut SDP relaxation as a minimization in accordance with (1) as opposed to a maximization as in [11].) In terms of the SDP (1), $n = |V|$, $m = n$, $C = \frac{1}{4}[W - \mathrm{Diag}(We)]$, and $A_i = e_i e_i^T$ for $i = 1, \ldots, n$, where $e_i \in \Re^n$ is the vector with a 1 in position i and 0 elsewhere. Notice that the sparsity pattern of C is exactly that of the graph G. For the full details on the maxcut SDP relaxation, we refer the reader to [11].

In terms of the nonlinear reformulation (N_r) of the maxcut SDP relaxation (17), the constraint $\mathrm{diag}(X) = e$ becomes the following: $\|R_{i\cdot}\|^2 = 1$ for all $i = 1, \ldots, m$. In words this means that a point R is feasible for (N_r) if and only if each row of R has norm equal to one. Since these "norm-one" constraints are separable, it is easy to see that (N_r) can be reformulated as an unconstrained problem having objective function

$$f(R) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \frac{\langle R_{i\cdot}, R_{j\cdot} \rangle}{\|R_{i\cdot}\| \, \|R_{j\cdot}\|} = \sum_{(i,j) \in E} c_{ij} \frac{\langle R_{i\cdot}, R_{j\cdot} \rangle}{\|R_{i\cdot}\| \, \|R_{j\cdot}\|} \tag{18}$$

Because of the simplicity of handling the constraints, we choose to optimize (N_r) using this unconstrained formulation rather than using the augmented Lagrangian algorithm of Section 3. The advantage is that, by applying the limited memory BFGS algorithm to this unconstrained problem, we obtain a feasible descent method that can still exploit the sparsity of C (as is evident by the final expression in (18)).

In our implementation of the maxcut SDP relaxation, we are able to test the idea of dynamically increasing the rank r as discussed at the end of Section 2. In the specific case of the maxcut SDP relaxation, it is easy to see that the constraint gradients $\{2\, e_i e_i^T R\}_{i=1}^m$ are linearly independent for all $R \in \Re^{n \times r}$, and so the linear independence assumption of Proposition 2.5 will always be satisfied. Moreover, if we obtain a local minimum of the unconstrained problem (18), this will clearly give us a corresponding local minimum of (N_r) which then satisfies the hypotheses of Proposition 2.5. Hence, the maxcut SDP relaxation is an ideal case for testing the idea of dynamically increasing the rank.

Our procedure for testing the dynamic update of r is a slight modification of the ideas presented in Proposition 2.5. Recall that $\bar{r}$, which is given by (5), is the smallest rank r that theoretically guarantees the equivalence of (N_r) with the SDP. (In this case, $\bar{r}$ is approximately equal to $\sqrt{2|V|}$.) Our procedure then is to define $r_j = \lceil (j/5)\bar{r} \rceil$ for $j = 1, \ldots, 5$ and to solve (N_{r_1}) through (N_{r_5}) successively, terminating the process either when the difference of two successive optimal values is sufficiently small or when (N_{r_5}) itself has been solved. (We note that $r_5 = \bar{r}$.) The problem (N_{r_1}) is initialized with $R^0 \in \Re^{n \times r_1}$ that is defined by normalizing the rows of the matrix

$$\tilde{R}^0_{ij} = \begin{cases} 1 + (1/nm)^{1/2} & \text{if } i = j, \\ 1 + (1/nm)^{1/2} & \text{if } i > j \text{ and } j = r_1, \\ (1/nm)^{1/2} & \text{otherwise.} \end{cases}$$

As with the Lovász theta SDP, the motivation behind this choice in starting point is to obtain a reasonable initial objective value while avoiding a structured pattern of zeros in

Table 8: The Torus Maxcut Graphs

graph	$\lvert V\rvert$	$\lvert E\rvert$	dens %	rank $\bar{r}$
toruspm3-8-50	512	1536	1.17	33
toruspm3-15-50	3375	10125	0.18	83
torusg3-8	512	1536	1.17	33
torusg3-15	3375	10125	0.18	83

the initial point. In a similar manner, once (N_{r_1}) has been solved and a local minimum R^1 obtained, the initial point for (N_{r_2}) is computed by appending $r_1 - r_2$ columns of small norm to R^1 and then normalizing. Initial points for (N_{r_3}), (N_{r_4}), and (N_{r_5}) can be calculated similarly (if necessary).

In our tests, a problem (N_{r_j}) is considered solved once a point R^* satisfying

$$\frac{\|\nabla f_j(R^*)\|}{\max(|f_j(R^*)|, 1)} < 10^{-5},$$

is obtained, where f_j is the objective function (18) in the space $\Re^{n \times r_j}$. In addition, we consider the difference between two successive optimal values f_j^* and f_{j+1}^* to be "sufficiently small" if

$$\frac{f_j^* - f_{j+1}^*}{\max(|f_{j+1}^*|, 1)} < 10^{-5}.$$

For comparison with the idea of dynamic rank, we also test solving $(N_{\bar{r}})$ directly. The initial point chosen for this computation is chosen similarly as for (N_{r_1}) above, and the overall stopping criterion is also as above, namely that the algorithm is terminated once a point R^* is obtained whose relative gradient norm is less than 10^{-5}.

In the following discussion, the two methods described above will be referred to as LR5 and LR1, respectively. We will compare these two methods with the same two methods used for comparison in the previous subsection—the spectral bundle method SB and the dual-scaling interior-point method DSDP. As stated previously, SB a dual ascent method, while DSDP maintains both primal and dual feasibility.

We first compare the four methods on a set of four graphs which were used in the Seventh DIMACS Implementation Challenge on Semidefinite and Related Optimization Problems [1]. They are the so called "torus" problems and their characteristics are listed in Table 8. (The format of Table 8 follows that of Table 1.)

Tables 9 and 10 provide objective values and accuracies for the four methods. The last three columns of Table 10 are similar to the last two columns of Table 3—each compares a primal method with a dual method. The first column, on the other hand, compares the two primal versions LR5 and LR1 (using the scaled difference between the two numbers). As a result, the first column indicates that the values found by LR5 and LR1 do not differ significantly, which indicates that both LR5 and LR1 converged to the same value as predicted by theory. Overall, the conclusion of Tables 9 and 10 is that each method was able to compute accurate solutions.

Table 9: Objective Values for the Torus Maxcut Graphs

graph	LR5	LR1	SB	DSDP-p	DSDP-d
toruspm3-8-50	−527.807	−527.805	−527.813	−527.509	−527.813
toruspm3-15-50	−3475.072	−3475.064	−3475.159	−3474.523	−3475.136
torusg3-8	−457.358	−457.357	−457.361	−456.932	−457.369
torusg3-15	−3134.517	−3134.443	−3134.592	−3132.688	−3134.572

Table 10: Accuracies for the Torus Maxcut Graphs

graph	LR5-LR1	LR5-SB	LR1-SB	DSDP
toruspm3-8-50	3.5e-06	1.1e-05	1.4e-05	5.8e-04
toruspm3-15-50	2.3e-06	2.5e-05	2.7e-05	1.8e-04
torusg3-8	2.3e-06	7.4e-06	9.7e-06	9.6e-04
torusg3-15	2.4e-05	2.4e-05	4.8e-05	6.0e-04

Finally, in Table 11, we give the times taken by each method. The table shows that LR5 outperforms all other methods. In particular, we can conclude that the idea of dynamically increasing the rank works well since LR5 outperforms LR1. In addition, we see that both of the low-rank methods outperform SB and DSDP and that DSDP is an order of magnitude slower than SB.

In Tables 12 through 14, we give computational results for LR5, LR1, and SB on an additional set of 25 graphs that come from the Gset collection of graphs mentioned in the previous subsection. The data in the tables is organized in a similar way as for Tables 8–11, and the tables support the same conclusions as above, namely that each method is accurate but that LR5 outperforms LR1 which outperforms SB. In particular, one can see that LR5 and LR1 have much stronger performance on the largest graphs. For example, in the case of G81, a graph having 20,000 vertices and 40,000 edges, LR5 is almost 500 times faster than SB.

Table 11: Times in Seconds for the Torus Maxcut Graphs

graph	LR5	LR1	SB	DSDP
toruspm3-8-50	2.7	5.7	9.9	13.1
toruspm3-15-50	25.7	154.4	288.2	2311.8
torusg3-8	3.3	3.9	9.2	10.9
torusg3-15	54.8	119.2	391.7	3284.3
totals	86.5	283.2	699.1	5620.1

Table 12: The Gset Maxcut Graphs

graph	$\lvert V\rvert$	$\lvert E\rvert$	dens %	rank $\bar{r}$
G01	800	19176	6.00	41
G11	800	1600	0.50	41
G14	800	4694	1.47	41
G22	2000	19990	1.00	64
G32	2000	4000	0.20	64
G35	2000	11778	0.59	64
G36	2000	11766	0.59	64
G43	1000	9990	2.00	45
G48	3000	6000	0.13	78
G51	1000	5909	1.18	45
G52	1000	5916	1.18	45
G55	5000	12498	0.10	101
G57	5000	10000	0.08	101
G58	5000	29570	0.24	101
G60	7000	17148	0.07	119
G62	7000	14000	0.06	119
G63	7000	41459	0.17	119
G64	7000	41459	0.17	119
G65	8000	16000	0.05	127
G66	9000	18000	0.04	135
G67	10000	20000	0.04	142
G70	10000	9999	0.02	142
G72	10000	20000	0.04	142
G77	14000	28000	0.03	168
G81	20000	40000	0.02	201

Table 13: Objective Values and Accuracies for the Gset Maxcut Graphs

graph	LR5	LR1	SB	LR5-LR1	LR5-SB	LR1-SB
G01	−12082.937	−12083.134	−12083.273	1.6e-05	2.8e-05	1.2e-05
G11	−629.157	−629.160	−629.173	4.5e-06	2.5e-05	2.0e-05
G14	−3191.559	−3191.545	−3191.589	4.2e-06	9.6e-06	1.4e-05
G22	−14135.718	−14135.750	−14136.039	2.3e-06	2.3e-05	2.0e-05
G32	−1567.617	−1567.621	−1567.655	2.4e-06	2.4e-05	2.1e-05
G35	−8014.556	−8014.616	−8014.796	7.4e-06	3.0e-05	2.2e-05
G36	−8005.919	−8005.931	−8006.020	1.5e-06	1.3e-05	1.1e-05
G43	−7032.190	−7032.091	−7032.254	1.4e-05	9.1e-06	2.3e-05
G48	−5999.965	−5999.893	−6000.000	1.2e-05	5.9e-06	1.8e-05
G51	−4006.251	−4006.191	−4006.286	1.5e-05	8.8e-06	2.4e-05
G52	−4009.604	−4009.510	−4009.669	2.3e-05	1.6e-05	4.0e-05
G55	−11039.200	−11038.721	−11039.851	4.3e-05	5.9e-05	1.0e-04
G57	−3885.370	−3885.368	−3885.520	6.4e-07	3.9e-05	3.9e-05
G58	−20135.842	−20134.731	−20136.327	5.5e-05	2.4e-05	7.9e-05
G60	−15221.909	−15220.476	−15222.803	9.4e-05	5.9e-05	1.5e-04
G62	−5430.739	−5430.721	−5430.950	3.3e-06	3.9e-05	4.2e-05
G63	−28243.390	−28240.765	−28244.623	9.3e-05	4.4e-05	1.4e-04
G64	−10465.820	−10465.145	−10465.972	6.4e-05	1.5e-05	7.9e-05
G65	−6205.282	−6205.286	−6205.591	7.4e-07	5.0e-05	4.9e-05
G66	−7076.909	−7076.917	−7077.266	1.1e-06	5.1e-05	4.9e-05
G67	−7744.070	−7744.071	−7744.497	7.7e-08	5.5e-05	5.5e-05
G70	−9861.247	−9860.081	−9861.723	1.2e-04	4.8e-05	1.7e-04
G72	−7808.196	−7808.160	−7808.600	4.6e-06	5.2e-05	5.6e-05
G77	−11045.093	−11045.058	−11045.771	3.2e-06	6.1e-05	6.5e-05
G81	−15655.148	−15655.162	−15656.282	8.9e-07	7.2e-05	7.2e-05

Table 14: Times in Seconds for the Gset Maxcut Graphs

graph	LR5	LR1	SB
G01	11.9	15.2	21.0
G11	3.0	7.1	67.9
G14	11.1	10.5	31.5
G22	22.8	32.1	89.3
G32	9.7	31.8	286.0
G35	40.1	51.1	211.0
G36	61.7	65.8	250.4
G43	10.1	14.3	21.9
G48	17.1	49.0	0.3
G51	19.1	12.1	48.1
G52	13.8	11.6	50.6
G55	50.4	189.1	25356.4
G57	54.4	185.1	2438.1
G58	316.8	239.8	3023.1
G60	67.8	267.4	57132.8
G62	94.7	463.1	3582.1
G63	296.7	660.9	7561.3
G64	485.0	1140.7	5960.4
G65	114.4	577.2	9014.0
G66	152.7	795.4	9465.6
G67	194.1	653.4	16044.8
G70	304.8	439.1	88540.0
G72	195.1	645.2	10505.9
G77	287.4	1003.1	39523.6
G81	500.8	1644.2	245992.8
totals	3335.5	9204.3	525218.9

Table 15: The Minimum Bisection Graphs

graph	$\lvert V\rvert$	$\lvert E\rvert$	dens %	rank $\bar{r}$
bm1	882	4711	1.21	43
biomedP	6514	629839	2.97	115
industry2	12637	798219	1.00	159

Table 16: Objective Values for the Minimum Bisection Graphs

graph	LR5	LR1	SB	DSDP-p	DSDP-d
bm1	23.440	23.440	23.439	23.423	23.415
biomedP	33.600	33.602	33.599	n/a	n/a
industry2	65.631	65.646	64.398	n/a	n/a

4.3 The minimum bisection SDP relaxation

The minimum bisection problem on a simple, undirected, edge-weighted graph $G = (V, E, W)$ is similar to the maxcut problem except that the partition of vertices into V_1 and V_2 is required to satisfy $|V_1| = |V_2|$. In particular, the number of vertices $n = |V|$ must be even. The minimum bisection problem can be relaxed as

$$\min \left\{ \frac{1}{4} [\mathrm{Diag}(We) - W] \bullet X : \mathrm{diag}(X) = e,\ e^T X e = 0,\ X \succeq 0 \right\}, \tag{19}$$

where all scalars, vectors, and matrices are as in (17. In fact, the only difference between (19) and (17) is the negated objective function and the additional constraint $e^T X e = 0$.

To solve (19), we handle the constraint $\mathrm{diag}(X) = e$ as we handled it for (17), that is, we alter the objective function and thereby eliminate the constraint. We handle the additional constraint $e^T X e = 0$ using the augmented Lagrangian techniques of Section 3. In addition, we test the idea of dynamically changing the rank as with the maxcut SDP relaxation. (It can be easily checked that the regularity assumptions hold for the reformulation (N_r) of the minimum bisection SDP.) Choices of various parameters and stopping criteria are made similarly as was done for the Lovász theta SDP and the maxcut SDP relaxation.

Tables 15 through 18 are organized similarly as in previous subsection, and they show the performance of the four algorithms on a collection of three minimum bisection SDPs obtained from the Seventh DIMACS Implementation Challenge. As with the maxcut SDP, the results demonstrates that LR5 outperforms each of the other methods.

A few remarks concerning the tables are in order. First, DSDP was unable to perform satisfactorily on two of the three problems; this is indicated by the symbol "n/a." Second, LR5, LR1, and SB were each given an upper bound of ten hours running time on each instance. Third, it is important to note that this time limit affects the accuracies in Table 17 as it seems SB was not able to converge in the given time.

Table 17: Accuracies for the Minimum Bisection Graphs

graph	LR5 feas	LR1 feas	LR5-LR1	LR5-SB	LR1-SB	DSDP
bm1	3.7e-04	2.1e-04	0.0e+00	4.3e-05	4.3e-05	3.4e-04
biomedP	3.3e-04	3.3e-04	6.0e-05	3.0e-05	8.9e-05	n/a
industry2	6.4e-04	6.5e-04	2.3e-04	1.9e-02	1.9e-02	n/a

Table 18: Times in Seconds for the Minimum Bisection Graphs

graph	LR5	LR1	SB	DSDP
bm1	22.1	45.2	69.1	520.1
biomedP	3291.4	8548.5	18586.6	n/a
industry2	8291.9	25253.3	36046.6	n/a
totals	11605.4	33847.0	54702.3	n/a

5 Final Remarks

In this paper, we have introduced a new nonlinear algorithm for solving semidefinite programs in standard form. The algorithm combines several ideas, namely (i) the factorization of positive semidefinite matrices, (ii) the rank of optimal SDP solutions, and (iii) first-order nonlinear optimization algorithms. Each of these three ideas contributes to the success of the algorithm. In particular, item (i) allows us to eliminate the difficult constraint $X \succeq 0$; item (ii) allows us to greatly reduce the number of variables; and item (iii) allows us to take advantage of sparsity in the problem data.

Regarding optimality conditions for the low-rank nonlinear formulation (N_r), we have developed some interesting sufficient conditions and have shown how they can be incorporated into a practical algorithm for solving SDPs. In addition, the practical behavior of the augmented Lagrangian algorithm and its variants indicate the likelihood of convergence to a global optimal solution even though the underlying nonlinear program is nonconvex.

The algorithm proposed in this paper also compares very favorably with other efficient algorithms for solving SDPs. In particular, the low-rank approach outperformed both the spectral bundle method and the dual-scaling interior-point method, two of the most successful codes for solving large-scale SDPs. For the maxcut SDP, we feel that the performance of our algorithm LR5 is very strong, and we believe that LR5 will make the solution of maxcut SDPs for sparse graphs with tens of thousands of vertices a routine activity. The performance of our algorithm on the Lovász theta SDP is also very strong; to the best of our knowledge, the computational results in this paper represent the best progress made on solving the Lovász SDP to date.

There are many ways that we can try to improve our method. One of the most important areas for improvement is the solution of the augmented Lagrangian subproblems (as for the Lovász theta SDP and the minimum bisection SDP). Currently, the subproblems are being solved slowly due mainly to poor convergence. If this convergence could be improved, then the overall method would benefit greatly. Another area for improvement is the theoretical convergence of the augmented Lagrangian algorithm to an optimal SDP solution. Although we have some theoretical justification that explains the observed practical convergence, we do not currently have a formal convergence proof. One idea would be to combine our method with a dual approach that guarantees the dual feasibility of the pair (S, y) used in the algorithm. Finally, another way to improve our algorithm is to extend it to solve other classes of SDPs, for example those having inequality constraints.

Acknowledgements

The authors would like to express their gratitude to Professor Takashi Tsuchiya of the Tokyo Institute of Statistical Mathematics for hosting the authors in Japan where partial research on this project was conducted. In addition, the authors wish to thank Professor Yin Zhang of Rice University for making available the computer on which the computational results have been performed. This computer was acquired in part with support from NSF Grant DMS-9872009. Finally, the authors are in debt to Helmberg, Rendl, and Kiwiel as well as Benson, Ye, and Zhang for making their code freely available for comparison.

References

[1] Seventh DIMACS implementation challenge on semidefinite and related optimization problems, November 2000. See the website: `http://dimacs.rutgers.edu/Challenges/Seventh/Instances/`.

[2] A. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete Computational Geometry*, 13:189–202, 1995.

[3] S. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. Research Report, Department of Management Science, University of Iowa, Iowa, 1998. To appear in SIAM Journal on Optimization.

[4] S. Burer and R.D.C. Monteiro. A projected gradient algorithm for solving the maxcut SDP relaxation. manuscript, School of ISyE, Georgia Tech, Atlanta, GA, 30332, USA, December 1998. To appear in *Optimization Methods and Software.*

[5] S. Burer, R.D.C. Monteiro, and Y. Zhang. Interior-point algorithms for semidefinite programming based on a nonlinear programming formulation. manuscript, School of ISyE, Georgia Tech, Atlanta, GA, 30332, USA, December 1999. To appear in *Computational Optimization and Applications.*

[6] S. Burer, R.D.C. Monteiro, and Y. Zhang. Solving semidefinite programs via nonlinear programming. Part I: Transformations and derivatives. manuscript, School of ISyE, Georgia Tech, Atlanta, GA, 30332, USA, September 1999. Submitted to *Mathematical Programming.*

[7] S. Burer, R.D.C. Monteiro, and Y. Zhang. Solving semidefinite programs via nonlinear programming. Part II: Interior-point methods for a subclass of SDPs. manuscript,

School of ISyE, Georgia Tech, Atlanta, GA, 30332, USA, October 1999. Submitted to *Mathematical Programming*.

[8] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, second edition, 1987.

[9] M. Fukuda and M. Kojima. Interior-point methods for lagrangian dual of semidefinite programs. manuscript, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguru-ku, Tokyo 152, Japan, December 2000.

[10] M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM Journal on Optimization*, 11:647–674, 2001.

[11] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, pages 1115–1145, 42 (1995).

[12] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2000.

[13] S. Homer and M. Peinado. Design and performance of parallel and distributed approximation algorithms for maxcut. manuscript, Dept. of Computer Science and Center for Computational Science, Boston University, 111 Cummington Street, Boston, MA, 02215, USA, 1995.

[14] D. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. AMS, 1996.

[15] L. Lovász. On the Shannon Capacity of a graph. *IEEE Transactions of Information Theory*, IT-25(1):1–7, January 1979.

[16] R.D.C. Monteiro and M. Todd. Path-following methods for semidefinite programming. In R. Saigal, L. Vandenberghe, and H. Wolkowicz, editors, *Handbook of Semidefinite Programming*. Kluwer Academic Publishers, 2000.

[17] G. Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of Operations Research*, 23:339–358, 1998.

[18] R. Saigal, L. Vandenberghe, and H. Wolkowicz. *Handbook of Semidefinite Programming*. Kluwer Academic Publishers, 2000.