

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/2609903>

# An Efficient Algorithm for Solving the MAXCUT SDP Relaxation

Article · January 1999

Source: CiteSeer

---

CITATIONS

11

READS

101

2 authors:



Samuel Burer

University of Iowa

61 PUBLICATIONS 2,474 CITATIONS

[SEE PROFILE](#)



Renato D. C. Monteiro

Georgia Institute of Technology

122 PUBLICATIONS 5,038 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



ADMM and its variants [View project](#)

# An Efficient Algorithm for Solving the MAXCUT SDP Relaxation\*

Samuel Burer<sup>†</sup>

Renato D.C. Monteiro<sup>‡</sup>

December 22, 1998

## Abstract

In this paper we present a projected gradient algorithm for solving the semidefinite programming (SDP) relaxation of the maximum cut (MAXCUT) problem. Coupled with a randomized method, this gives a very efficient approximation algorithm for the MAXCUT problem. We report computational results comparing our method with two earlier successful methods on problems with dimension up to 3000.

**Keywords:** semidefinite program, maxcut, randomized algorithm, semidefinite relaxation, gradient projection method, approximation algorithms.

**AMS 1991 subject classification:** 65K05, 90C25, 90C27, 90C30, 90C90.

## 1 Introduction

In this paper we develop a specialized algorithm for solving the semidefinite programming (SDP) relaxation of the maximum cut (MAXCUT) problem. The MAXCUT problem has many applications, e.g. in VLSI design and statistical physics (see [2, 4, 5, 19, 21]). Several algorithms have been proposed to find either exact or approximate solutions to this problem. As for many combinatorial optimization problems, the MAXCUT problem can be formulated as a quadratic programming (QP) problem in binary (or  $\pm 1$ ) variables. The idea that these problems can be naturally relaxed to SDP problems was first observed in Lovász [16] and Shor [22] and has been used by several authors (e.g., see [1, 6, 14, 17, 18, 20, 23]). Goemans and Williamson [9] developed a randomized algorithm for the MAXCUT problem, based on solving its SDP relaxation, which provides an approximate solution guaranteed to be within a factor of 0.87856 of its optimal value. In practice, their algorithm has been observed to find solutions which are significantly closer to the MAXCUT optimal value.

The major effort in Goemans and Williamson's method lies in the solution of the MAXCUT SDP relaxation. A naive use of an algorithm designed for solving general SDP problems drastically limits the size of the problem that can be solved. Efficient algorithms for solving the MAXCUT SDP relaxation have recently been developed which take into account its special structure. One approach to solve these problems is with the use of interior-point methods (see [3, 7, 8, 12, 15]). Among

---

\*The work of these authors was based on research supported by the National Science Foundation under grants INT-9600343 and CCR-9700448.

<sup>†</sup>School of Mathematics, Georgia Tech, Atlanta, Georgia 30332, USA. (email: burer@math.gatech.edu).

<sup>‡</sup>School of ISyE, Georgia Tech, Atlanta, Georgia 30332, USA. (email: monteiro@isye.gatech.edu).

these implementations, the one by Benson et al. [3] based on a potential-reduction dual-scaling interior-point method is the most efficient and the best suited for taking advantage of the special structure of the MAXCUT SDP relaxation. In addition to interior-point methods, other nonlinear programming methods have recently been proposed to solve the MAXCUT SDP relaxation (see [11, 13]). The approach used in Helmburg and Rendl [11] consists of solving a certain partial Lagrangian dual problem, whose objective function is nondifferentiable, using the usual bundle method for convex programming. Even though their method effectively takes advantage of the special structure of the SDP problem, its performance depends more on the ‘conditioning’ of the instance rather than its dimension and/or sparsity. On the other hand, Homer and Peinado [13] use the change of variables  $X = VV^T$ ,  $V \in \Re^{n \times n}$ , where  $X$  is the primal matrix variable of the MAXCUT SDP relaxation, to transform it into a constrained nonlinear programming problem in the new variable  $V$ . Using the specific structure of the MAXCUT SDP relaxation, they reformulate the constrained problem as an unconstrained problem and use the standard steepest ascent method on the latter problem. Their method tends to perform a large number of iterations and to possess slow asymptotic convergence, but it has the advantage of having very cheap iterations and thus can quickly obtain feasible solutions lying within, say, 0.2% (in relative error) of the SDP optimal value.

In this paper, we propose a variant of Homer and Peinado’s method based on the constrained nonlinear programming reformulation of the MAXCUT SDP relaxation obtained by using the change of variable  $X = LL^T$ , where  $L$  is a lower triangular matrix (possibly having negative diagonal elements). Our computational experience with our method indicates that it has similar convergence properties as Homer and Peinado’s method. This, together with lower storage and computational requirements due to the triangular structure of  $L$ , makes our method substantially faster than their method.

Our paper is organized as follows. In Section 2, we describe the MAXCUT problem, its corresponding  $\pm 1$ -QP reformulation, its SDP relaxation and the two constrained nonlinear programming reformulations of this SDP. In Section 3, we describe our method from the point of view that it, as well as Homer and Peinado’s method, can be interpreted as a projected gradient method applied to a constrained nonlinear programming reformulation of the MAXCUT SDP relaxation. Even though our method can be derived in the same way as Homer and Peinado’s method, its interpretation as a projected gradient method gives it a more intuitive appeal which we believe makes it more suitable for generalization. In Section 4, we describe the basic steps of our method from a computational point of view and discuss how the lower triangular structure of  $L$  can be exploited to efficiently implement the steps of our algorithm. We provide an analysis of the computational complexity of each iteration of our method and observe that this complexity depends on the ordering of the vertices of the graph. We then propose a vertex reordering heuristic which improves the overall running time of the code. We also discuss how to efficiently implement the Armijo line search used in our method. In Section 5, we present computational results comparing our method with Benson et al.’s and Homer and Peinado’s methods. Our main conclusions are: i) our method is considerably faster than the two other methods when the goal is to obtain approximate solutions that are within 0.2% (in relative error) of the SDP optimal value; ii) our method as well as Homer and Peinado’s method exhibit slow asymptotic convergence and hence should not always be used to obtain highly accurate solutions; and iii) it requires less memory than the other two methods.

### 1.1 Notation and terminology

In this paper,  $\Re$ ,  $\Re^n$ , and  $\Re^{n \times n}$  denote the space of real numbers, real  $n$ -dimensional column vectors, and real  $n \times n$  matrices, respectively. By  $\mathcal{S}^n$  we denote the space of real  $n \times n$  symmetric matrices, and we define  $\mathcal{S}_+^n$  and  $\mathcal{S}_{++}^n$  to be the subsets of  $\mathcal{S}^n$  consisting of the positive semidefinite and positive definite matrices, respectively. We write  $A \succeq 0$  and  $A \succ 0$  to indicate that  $A \in \mathcal{S}_+^n$  and  $A \in \mathcal{S}_{++}^n$ , respectively. We let  $\text{tr}(A)$  denote the trace of a matrix  $A \in \Re^{n \times n}$ , namely the sum of the diagonal elements of  $A$ . Moreover, for  $A, B \in \Re^{n \times n}$ , we define  $A \bullet B \equiv \text{tr}(A^T B)$ .

We adopt the convention of denoting matrices by capital letters and matrix entries by lowercase letters with double subscripts. For example, a matrix  $A \in \Re^{n \times n}$  has entries  $a_{ij}$  for  $i, j = 1, \dots, n$ . In addition, we denote the rows of a matrix by lowercase letters with single subscripts. For example,  $A \in \Re^{n \times n}$  has rows  $a_i$  for  $i = 1, \dots, n$ . In this paper, we will often find it necessary to compute the dot product of two row vectors  $a_i$  and  $b_j$  which arise as rows of the matrices  $A$  and  $B$ . Instead of denoting this dot product as  $a_i b_j^T$ , we will denote it as  $\langle a_i, b_j \rangle$ .

## 2 The MAXCUT Problem and Its Relaxations

In this section, we give an integer quadratic formulation of the MAXCUT problem and describe some of its relaxations. The first relaxation, originally introduced by Goemans and Williamson, is an SDP problem, while the second, which is used as the basis of our improved algorithm, is a quadratic maximization problem over the set of real lower triangular matrices with unit-length rows.

Let  $G$  be an undirected, simple graph (i.e., a graph with no loops or parallel edges) with vertex set  $V = \{1, \dots, n\}$  and edge set  $E$  whose elements are unordered pairs of distinct vertices denoted by  $\{i, j\}$ . Let  $W = (w_{ij}) \in \mathcal{S}^n$  be a matrix of nonnegative weights such that  $w_{ij} = w_{ji} = 0$  whenever  $\{i, j\} \notin E$ . For  $S \subseteq V$ , the set  $\delta(S) = \{\{i, j\} \in E : i \in S, j \notin S\}$  is called the cut determined by  $S$ . (When  $S = \{i\}$  we denote  $\delta(S)$  simply by  $\delta(i)$ .) The maximum cut (MAXCUT) problem on  $G$  is to find  $S \subseteq V$  such that

$$w(\delta(S)) \equiv \sum_{\{i, j\} \in \delta(S)} w_{ij}$$

is maximized. We refer to  $w(\delta(S))$  as the weight of the cut  $\delta(S)$ .

The MAXCUT problem can be formulated as the integer quadratic program

$$\begin{aligned} & \text{maximize} && \frac{1}{2} \sum_{i < j} w_{ij} (1 - y_i y_j) \\ & \text{subject to} && y_i \in \{-1, 1\}, \quad i = 1, \dots, n. \end{aligned} \tag{Q}$$

For any feasible solution  $y = (y_1, \dots, y_n)$  of (Q), the set  $S = \{i \in V : y_i = 1\}$  defines a cut  $\delta(S)$  which has weight equal to the objective value of  $y$ . The key property of this formulation is that  $\frac{1}{2}(1 - y_i y_j)$  can take on only two values—either 0 or 1—allowing us to model within the objective function the appearance of an edge in a cut. It is interesting to note that, for any fixed unit vector  $u \in \Re^n$ , (Q) can be reformulated as the problem of finding the maximum of the set  $\{\frac{1}{2} \sum_{i < j} w_{ij} (1 - v_i^T v_j) : v_i \in \{-u, u\}, i = 1, \dots, n\}$  since the key property that  $\frac{1}{2}(1 - v_i^T v_j)$  is either

0 or 1 still holds. In fact, this leads to the following relaxation of  $(Q)$  introduced by Goemans and Williamson [9]:

$$\begin{aligned} & \text{maximize} && \frac{1}{2} \sum_{i < j} w_{ij} (1 - v_i^T v_j) \\ & \text{subject to} && v_i \in S_{n-1}, \quad i = 1, \dots, n, \end{aligned} \tag{P}$$

where  $S_{n-1}$  denotes the  $(n-1)$ -dimensional unit sphere in  $\Re^n$ . It is the primary result of Goemans and Williamson's paper that a solution of  $(P)$  used within a certain randomized procedure yields a cut with expected weight at least 0.87856 times the weight of a maximum cut. It is also worth mentioning that Homer and Peinado's method is based on the relaxation  $(P)$ .

Goemans and Williamson also showed how  $(P)$  can be recast as a semidefinite program. Given  $v_1, \dots, v_n \in S_{n-1}$ , if we let  $V$  denote the  $n \times n$  matrix whose  $i$ -th column is  $v_i$ , then  $X = V^T V$  is positive semidefinite with  $x_{ii} = 1$  for  $i = 1, \dots, n$ . Conversely, a positive semidefinite  $X$  with  $x_{ii} = 1$  for  $i = 1, \dots, n$  gives rise to unit vectors  $v_1, \dots, v_n$  via the decomposition  $X = V^T V$ ,  $V \in \Re^{n \times n}$ . (Such a decomposition exists for each  $X \succeq 0$ .) The SDP reformulation is thus

$$\begin{aligned} & \text{maximize} && \frac{1}{2} \sum_{i < j} w_{ij} (1 - x_{ij}) \\ & \text{subject to} && x_{ii} = 1, \quad i = 1, \dots, n, \\ & && X \succeq 0. \end{aligned}$$

By using the symmetry of  $X$  and  $W$  along with the fact that  $w_{ii} = 0$  for  $i = 1, \dots, n$ , the objective function  $\frac{1}{2} \sum_{i < j} w_{ij} (1 - x_{ij})$  can be rewritten as

$$\frac{1}{4} \sum_{i,j} w_{ij} (1 - x_{ij}) = \frac{1}{4} \sum_{i,j} w_{ij} - \frac{1}{4} W \bullet X.$$

So, if we let  $C$  denote  $-\frac{1}{4}W$  and  $d$  denote  $\frac{1}{4} \sum_{i,j} w_{ij}$ , then the above formulation can be rewritten as the following SDP problem:

$$\begin{aligned} & \text{maximize} && C \bullet X + d \\ & \text{subject to} && (e_i e_i^T) \bullet X = 1, \quad i = 1, \dots, n, \\ & && X \succeq 0, \end{aligned} \tag{SP}$$

where  $e_i$  denotes the  $i$ -th unit vector.

We now state the nonlinear programming reformulation of  $(SP)$  which is the basis of our algorithm for finding an approximate solution of the MAXCUT problem. Let  $\mathcal{L}^n$  denote the set of real lower triangular  $n \times n$  matrices, and let  $\mathcal{L}_{++}^n$  and  $\mathcal{L}_{++}^n$  denote the subsets of  $\mathcal{L}^n$  whose elements have nonnegative diagonal entries and positive diagonal entries, respectively. For every  $X \in \mathcal{S}_{++}^n$ , there exists a unique  $L \in \mathcal{L}_{++}^n$  such that  $X = LL^T$ , and  $L$  is called the Cholesky factor of  $X$ . In addition, for every  $X \in \mathcal{S}_+^n$ , there exists an  $L \in \mathcal{L}_+$  such that  $X = LL^T$ , though  $L$  is not necessarily unique.

This triangular decomposition of positive semidefinite matrices motivates the following reformulation of  $(SP)$ :

$$\begin{aligned} & \text{maximize} && C \bullet (LL^T) + d \\ & \text{subject to} && (e_i e_i^T) \bullet (LL^T) = 1, \quad i = 1, \dots, n, \\ & && L \in \mathcal{L}^n. \end{aligned} \tag{LP}$$

Notice that we have replaced the requirement that  $X$  be positive semidefinite with the condition that  $L$  be in  $\mathcal{L}^n$  rather than  $L$  be in  $\mathcal{L}_+^n$ . We prefer the reformulation with the condition that  $L$  be in  $\mathcal{L}^n$  since it avoids inequality constraints. In fact, limited computational testing has revealed that the method based on the reformulation  $(LP)$  is superior to a variant for solving the reformulation with the constraint  $L \in \mathcal{L}_+^n$ .

In the following sections, we will sometimes find it more useful to describe  $(LP)$  in terms of the rows of  $L$ . More precisely, if  $\ell_i$  is the  $i$ -th row of  $L$ , then  $(LP)$  can also be stated as

$$\begin{aligned} & \text{maximize} && \sum_{i=1}^n \sum_{j=1}^n c_{ij} \langle \ell_i, \ell_j \rangle + d \\ & \text{subject to} && \langle \ell_i, \ell_i \rangle = 1, \quad i = 1, \dots, n, \\ & && \ell_{i(i+1)} = \dots = \ell_{in} = 0, \quad i = 1, \dots, n. \end{aligned}$$

### 3 The Algorithm Based on the Lower Triangular Relaxation

In this section, we develop and discuss the projected gradient algorithm used to solve  $(LP)$ . Before giving the basic steps of the algorithm, however, we introduce a few definitions. First, we define  $\varphi : \mathcal{L}^n \rightarrow \mathbb{R}$  by  $\varphi(L) = C \bullet (LL^T) + d$ . Second, let  $\text{low} : \mathbb{R}^{n \times n} \rightarrow \mathcal{L}^n$  be the operator which maps  $A \in \mathbb{R}^{n \times n}$  into the matrix  $L \in \mathcal{L}^n$  such that  $\ell_{ij} = a_{ij}$  if  $i \geq j$ , and  $\ell_{ij} = 0$  if  $i < j$ . In addition, given a matrix  $L \in \mathcal{L}^n$  with rows  $\ell_1, \dots, \ell_n$ , we define the operator  $s : \mathcal{L}^n \rightarrow \mathcal{L}^n$  entry-wise as

$$[\mathcal{U}(L)]_{ij} = \frac{\ell_{ij}}{\|\ell_i\|},$$

i.e.,  $\mathcal{U}$  applied to  $L$  normalizes the rows of  $L$ .

Given a matrix  $L^k$  which is feasible for  $(LP)$ , the  $k$ -th iteration of the projected gradient algorithm consists of the following steps:

1. Compute the gradient  $\tilde{P}^k$  for the function  $\varphi$  at  $L^k$ .
2. Calculate  $P^k$ , the projection of  $\tilde{P}^k$  onto the tangent subspace obtained by linearizing the constraints  $(e_i e_i^T) \bullet (LL^T)$ ,  $i = 1, \dots, n$ , at  $L^k$ .
3. Choose a step-size  $\alpha_k > 0$  such that  $\varphi(\mathcal{U}(L^k + \alpha_k P^k)) > \varphi(L^k)$ .
4. Set  $L^{k+1} = \mathcal{U}(L^k + \alpha_k P^k)$ .

In the following paragraphs, we discuss the details of these steps.

In step 1 above, we compute the gradient  $\tilde{P}^k$  of the function  $\varphi(L) = C \bullet (LL^T) + d$  at the current iterate  $L^k$ . The formula for the gradient is

$$\tilde{P}^k = 2 \text{low}(CL^k).$$

This formula shows that the computation of the gradient amounts to a single matrix multiplication, and in the event that  $C$  is sparse, the gradient can be computed taking advantage of sparsity, thus speeding up the algorithm on large, sparse problems.

The gradient  $\tilde{P}^k$  is an ascent direction for  $\varphi$  at  $L^k$ , but moving along  $\tilde{P}^k$  does not maintain feasibility due to the curvature of the feasible region. (In fact, feasibility is lost by moving along

any direction.) So, as a compromise, we project the gradient onto the tangent subspace at the current iterate of the manifold defined by the feasibility constraints. We denote this projection by  $P^k$ . Linearizing the constraints, we see that  $P^k$  must satisfy

$$(e_i e_i^T) \bullet (P^k (L^k)^T) = 0, \quad i = 1, \dots, n.$$

This condition is easier to handle if we rewrite it in terms of the rows of  $P^k$  and  $L^k$ . If  $p_i^k$  denotes the  $i$ -th row of  $P^k$ , then the above condition is equivalent to

$$\langle p_i^k, \ell_i^k \rangle = 0, \quad i = 1, \dots, n,$$

i.e.,  $p_i^k$  must be orthogonal to  $\ell_i^k$ . Thus,  $p_i^k$  is obtained by projecting  $\tilde{p}_i^k$  onto the hyperplane whose normal is  $\ell_i^k$ , that is

$$p_i^k = \tilde{p}_i^k - \frac{\langle \tilde{p}_i^k, \ell_i^k \rangle}{\langle \ell_i^k, \ell_i^k \rangle} \ell_i^k = \tilde{p}_i^k - \langle \tilde{p}_i^k, \ell_i^k \rangle \ell_i^k, \quad i = 1, \dots, n, \quad (1)$$

where the second equality follows from the fact that  $\|\ell_i^k\| = 1$ .

When the projected gradient  $P^k$  is nonzero, then it is an ascent direction for the function  $\varphi(L)$  at  $L^k$ , that is  $\varphi(L^k + \alpha P^k) > \varphi(L^k)$  for all sufficiently small  $\alpha > 0$ , due to the fact that

$$\frac{d}{d\alpha} (\varphi(L^k + \alpha P^k)) \Big|_{\alpha=0} = \nabla \varphi(L^k) \bullet P^k = \tilde{P}^k \bullet P^k = \|P^k\|_F^2 > 0.$$

Using the fact that  $L^k$  has unit-length rows, one can easily verify that

$$\frac{d}{d\alpha} (\mathcal{U}(L^k + \alpha P^k)) \Big|_{\alpha=0} = P^k,$$

and hence that

$$\frac{d}{d\alpha} (\varphi(\mathcal{U}(L^k + \alpha P^k))) \Big|_{\alpha=0} = \nabla \varphi(L^k) \bullet P^k = \tilde{P}^k \bullet P^k = \|P^k\|_F^2 > 0. \quad (2)$$

This implies that  $P^k$  is also an ascent direction for  $\varphi(\mathcal{U}(L))$ , that is  $\varphi(\mathcal{U}(L^k + \alpha P^k)) > \varphi(L^k)$  for sufficiently small  $\alpha > 0$ .

When  $P^k = 0$ , the following simple result whose proof is left to the reader states that  $L^k$  is a stationary point of  $(LP)$ , that is there exists  $\lambda^k \in \Re^n$  such that

$$\text{low} (CL^k) = \text{low} \left( \sum_{i=1}^n \lambda_i^k (e_i e_i^T) L^k \right), \quad (3)$$

or equivalently,

$$\tilde{p}_i^k = \lambda_i^k \ell_i^k, \quad i = 1, \dots, n.$$

**Proposition 3.1**  $P^k = 0$  if and only if there exists  $\lambda^k \in \Re^n$  such that (3) holds, in which case

$$\lambda_i^k = \langle \tilde{p}_i^k, \ell_i^k \rangle, \quad i = 1, \dots, n.$$

Not every stationary point of  $(LP)$  is a global solution of it. The following proposition gives sufficient conditions for a stationary point of  $(LP)$  to be a global solution.

**Proposition 3.2** *Assume that  $(L^k, \lambda^k) \in \mathcal{L}^n \times \mathbb{R}^n$  satisfies (3) and define*

$$S^k \equiv \sum_{i=1}^n \lambda_i^k (e_i e_i^T) - C.$$

*If  $S^k \succeq 0$  then  $L^k$  is a global solution of  $(LP)$ .*

**Proof.** First observe that  $L^k$  is a global solution of  $(LP)$  if and only if  $X^k \equiv L^k(L^k)^T$  is an optimal solution of the semidefinite program  $(SP)$ . We will henceforth show that the latter condition holds. For this, it is enough to show that  $X^k S^k = 0$ , since then  $X^k$  and  $(\lambda^k, S^k)$  is a pair of primal and dual optimal solutions of  $(SP)$ . By (3), we have that  $\text{low}(S^k L^k) = 0$ , that is  $S^k L^k$  is a strictly upper triangular matrix. This implies that  $S^k X^k = (S^k L^k)(L^k)^T$  is also a strictly upper triangular matrix, and hence that  $X^k \bullet S^k = \text{tr}(S^k X^k) = 0$ . Using the fact that  $X^k \succeq 0$  and  $S^k \succeq 0$ , it is now easy to see that  $X^k S^k = 0$ . ■

After computing  $P^k$ , the algorithm selects a step-size  $\alpha_k > 0$  such that  $\varphi(\mathcal{U}(L^k + \alpha_k P^k))$  is sufficiently larger than  $\varphi(L^k)$ . A line search along  $P^k$  must be performed to find such  $\alpha_k$ , and for this, the algorithm uses the Armijo line search technique. Given constants  $\sigma \in (0, 1)$  and  $\bar{\alpha} > 0$ , the Armijo line search chooses  $\alpha_k$  as the largest scalar  $\alpha$  from the set  $\{\bar{\alpha}/2^j : j = 0, 1, 2, \dots\}$  satisfying

$$\varphi(\mathcal{U}(L^k + \alpha P^k)) - \varphi(L^k) \geq \sigma \alpha (\tilde{P}^k \bullet P^k) = \sigma \alpha \sum_{i=1}^n \langle p_i^k, \tilde{p}_i^k \rangle. \quad (4)$$

In view of (2), such an  $\alpha$  necessarily exists.

We are now ready to state the algorithm to solve  $(LP)$ .

#### Algorithm:

Let  $L^0$  be a feasible point of  $(LP)$  and let  $\bar{\alpha} > 0$  and  $\sigma \in (0, 1)$  be given.

**For**  $k = 0, 1, 2, \dots$

    Compute  $\tilde{P}^k = 2 \text{low}(C L^k)$ .

    Calculate  $P^k$  by the formula  $p_i^k = \tilde{p}_i^k - \langle \tilde{p}_i^k, \ell_i^k \rangle \ell_i^k$  for  $i = 1, \dots, n$ .

    Choose the step-size  $\alpha_k > 0$  using the Armijo rule described above.

    Set  $L^{k+1} = \mathcal{U}(L^k + \alpha_k P^k)$ .

In view of Proposition 3.1, one possible termination criterion that can be used in the above algorithm is the condition that  $\|P^k\|_F < \varepsilon$ , for some prespecified constant  $\varepsilon > 0$ . It is possible to show that every accumulation point of the sequence  $\{L^k\}$  generated by the above algorithm is a stationary point of  $(LP)$ . Clearly, there is no guarantee that such stationary points will be global solutions of  $(LP)$ , but since  $(LP)$  does not have any local solutions, the possibility that  $\{L^k\}$  has accumulation points which are not global solutions is unlikely. In fact, in our numerical experiments we have observed that  $\{L^k\}$  always converges to the solution set of  $(LP)$ .

## 4 Details of the Implementation

In this section, we provide the details of our implementation, including the procedures to compute function values and gradients of  $\varphi(L)$ , the overall computational complexity of a general iteration of our method, and the procedure for selecting the step-size.

### 4.1 Complexity per iteration

In this subsection we derive the overall complexity of an iteration of our method. We adopt the same convention as in Golub and Van Loan [10] for counting flops, that is a flop is a floating point operation (e.g., the inner product of two  $n$ -vectors involves  $2n$  flops).

Useful in the derivation of the complexities will be the quantity

$$m(G) \equiv \sum_{\{i,j\} \in E} \min\{i,j\}$$

defined on our input graph  $G$ . It is easy to see that, if  $G$  is connected, then  $m(G)$  is smallest when  $G$  is a star with center vertex 1. In this case,  $m(G) = n - 1$ . On most ‘random’ graphs, however,  $m(G)$  is on the order of  $n^2$  or higher. We adopt the convention of stating the final complexities derived below in the form  $\tau m(G) + \mathcal{O}(\cdot)$  for some scalar  $\tau > 0$ , even though the term inside the  $\mathcal{O}$  operator can sometimes be of comparable order to  $\tau m(G)$ .

The first basic step of an iteration of our method is the computation of the gradient  $\tilde{P}^k = 2 \text{low}(CL^k)$  of the function  $\varphi(L) = C \bullet (LL^T) + d$  at the point  $L^k$ . The last  $n - i$  components of its  $i$ -th row  $\tilde{p}_i^k$  are equal to zero since  $\tilde{P}^k \in \mathcal{L}^n$  and the first  $i$  components are equal to the first  $i$  components of

$$2c_i L^k = 2 \sum_{j=1}^n c_{ij} \ell_j^k = 2 \sum_{\{i,j\} \in \delta(i)} c_{ij} \ell_j^k, \quad (5)$$

where the second equality comes from the fact that  $c_{ij} = 0$  whenever  $\{i,j\} \notin E$ . Note that only the first  $\min\{i,j\}$  components of the term  $c_{ij} \ell_j^k$  contribute to the computation of the first  $i$  components of (5), since  $\ell_{jh}^k = 0$  for  $h > j$ . Hence, each of the pairs  $(i,j)$  and  $(j,i)$  with  $\{i,j\} \in E$  contributes exactly  $2 \min\{i,j\}$  flops in the computation of  $\tilde{P}^k$ . So the overall cost of computing the gradient is  $4m(G) + \mathcal{O}(n^2)$  flops.

The second basic step of an iteration of our method is the computation of the projected gradient  $P^k$  according to (1). An immediate verification reveals that its computation takes  $\mathcal{O}(n^2)$  flops.

The third basic step of an iteration of our method is the determination of the step-size according to the Armijo rule. We first derive the number of flops to compute the term  $\varphi(\mathcal{U}(L^k + \alpha P^k))$ , for a given scalar  $\alpha$ , which appears in the left hand side of (4). Indeed, let  $\tilde{L}$  denote  $L^k + \alpha P^k$ . Then

$$\varphi(\mathcal{U}(L^k + \alpha P^k)) = \sum_{i=1}^n \sum_{j=1}^n c_{ij} \frac{\langle \tilde{\ell}_i, \tilde{\ell}_j \rangle}{\|\tilde{\ell}_i\| \|\tilde{\ell}_j\|} = 2 \sum_{\{i,j\} \in E} c_{ij} \frac{\langle \tilde{\ell}_i, \tilde{\ell}_j \rangle}{\|\tilde{\ell}_i\| \|\tilde{\ell}_j\|}, \quad (6)$$

where  $\tilde{\ell}_i$  denotes the  $i$ -th row of  $\tilde{L}$ . Noting that each inner product  $\langle \tilde{\ell}_i, \tilde{\ell}_j \rangle$  can be computed in  $2 \min\{i,j\}$  flops and each norm  $\|\tilde{\ell}_j\|$  can be computed in  $\mathcal{O}(n)$  flops, we conclude that the overall

complexity to evaluate  $\varphi(\mathcal{U}(L^k + \alpha P^k))$  is  $2m(G) + \mathcal{O}(n^2)$  flops. Letting  $\mathcal{I}_k$  denote the number of trial step-sizes  $\alpha$  generated by the Armijo rule in the  $k$ -th iteration of our method and noting that the right hand side of (4) can be evaluated in  $\mathcal{O}(n^2)$  flops, we easily see that the Armijo rule can be carried out in  $2(\mathcal{I}_k + 1)m(G) + \mathcal{O}(\mathcal{I}_k n^2)$  flops. (The term  $\mathcal{I}_k + 1$  is the total number of times  $\varphi(\mathcal{U}(L^k + \alpha P^k))$  needs to be evaluated including for  $\alpha = 0$ .)

A clever implementation of the Armijo rule allows us to reduce its complexity to either  $2m(G) + \mathcal{O}(\mathcal{I}_k|E| + n^2)$  or  $4m(G) + \mathcal{O}(\mathcal{I}_k|E| + n^2)$  flops depending upon whether  $\alpha = \bar{\alpha}$  is accepted by the Armijo rule or not. In what follows we discuss how this can be accomplished. First we discuss how the terms  $\langle \tilde{\ell}_i, \tilde{\ell}_j \rangle$  with  $\{i, j\} \in E$  can be computed efficiently for different values of  $\alpha$ . We have

$$\begin{aligned}\langle \tilde{\ell}_i, \tilde{\ell}_j \rangle &= \langle \ell_i^k + \alpha p_i^k, \ell_j^k + \alpha p_j^k \rangle \\ &= \langle \ell_i^k, \ell_j^k \rangle + \alpha \left( \langle \ell_i^k, p_j^k \rangle + \langle p_i^k, \ell_j^k \rangle \right) + \alpha^2 \langle p_i^k, p_j^k \rangle.\end{aligned}\quad (7)$$

For  $\alpha = 0$ , this term reduces to  $\langle \ell_i^k, \ell_j^k \rangle$  which we may assume has already been computed in the previous iteration since

$$\langle \ell_i^k, \ell_j^k \rangle = \frac{\langle \ell_i^{k-1} + \alpha_{k-1} p_i^{k-1}, \ell_j^{k-1} + \alpha_{k-1} p_j^{k-1} \rangle}{\|\ell_i^{k-1} + \alpha_{k-1} p_i^{k-1}\| \|\ell_j^{k-1} + \alpha_{k-1} p_j^{k-1}\|}.$$

Hence evaluation of (7) for every  $\{i, j\} \in E$  at  $\alpha = 0$  is free. Note that once we evaluate (7) at  $\alpha = \bar{\alpha}$  and  $\langle p_i^k, p_j^k \rangle$  for every  $\{i, j\} \in E$ , then it is possible to determine the value of  $\langle \ell_i^k, p_j^k \rangle + \langle p_i^k, \ell_j^k \rangle$  for every  $\{i, j\} \in E$  in  $\mathcal{O}(|E|)$  flops. Hence the value of (7) for every  $\{i, j\} \in E$  can be computed for any other value of  $\alpha$  in  $\mathcal{O}(|E|)$  flops. Note also that  $\langle p_i^k, p_j^k \rangle$  does not need to be evaluated if  $\alpha = \bar{\alpha}$  is accepted by the Armijo rule. Hence the overall contribution of the computation of the terms  $\langle \tilde{\ell}_i, \tilde{\ell}_j^k \rangle$  with  $\{i, j\} \in E$  towards the complexity of the Armijo rule is either  $2m(G) + \mathcal{O}(\mathcal{I}_k|E|)$  or  $4m(G) + \mathcal{O}(\mathcal{I}_k|E|)$  flops depending upon whether  $\alpha = \bar{\alpha}$  is accepted or not.

We now discuss the contribution of the computation of the norms  $\|\tilde{\ell}_i\|$  that appear in (6). Letting  $i = j$  in (7) and using the fact that  $\ell_i^k$  is orthogonal to  $p_i^k$ , we have

$$\|\tilde{\ell}_i\|^2 = \langle \tilde{\ell}_i, \tilde{\ell}_i \rangle = \langle \ell_i^k, \ell_i^k \rangle + \alpha^2 \langle p_i^k, p_i^k \rangle = 1 + \alpha^2 \|p_i^k\|^2.$$

Hence, once the norms  $\|p_i^k\|$ ,  $i = 1, \dots, n$ , are obtained, computation of  $\|\tilde{\ell}_i\|$ ,  $i = 1, \dots, n$ , takes  $\mathcal{O}(n)$  for each  $\alpha$ . Hence, the overall contribution of these terms towards the complexity of the Armijo rule is  $\mathcal{O}(\mathcal{I}_k n + n^2)$ . Since all other operations to compute  $\varphi(\mathcal{U}(L^k + \alpha P^k))$  take  $\mathcal{O}(|E|)$  flops for each  $\alpha$  and since the term

$$\tilde{P}^k \bullet P^k = P^k \bullet P^k = \sum_{i=1}^n \|p_i^k\|^2$$

that appears in right hand side of (4) takes  $\mathcal{O}(n)$  flops to compute, the overall complexity stated for the Armijo line search follows. (Here we are adopting the convention that  $|E| \geq n$ .)

The fourth and last basic step of our algorithm is easily seen to take  $\mathcal{O}(n^2)$  flops. Hence, the overall complexity of the  $k$ -th iteration of our method is either  $6m(G) + \mathcal{O}(\mathcal{I}_k|E| + n^2)$  or  $8m(G) + \mathcal{O}(\mathcal{I}_k|E| + n^2)$  flops.

	$m(G)$	sec/iter	$m(G')$	sec/iter
G1	5116819	4.96	4371073	4.32
G11	633764	1.05	320800	0.80
G14	667919	1.20	529547	1.09
G22	13334305	16.56	10210289	13.36
G48	8823650	16.60	4503000	12.05

Table 1: Improvement found by the vertex reordering.

## 4.2 Complexity reduction by vertex reordering

From the previous subsection, we conclude that the computational complexity of the  $k$ -th iteration of our algorithm is  $O(m(G) + \mathcal{I}_k|E| + n^2)$  flops. Since  $m(G)$  clearly depends on the way the vertices of  $G$  are labeled, a natural question is: can the vertices of  $G$  be reordered to form a graph  $G'$  such that  $m(G') < m(G)$ , thus speeding up the running time? This leads to the optimization problem of minimizing  $\sum_{\{i,j\} \in E} \min\{\pi(i), \pi(j)\}$  over all permutations  $\pi : V \rightarrow V$  for the graph  $G$ . We let  $M(G)$  denote its optimal value.

Since we believe that the above optimization problem is NP-hard, in what follows we only propose a greedy heuristic to approximate  $M(G)$  for a graph  $G$ . The heuristic comes from the idea that a vertex with high degree should be labeled with a small number. Before stating the heuristic, we give a definition: if  $H$  is a graph with vertex  $v$ , then  $H \setminus v$  denotes the graph obtained by removing  $v$  and all edges incident to  $v$  from  $H$ . The reordering algorithm is as follows:

**Reorder:**

```

Set  $G_1 = G$ .
for  $k = 1, \dots, n$ 
    Let  $i$  be a maximum-degree vertex of  $G_k$ .
    Set  $\pi(i) = k$ .
    Set  $G_{k+1} = G_k \setminus i$ .
end
For  $i = 1, \dots, n$ , relabel the  $i$ -th vertex of  $G$  as vertex  $\pi(i)$  in the new graph.

```

Unfortunately, this greedy heuristic does not give an optimal solution in all cases. In fact, for graphs  $G$  which are already nicely ordered, the heuristic may find a reordered graph  $G'$  such that  $m(G') > m(G)$ . The greedy heuristic is fast, however, and in all our test problems, the ratio  $m(G')/m(G)$  was between 0.51 and 0.86. This improvement translates into a sizable decrease in the average time required for an iteration of our method as can be seen in Table 1.

## 5 Summary of Computational Results

In this section we present computational results comparing our method to two earlier methods to find approximate solutions to the MAXCUT problem based on solving its SDP relaxation. These are Benson et al.'s method [3] which solves the SDP relaxation using a potential-reduction dual scaling interior-point method and Homer and Peinado's method [13] which is equivalent to solving the relaxation  $(P)$  using a projected gradient method similar to ours.

We implemented our projected gradient algorithm to solve the MAXCUT SDP relaxation in ANSI C and ran all test problems on a Sparc 20 with 160 MB of RAM. In all our test problems, we chose the initial iterate  $L^0$  to be the  $n \times n$  identity matrix. We also chose the Armijo line search constant  $\sigma$  to be equal to 0.005, and our choice of  $\bar{\alpha}$  in each iteration was determined as follows: every ten iterations,  $\bar{\alpha}$  was set to 4; otherwise,  $\bar{\alpha}$  was set to 1.03 times the step-size used in the previous iteration. We found that this scheme for choosing  $\bar{\alpha}$  resulted in fewer and faster iterations than, say, setting  $\bar{\alpha}$  equal to 1 in every iteration.

We also implemented the randomized cut generation scheme of Goemans and Williamson. Once our projected gradient algorithm finds an (approximate) optimal solution  $L^*$  of  $(LP)$ , we generate a random unit vector  $u \in S_{n-1}$  from a uniform distribution over  $S_{n-1}$  and compute  $v = L^*u$ . We then form the set  $S \equiv \{i \in V : v_i \geq 0\}$ , which determines the random cut  $\delta(S)$  with expected weight at least 0.87856 times the weight of a maximum cut. In fact, we repeat this randomized procedure  $n$  times and save the best of the  $n$  corresponding cuts.

Our test problems come from the same set of problems that Helmberg and Rendl [11] and Benson et al. [3] used to test their own methods of solving the MAXCUT SDP relaxation. The problems in this set are random weighted graphs generated by a machine-independent graph generator, **rudy**, created by G. Rinaldi. We have selected problems from this set varying in size from  $n = 800$  to  $n = 3000$  and in edge density from 0.17% to 6.12%.

Tables 2 and 3 compare the performance of Benson et al.'s method, Homer and Peinado's method, and our method on 18 representative problems. For each of the 18 problems, we list in the first column the problem name, its dimension, its sparsity and an upper bound on the MAXCUT SDP optimal value. (The upper bounds for these problems were found by running Benson et al.'s method on each problem and terminating once a relative duality gap of  $10^{-6}$  had been reached. We chose the final dual objective value given by Benson et al.'s method as the upper bound.) We also give the the number of iterations and amount of time (in seconds) each of the three methods took to find a feasible solution of the MAXCUT SDP relaxation whose objective value is within 0.2% (in relative error) of the upper bound. For each of the three methods, we stopped after obtaining a solution whose objective value was within 0.2% of the upper bound, and used this solution to compute  $n$  random cuts, the best of which we report under the heading 'cval.' In Tables 2 and 3, we also repeat the same procedures for the accuracy 0.02%. The last column, labeled 'ctime,' gives the time taken by each method to compute  $n$  random cuts.

Note that, in contrast to Benson et al.'s method, the number of iterations performed by the other two methods increase drastically as the required relative error is reduced from 0.2% to 0.02%. Further reduction of the required relative error will make this type of behavior even more evident. This slow asymptotic convergence is not surprising in view of the first-order nature of Homer and Peinado's method and our method.

In Table 4, we give the memory usage (in MB) of the three methods on nine of the 18 problems presented in Tables 2 and 3. This table demonstrates that our method requires less memory than the other two methods.

Tables 2 and 3 compare our method to Benson et al.'s and Homer and Peinado's methods based on a stopping criteria which requires a good upper bound on the optimal value of the MAXCUT SDP relaxation. Such an upper bound is of course not available in general, and in Table 5 we present the results of our method on 27 problems when an alternate stopping criteria is used. The

		0.2%			0.02%			ctime
		iter	time	cval	iter	time	cval	
<b>G1</b> <b>800 6.12%</b> <b>12083.1975</b>	BYZ	16	1956	11440	18	2197	11439	164
	HP	57	710	11332	99	1231	11373	118
	BM	32	140	11370	89	383	11401	84
<b>G2</b> <b>800 6.12%</b> <b>12089.43</b>	BYZ	16	1975	11420	18	2218	11407	165
	HP	56	696	11355	98	1215	11396	117
	BM	33	145	11360	81	351	11377	84
<b>G11</b> <b>800 0.63%</b> <b>629.1652</b>	BYZ	14	268	528	18	344	528	11
	HP	73	172	528	583	1357	532	108
	BM	250	203	520	1715	1386	530	38
<b>G12</b> <b>800 0.62%</b> <b>623.8745</b>	BYZ	16	324	532	18	344	530	12
	HP	81	193	530	511	1211	536	108
	BM	221	181	522	1191	980	526	39
<b>G14</b> <b>800 1.59%</b> <b>3191.5675</b>	BYZ	23	779	2984	25	847	2985	35
	HP	103	435	2954	298	1255	2960	110
	BM	51	57	2940	181	201	2958	47
<b>G15</b> <b>800 1.58%</b> <b>3171.5575</b>	BYZ	27	881	2975	29	948	2977	44
	HP	109	456	2934	351	1465	2940	111
	BM	62	70	2937	181	200	2964	46

Table 2: Comparison of the three methods:  $n = 800$ .

		0.2%			0.02%			
		iter	time	cval	iter	time	cval	ctime
<b>G43</b> <b>1000 2.10%</b> <b>7032.2225</b>	BYZ	16	2720	6508	19	3234	6514	238
	HP	48	462	6449	81	777	6473	221
	BM	41	135	6435	127	408	6477	141
<b>G44</b> <b>1000 2.10%</b> <b>7027.885</b>	BYZ	16	2714	6505	18	3049	6506	217
	HP	48	463	6446	84	808	6466	222
	BM	41	132	6437	108	347	6466	139
<b>G51</b> <b>1000 1.28%</b> <b>4006.255</b>	BYZ	24	1482	3754	26	1608	3738	63
	HP	123	827	3704	348	2335	3711	219
	BM	67	114	3698	209	354	3719	90
<b>G52</b> <b>1000 1.28%</b> <b>4009.64</b>	BYZ	27	1686	3739	31	1936	3753	63
	HP	127	859	3701	392	2650	3721	218
	BM	61	107	3692	202	352	3721	90
<b>G22</b> <b>2000 1.05%</b> <b>14135.945</b>	BYZ	22	30847	12990	24	33626	12978	1744
	HP	52	2160	12864	108	4468	12929	1986
	BM	41	558	12887	131	1758	12955	1065
<b>G23</b> <b>2000 1.05%</b> <b>14142.12</b>	BYZ	22	31374	12967	24	34206	12984	1783
	HP	52	2149	12887	101	4154	12913	1994
	BM	41	556	12892	121	1644	12923	1070
<b>G32</b> <b>2000 0.25%</b> <b>1567.63975</b>	BYZ	24	6342	1316	27	7134	1314	90
	HP	98	1490	1302	641	9700	1318	1964
	BM	235	1215	1286	1803	9453	1294	609
<b>G33</b> <b>2000 0.25%</b> <b>1544.3125</b>	BYZ	18	4806	1284	21	5605	1278	96
	HP	110	1654	1280	666	9988	1294	1971
	BM	262	1381	1258	1760	9926	1274	633
<b>G35</b> <b>2000 0.64%</b> <b>8014.74</b>	BYZ	31	14879	7442	34	16328	7452	442
	HP	186	5324	7357	562	16071	7388	1975
	BM	103	718	7363	254	1803	7434	705
<b>G36</b> <b>2000 0.64%</b> <b>8005.965</b>	BYZ	38	18904	7440	40	19907	7440	457
	HP	224	6406	7336	729	20818	7386	1977
	BM	120	885	7372	552	3972	7393	716
<b>G48</b> <b>3000 0.17%</b> <b>6000.000</b>	BYZ	12	10817	6000	14	12620	6000	270
	HP	na	na	na	na	na	na	na
	BM	106	1315	6000	375	4547	6000	2179
<b>G49</b> <b>3000 0.17%</b> <b>6000.000</b>	BYZ	12	10734	6000	14	12516	6000	257
	HP	na	na	na	na	na	na	na
	BM	92	1125	6000	529	6517	6000	2220

Table 3: Comparison of the three methods:  $n = 1000$  to  $n = 3000$ . (The symbol ‘na’ indicates that the problem was unable to run on the Sparc 20 due to memory constraints.)

	BYZ	HP	BM
<b>G1</b>	12	17	7
<b>G11</b>	7	17	6
<b>G14</b>	8	17	6
<b>G43</b>	16	24	9
<b>G51</b>	12	23	9
<b>G22</b>	58	78	32
<b>G32</b>	40	77	32
<b>G35</b>	44	77	32
<b>G48</b>	89	na	70

Table 4: Memory usage (in MB) of the three methods on nine problems. (The symbol ‘na’ indicates that the problem was unable to run on the Sparc 20 due to memory constraints.)

stopping criteria that we have chosen is as follows. For  $k \geq 1$ , let

$$r_k = \frac{\varphi(L^k) - \varphi(L^{k-1})}{\varphi(L^k)}.$$

The method terminates once some  $k \geq 5$  is found such that

$$r_{k-4} + r_{k-3} + r_{k-2} + r_{k-1} + r_k < 0.0001. \quad (8)$$

In Table 5, we present data pertaining to each of the three stages of our algorithm: the vertex reordering, the projected gradient algorithm to solve the MAXCUT SDP relaxation, and the generation of the random cuts which approximate the solution of the MAXCUT problem. For the vertex reordering, we give the value  $m(G)$  of the input graph  $G$  as well as the ratio by which  $m(G)$  is reduced via the vertex reordering. More precisely, the vertex reordering finds a graph  $G'$ , and we report the ratio  $m(G')/m(G)$  under the heading ‘ratio.’ In addition, we give the time required by the vertex reordering. For the projected gradient algorithm, we report the number of iterations, the objective value of the final iterate, and the time (in seconds) required by the algorithm. Under the heading ‘zeros,’ we also give the ratio between the number of zero entries and the total number of entries in the lower half of the final iterate. Since we have implemented our cut generation scheme to take advantage of the sparsity of the final iterate, this ratio gives insight into the time needed to compute the  $n$  random cuts. For the generation of the random cuts, we report the weight of the best of  $n$  randomly generated cuts as well as the time needed to compute the  $n$  cuts. In order to illustrate the quality of the cuts produced by our algorithm, we also give the ratio between the weight of the best cut and the upper bound of the SDP relaxation for the problem. (These upper bounds were found by letting Benson et al.’s code run to a relative duality gap of  $10^{-6}$  just as above.) Notice that, for all problems except those with negative edge weights, the ratio is well above the theoretical guarantee of 0.87856. Finally, we report the total time required by all three stages under the heading ‘ttime.’

problem			reordering			sdp				cut			
name	dim	spars	$m(G)$	ratio	time	iter	val	zeros	time	val	ratio	time	ttime
G1	800	6.12%	5116819	0.85	4	69	12078.90	0.03	299	11392	0.94	83	386
G2	800	6.12%	5089021	0.86	4	60	12084.13	0.03	264	11368	0.94	84	352
G3	800	6.12%	5143322	0.84	4	59	12077.55	0.03	257	11419	0.94	84	345
G11	800	0.63%	633764	0.51	0	158	627.04	0.49	127	528	0.84	38	165
G12	800	0.63%	627506	0.51	0	130	621.61	0.48	106	522	0.84	40	145
G13	800	0.63%	615449	0.52	0	127	645.10	0.46	105	542	0.84	40	145
G14	800	1.59%	667919	0.79	1	73	3187.91	0.39	80	2957	0.93	57	138
G15	800	1.58%	636554	0.80	1	96	3169.11	0.41	106	2958	0.93	45	152
G16	800	1.59%	643689	0.80	1	95	3172.72	0.41	105	2961	0.93	55	161
G22	2000	1.05%	13334305	0.77	7	69	14123.70	0.10	924	12912	0.91	1066	1997
G23	2000	1.05%	13371095	0.77	7	59	14129.87	0.10	799	12888	0.91	1071	1877
G24	2000	1.05%	13317798	0.77	7	80	14131.32	0.10	1064	12968	0.92	1070	2141
G32	2000	0.25%	3960500	0.51	2	138	1560.75	0.49	709	1280	0.82	609	1318
G33	2000	0.25%	3950705	0.51	2	150	1537.60	0.47	781	1248	0.81	634	1417
G34	2000	0.25%	3921650	0.51	2	129	1541.66	0.48	672	1264	0.82	621	1295
G35	2000	0.64%	3891907	0.80	4	110	8000.21	0.42	760	7376	0.92	701	1465
G36	2000	0.64%	4023560	0.81	4	159	7996.19	0.40	1152	7363	0.92	717	1873
G37	2000	0.64%	3930797	0.81	4	110	8009.29	0.41	773	7387	0.92	710	1487
G43	1000	2.10%	3374697	0.76	2	68	7027.17	0.10	216	6480	0.92	140	358
G44	1000	2.10%	3301126	0.78	2	60	7022.80	0.10	191	6468	0.92	140	333
G45	1000	2.10%	3342631	0.77	2	66	7020.36	0.10	212	6475	0.92	139	353
G48	3000	0.17%	8823650	0.51	3	89	5983.75	0.48	1076	6000	1.00	2185	3264
G49	3000	0.17%	8710030	0.52	3	110	5990.68	0.47	1344	6000	1.00	2224	3571
G50	3000	0.17%	8654425	0.52	3	69	5974.22	0.47	851	5840	0.98	2200	3054
G51	1000	1.28%	974026	0.82	1	98	4002.28	0.41	167	3715	0.93	89	257
G52	1000	1.28%	1021972	0.80	1	87	4005.61	0.41	152	3698	0.92	89	242
G53	1000	1.28%	1020585	0.81	1	110	4006.90	0.40	190	3727	0.93	92	283

Table 5: Results of approximating 27 instances of MAXCUT.

## 6 Final Remarks

In this paper we have proposed a projected gradient variant of Homer and Peinado's method for solving the MAXCUT SDP relaxation which, when used in conjunction with the randomized cut procedure of Goemans and Williamson, gives a very efficient procedure for obtaining an approximate solution to the MAXCUT problem. In our computational experiments, our method with the stopping criterion (8) has performed considerably faster than Benson et al.'s and Homer and Peinado's methods, even though the quality of the cuts generated were slightly inferior to the cuts obtained by Benson et al.'s method. In addition, we have observed that our method requires less memory than these two methods.

In Section 5, we tested our method on problems with at most  $n = 3000$  mainly due to memory limitations. The next largest problems in the set of test problems that we attempted to solve were of size  $n = 5000$ , but these problems did not entirely fit in the memory of our test machine. Since our code is not currently tuned to handle problems which require swap space, our method did not perform well on problems of this size. We plan to add this capability in a future version of our code. Another way to enable the solution of larger problems is to parallelize our method in the same way as Homer and Peinado did for their method.

There are many possible ways one can try to improve and/or extend our method. One possibility is to incorporate second order information to speed up its asymptotic convergence. Another is to modify it so that it will be able to solve other SDP problems other than the MAXCUT SDP relaxation.

## Acknowledgment

The authors are grateful to Benson, Ye and Zhang, and Homer and Peinado for making their code available to us. The authors are also in debt to Paulo Zanjácomo for helping write a preliminary MATLAB implementation of our method.

## References

- [1] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.
- [2] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical optimization and circuit layout design. *Operations Research*, 36(3):493–513, 1988.
- [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.
- [4] K.C. Chang and D.H. Du. Efficient algorithms for layer assignment problems. *IEEE Transactions on Computer-Aided Design*, CAD-6(1):67–78, 1987.
- [5] R. Chen, Y. Kajitani, and S. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Transactions on Circuits and Systems*, CAS-30(5):284–299, 1983.

- [6] T. Fujie and M. Kojima. Semidefinite programming relaxation for nonconvex quadratic programs. *J. Global Optim.*, 10:168–189, 1997.
- [7] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata. Numerical evaluation of SDPA (Semidefinite Programming Algorithm). Research Report B-330, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguru-ku, Tokyo 152, Japan, September 1997.
- [8] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997.
- [9] 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).
- [10] G. H. Golub and C. E. Van Loan. *Matrix Computations: Second Edition*. The John Hopkins University Press, Baltimore, MA 21211, 1989.
- [11] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. Manuscript, Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, Takustrasse 7, D-14195 Berlin, Germany, 1997.
- [12] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.
- [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] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. Technical Report DIKU-TR-97/9, Department of Computer Sciences, University of Copenhagen, July 1997.
- [15] C-J. Lin and R. Saigal. On solving large scale semidefinite programming problems - a case study of quadratic assignment problem. Technical Report, Dept. of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI 48109-2177, 1997.
- [16] L. Lovász. On the Shannon Capacity of a graph. *IEEE Transactions on Information Theory*, IT-25(1):1–7, January 1979.
- [17] L. Lovász and A. Schrijver. Cones of matrices and setfunctions, and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.
- [18] Y. Nesterov. Semidenite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.
- [19] R.Y. Pinter. Optimal layer assignment for interconnect. *J. VLSI Comput. Syst.*, 1:123–137, 1984.
- [20] S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for 0-1 quadratic programming. *Journal of Global Optimization*, 7:51–73, 1995.

- [21] S. Poljak and Z. Tuza. The max-cut problem — a survey. In W. Cook, L. Lovász, and P. Seymour, editors, *Special Year on Combinatorial Optimization*, DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society. To be published.
- [22] N.Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Science*, 25:1–11, 1987. Originally published in Tekhnicheskaya Kibernetika, No. 1, 1987, pp. 128–139.
- [23] Y. Ye. Approximating quadratic programming with quadratic constraints. Working paper, Department of Management Science, The University of Iowa, Iowa City, 1997. To appear in *Mathematical Programming*.