Mathematical Programming 82 (1998) 291–315

Solving quadratic (0,1)-problems by semidefinite programs and cutting planes

Christoph Helmberg [a,*,1], Franz Rendl [b,2]

[a] *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, 14195 Berlin, Germany*
[b] *Technische Universität Graz, Institut für Mathematik, Steyrergasse 30, A-8010 Graz, Austria*

Received 7 December 1995; revised manuscript received 5 September 1997

Abstract

We present computational experiments for solving quadratic (0, 1) problems. Our approach combines a semidefinite relaxation with a cutting plane technique, and is applied in a Branch and Bound setting. Our experiments indicate that this type of approach is very robust, and allows to solve many moderately sized problems, having say, less than 100 binary variables, in a routine manner. © 1998 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Quadratic (0,1)-programming; Max-cut problem; Semidefinite program; Branch and bound

1. Quadratic (0,1)-problems

A basic problem in discrete optimization consists in optimizing a quadratic function over some hypercube. This type of problem is NP-hard, and it is still considered a computational challenge to solve general modest-size problems of this type to optimality.

Quadratic programming over the vertices of a hypercube appears in various equivalent formulations in the literature. We use the following model, which corresponds to the problem of finding a cut of maximum weight in an edge weighted undirected graph.

$$\text{(MC)} \qquad mc := \max x^t L x \quad \text{such that } x \in \{-1, 1\}^n. \tag{1}$$

* Corresponding author. E-mail: helmberg@zib.de;web:http://www.zib.de/helmberg.

[1] Large parts of this paper were prepared while the author was working at the Christian Doppler Laboratory for Discrete Optimization at Technische Universität Graz.

[2] E-mail: rendl@opt.math.tu-graz.ac.at.

0025-5610/98/$19.00 © 1998 The Mathematical Programming Society, Inc.
Published by Elsevier Science B.V.
PII S0025-5610(97)00111-1

This problem is one of the basic NP-hard combinatorial optimization problems. Its scientific interest has several reasons. First there are many applications that lead directly to (MC). These range from the physics of spin glasses to nonsingularity of interval matrices. A detailed description of various applications can be found in the survey paper by Poljak and Tuza [39]. Secondly, the combinatorial structure of the so-called *cut polytope* associated with problem (1) provides a basic ingredient for integer linear programming. A recent summary of theoretical properties of the cut polytope can be found in the forthcoming book by Deza and Laurent [19]. Some connections to general linear integer programs are elaborated in [25]. Finally from a more practical side, the max-cut problem has remained a challenge in the design of algorithms for its solution. In Section 3 we review the most interesting solution approaches for this problem.

Our primary goal is to provide computational experiments for problems of the form (1), based on a recently introduced semidefinite relaxation. Our main conclusion, substantiated by computational experiments on a big variety of test problems, lies in the observation that this semidefinite relaxation, combined with linear cutting plane techniques provides a strong machinery to solve (1). Our approach is very robust, and much less sensitive to structural properties than most of the other published solution methods.

When this paper was written there was no efficient routine for fixing variables within the Branch and Bound approach. In the meantime such a routine has been developed in [24]. The computational results presented in [24] build on the present paper.

The paper is structured as follows. We first recall various equivalent formulations of (1). Since the main contribution of this paper are computational experiments, we find it convenient for the reader to review the currently best solution methods for max-cut problems. This is done in Section 3. In Section 4 we provide the mathematics underlying the semidefinite relaxation used in this paper. We also describe the class of cutting planes that we use. In Section 6 we provide algorithmic details on how we actually solve our semidefinite program. The main part of the paper contains the computational experiments. We first describe the data sets we are using. Then we show the performance of our approach applied directly to these problems. Finally, we use our methodology in a Branch and Bound setting to solve problems to optimality. We conclude with a discussion of our findings and show strong points and weaknesses of our approach, as compared to previous work.

2. Equivalent formulations of (MC)

Problem (1) can be recast in several essentially equivalent forms. To start, it is well known, see e.g. [3], that the adjunction of a linear term $c^t x$ in the cost function of (1) does not change the problem significantly, because

$$\max x^t L x + 2c^t x \quad \text{such that } x \in \{-1, 1\}^n \tag{2}$$

is equivalent to

$$\max y^t \hat{L} y \quad \text{such that} \quad y \in \{-1,1\}^{n+1},\ y_{n+1} = 1,$$

where

$$\hat{L} := \begin{pmatrix} L & c \\ c^t & 0 \end{pmatrix}.$$

Since this cost function is symmetric, $y_{n+1} = 1$ need not be maintained explicitly. Therefore the inclusion of a linear term leads to a problem of the form (1), of size increased by one. It is also straightforward to see that Eq. (2) is equivalent to *Quadratic (0,1) programming* (QP),

$$\text{(QP)} \qquad \min x^t Q x + q^t x \quad \text{such that } x \in \{0,1\}^n.$$

This has been observed by many researchers, see e.g. [21,3,13].

Finally, we recall how (1) relates to the *max-cut Problem*. This connection was established for instance by Mohar and Poljak [34]. Let G be an undirected graph on vertex set $V = \{1,\ldots,n\}$ with edge weights $\{c_e\colon\ e \in E(G)\}$, given by its adjacency matrix $A = (a_{ij})$ where $a_{ij} = a_{ji} = c_e$ *for* $e \in E(G)$, $e = (ij)$, $a_{ij} = 0$ otherwise.

The max-cut Problem asks to divide V into two sets $(S, V\backslash S)$ so as to maximize the weight of the edges 'cut' by the partition.

$$\text{(Max-Cut)} \qquad \mathrm{mc(G)} := \max_{S \subseteq V} \sum_{i \in S, j \notin S} a_{ij}.$$

Let us represent partitions $(S, V\backslash S)$ by vectors $x \in \{-1,1\}^n$ with $x_i = 1$ only if $i \in S$. We denote by e the vector of ones. Using the *Laplacian L of G*, defined as

$$L := \mathrm{diag}(Ae) - A,$$

it can easily be checked that

$$\frac{1}{4} x_S^t L x_S = \sum_{i \in S, j \notin S} a_{ij}, \tag{3}$$

if $x_S \in \{-1,1\}^n$ represents the partition $(S, V\backslash S)$. Thus (1) contains the max-cut Problem as a special case. On the other hand, any symmetric matrix M can be written as the Laplace matrix of a graph plus some diagonal matrix D, $M = L + D$. Since

$$x^t D x = \mathrm{tr} D \quad \text{for all } x \in \{-1,1\}^n \tag{4}$$

holds for diagonal matrices D, it is clear that (1) is not more general than max-cut.

3. Overview of solution approaches

Linear Programming-based methods: Working with binary variables associated to the edges of the graph in question, the weight of cuts is a linear function of these edge variables. It is therefore not surprising that attempts were made to find tight linear descriptions containing the cut polytope, i.e. the convex hull of all characteristic

vectors representing cuts in the edge space. A substantial computational study based on Linear Programming (LP) and cutting planes is given by Barahona et al. [3]. These experiments suggest that LP-based methods are likely to be efficient only if the graph is reasonably sparse. (Graphs with 100 vertices and edge density of 10% can be handled by this approach, while it fails on dense graphs with 50 vertices.) This observation is further substantiated by Barahona and Titan [4] and Barahona [2], where max-cut problems on toroidal grid graphs are solved to optimality. (These graphs are highly structured and have about $2|V|$ edges.) In [2] max-cut problems on toroidal grids of sizes up to 35×35 are solved to optimality using cutting planes in conjunction with Linear Programming. The currently strongest results on toroidal grids are contained in a recent study by De Simone et al [14], where max-cut problems on toroidal grids of sizes up to 100×100, i.e. 10,000 vertices, are solved by Linear Programming-based branch and cut techniques.

Another variant of a cutting plane algorithm is proposed by De Simone and Rinaldi [15]. Here the authors work on the complete graph and try to optimize over a very general class of linear inequalities, valid for max-cut, the so-called *hypermetric inequalities*. Again, the computational success seems to be very dependent on the actual number of nonzero edges in the graph, and it deteriorates quickly, as the number of vertices increases.

Several authors approach (QP) with techniques developed for *pseudo-Boolean functions*, see e.g. [22]. The concept of *roof dual* studied in [22] corresponds to a linear relaxation of the problem over a subset of the *triangle inequalities*. Further theoretical results, including a variety of facet defining inequalities for the cut polytope are contained in the papers [5–7]. Computational results based on quadratic posiforms are described in [8]. Several heuristics to obtain integer solutions are investigated on graphs having up to 300 vertices.

Branch and Bound with Preprocessing: Pardalos and Rodgers [35,36] solve (QP) by Branch and Bound using a preprocessing phase where they try to fix some of the variables. The main idea lies in the observation that x_i can be fixed if the partial derivative of the cost function with respect to x_i does not change sign over the convex hull of the feasible points. It is rather surprising that the computational performance of this approach is quite similar to the results reported in [3], even though the approaches are completely unrelated. The method works well, if some sort of diagonal dominance holds for the cost function, or if the problem is very sparse. It seems to fail on dense problems of quite modest size ($n = 50$).

Kalantari and Bagchi [27] propose a Branch and Bound scheme where they bound (QP) by a linear convex envelope from below. Computational results on small problems ($n \leqslant 50$) indicate that again the efficiency of the approach deteriorates quickly with increasing problem size.

There exist several older computational studies dealing with our problem, such as Körner [28], Carter [10] or Williams [44]. The results contained in these papers are either on very small problems or are dominated by [3] and [35], so we do not go into further details.

Eigenvalue-based approaches: A last group of papers approaches the max-cut problem using eigenvalues. The basic idea here is to represent cuts in the vertex oriented way of *cut vectors* $x \in \{-1,1\}^n$. Mohar and Poljak [34] were the first to observe that

$$4mc(G) \leqslant \max_{x^t x = n} x^t L x = n\lambda_{\max}(L).$$

(The factor 4 comes from Eq. (3).) Therefore the largest eigenvalue of the Laplacian provides an upper bound on the weight of a maximum cut. A further improvement is proposed in [12] using Eq. (4) to obtain

$$4mc(G) \leqslant \varphi(G) := \min_{u^t e = 0} n\lambda_{\max}(L + \operatorname{diag}(u)). \tag{5}$$

Computational experiments are contained in [42] and in [40], where this bound is applied to huge graphs (with up to 50,000 vertices and several million edges). Using some of the combinatorial properties of $\varphi(G)$ investigated in [11], the paper [42] provides the first results of this bound in a Branch and Bound setting. In [41] it is shown, that the eigenvalue relaxation can alternatively be formulated as a semidefinite program, leading to computationally more stable algorithms. Computational results using this semidefinite setting are given in [26,9,23]. In his thesis, Burkard [9] investigates the basic model (SDP), formally introduced below, in a Branch and Bound setting, Helmberg [23] sets the algorithmic framework for combining (SDP) with cutting planes. Finally, [26] contains the first experiments combining the semidefinite model with polyhedral approaches in a general setting.

After this short summary of existing solution methods we will now provide the details underlying the semidefinite relaxation used in the present paper. Our computational work is an outgrowth of [26,23,25] and builds on the results contained in these papers.

4. Semidefinite relaxation

In this section we recall the semidefinite relaxation introduced in [41] to bound (MC). The starting point is the following simple observation,

$$x^t L x = \operatorname{tr} L(xx^t).$$

Let $\mathscr{F} = \{-1,1\}^n$ denote the feasible set of (MC). We consider the set $\mathscr{P}_{\mathscr{C}} := \operatorname{conv}\{xx^t : x \in \mathscr{F}\}$, the so-called *cut polytope*. With this notation (MC) is equivalent to

$$mc = \max \operatorname{tr} LX \quad \text{such that } X \in \mathscr{P}_{\mathscr{C}}.$$

Since a complete description of the polyhedron $\mathscr{P}_{\mathscr{C}}$ is currently not known, and unlikely to be available by a simple oracle due to the NP-hardness of (MC), we approximate the feasible set as follows. By construction we have

$$X \in \mathscr{P}_{\mathscr{C}} \Rightarrow X \succeq 0, \quad \operatorname{diag}(X) = e.$$

In [32] the set $\{X \succeq 0\colon \operatorname{diag}(X) = e\}$ is called *elliptope*. The basic semidefinite relaxation, used in [41,26] is

$$\text{(SDP)} \qquad \varphi(G) = \max \operatorname{tr} LX \quad \text{such that } \operatorname{diag}(X) = e,\ X \succeq 0. \tag{6}$$

We call this a *Semidefinite Program* in the matrix variable X, because it is a linear problem in X with the additional semidefiniteness constraint $X \succeq 0$. It can be solved in polynomial time using the ellipsoid method, if the accuracy of the solution is prespecified by some constant. (The fact that the optimal solution values of Eqs. (5) and (6) coincide was observed by Schrijver [43], see also [41] for extensions.)

Before we show how this relaxation can be further tightened by cutting planes, we summarize some useful properties of (SDP). Goemans and Williamson [20] have recently shown that (SDP) applied to a max-cut problem on nonnegative edge weights leads to an error of at most 13.8%. More precisely they prove the following theorem.

Theorem 4.1 [20]. *If G is a graph on nonnegative edgeweights, then $\varphi(G) \leqslant 1.138$ $\operatorname{mc}(G)$.*

Delorme and Poljak examine the behavior of (SDP) on some classes of random graphs. In particular they prove the following statement.

Theorem 4.2 [11]. *Let $G_{n,p}$ be a random graph on n vertices with edge probability p, $(0 < p < 1)$, then*

$$\lim_{n\to\infty} \frac{\varphi(G)}{mc(G)} = 1$$

with probability 1.

Finally, Laurent and Poljak [30] investigate the geometry of the set, described by the constraints of (SDP). They show that Theorem 4.1 can be further improved if the adjacency matrix A has the form $A = aa^{\mathrm{t}}$.

Theorem 4.3 [30]. *Let G be a graph with adjacency matrix $A = aa^{\mathrm{t}}$, and $a \geqslant 0$. Then $\varphi(G) \leqslant 1.125$ $\operatorname{mc}(G)$. Moreover there is a simple characterization in terms of a, for $\varphi(G) = mc(G)$ to hold.*

In view of these theoretical properties it is promising to use (SDP) as a starting point to approximate (MC). On the other hand, the Branch and Bound results contained in [42] and [9] indicate that (SDP) by itself may still be too weak to solve larger problems to optimality. (Solving instances with 70 vertices by Branch and Bound using only (SDP) could not be done routinely, as demonstrated in [9].) Therefore we go one more step and combine (SDP) with a cutting plane approach, as in integer linear programming. In the next section we describe the linear inequalities used as cutting planes.

5. Hypermetric inequalities as cutting planes

Suppose we have solved the relaxation (SDP) yielding an optimal solution X. How could we determine whether or not $X \in \mathscr{P}_{\mathscr{G}}$? Since the max-cut Problem is NP-hard, it is of course highly unlikely that there is a statement of the form: $X \in \mathscr{P}_{\mathscr{G}}$ if and only if some polynomially checkable condition is satisfied by X. We content ourselves therefore with a partial description of $\mathscr{P}_{\mathscr{G}}$, given by the hypermetric inequalities. These have been thoroughly investigated for instance by Deza and Laurent [17,18].

Since hypermetric inequalities play a fundamental role in our approach, we review their derivation and some basic facts. Let $b \in \mathbb{R}^n$ be a vector of integers, such that

$$\min\{|b^t x|: x \in \mathscr{F}\} = 1.$$

A possible choice for b having left-hand side value at least one would be to choose the integers $b_1, \ldots, b_{n-1}$ arbitrarily, and then select b_n so that $\sum_{i \leqslant n} b_i$ is odd. Any such vector gives rise to a valid constraint in our matrix model via

$$|x^t b| \geqslant 1 \iff b^t x x^t b \geqslant 1 \iff \operatorname{tr}(bb^t)(xx^t) \geqslant 1.$$

Thus $X \in \mathscr{P}_{\mathscr{G}}$ only if $\operatorname{tr}(bb^t)X \geqslant 1$. Note that any X feasible for (SDP) satisfies $b^T X b \geqslant 0$. Therefore any feasible X violates the inequality by at most one.

Let us introduce the set $\mathscr{B} := \{b \in Z^n: \sum_i b_i \text{ odd}\}$ and define

$$\mathscr{P}_{\mathscr{H}} = \{X: \operatorname{tr}(bb^t)X \geqslant 1 \text{ for all } b \in \mathscr{B}\}.$$

This gives the following relaxation.

$$\mathscr{P}_{\mathscr{G}} \subseteq \mathscr{P}_{\mathscr{H}}.$$

The set $\mathscr{P}_{\mathscr{H}}$ is generated by the intersection of an infinite number of half-spaces. Nonetheless it has been shown recently that this set is polyhedral, see [16]. The bad news is that it is currently not known how to decide efficiently whether $X \in \mathscr{P}_{\mathscr{H}}$ holds or not. Since we are going to use hypermetric inequalities as cutting planes in our model, we will describe later some heuristics to test whether $X \notin \mathscr{P}_{\mathscr{H}}$.

Even though a complete description of $\mathscr{P}_{\mathscr{H}}$ by facet defining inequalities is not known, there exist several special classes of facet defining inequalities for $\mathscr{P}_{\mathscr{G}}$, that are hypermetric.

Perhaps the most obvious class of inequalities are the *triangle inequalities*. They state that

$$x_{ij} + x_{ik} + x_{jk} \geqslant -1, \qquad x_{ij} - x_{ik} - x_{jk} \geqslant -1$$

for all triples (i,j,k) of distinct vertices of G. (To see that this is some hypermetric inequality, simply take b to be the characteristic vector of the triangle (i,j,k) in the first case, and set $b_k = -1$ in the second case.) The polyhedron, defined by these conditions only is called the *Metric Polytope*. This polytope is the basis for many polyhedral approaches to solve (MC), see for instance [3,15]. Its popularity is also due to the fact that the corresponding max-cut relaxation is exact for graphs not contractible to K_5 [1]. Its relation to the elliptope has been studied in [31,32].

A generalization of the triangle inequalities to general cliques of odd order leads to the *clique inequalities*. Let $|b|$ be the characteristic vector of a clique of size k odd (b has k nonzeros of value $+1$ or -1). Then

$$\operatorname{tr}(bb^{\mathrm{t}})X \geqslant 1$$

holds for all $X \in \mathscr{P}_{\mathscr{C}}$.

6. Implementation

This section gives a detailed description of our implementation which is an improved version of [23]. It employs the primal–dual path-following algorithm of [26] as basic optimization tool. We start with recapitulating some characteristics of this interior point code which are fundamental for the understanding of the general approach. Section 6.2 gives the parameters used for computing (SDP) and describes the heuristic for generating good cuts. Finally, we explain our procedures for finding, selecting, and adding violated hypermetric inequalities.

6.1. Interior point code

The algorithmic framework is designed to handle primal–dual pairs of the form

$$\begin{aligned} &\text{maximize} && \langle C, X\rangle + \langle b, s\rangle \\ &\text{subject to} && a - \mathscr{A}(X) - \mathscr{B}(s) = 0, \\ &&& X \succeq 0,\ s \geqslant 0, \end{aligned}$$

$$\begin{aligned} &\text{minimize} && \langle a, y\rangle \\ &\text{subject to} && Z + C - \mathscr{A}^{\mathrm{t}}(y) = 0, \\ &&& t + b - \mathscr{B}^{\mathrm{t}}(y) = 0, \\ &&& Z \succeq 0,\ t \geqslant 0, \end{aligned}$$

where $\mathscr{A}(\cdot)\colon \mathbb{R}^{n\times n} \to \mathbb{R}^k$ and $\mathscr{B}(\cdot)\colon \mathbb{R}^m \to \mathbb{R}^k$ are linear operators on the positive semidefinite matrix variable X and on the nonnegative vector s, respectively. $\mathscr{A}^{\mathrm{t}}(\cdot)$ and $\mathscr{B}^{\mathrm{t}}(\cdot)$ denote the corresponding adjoint operators. For technical reasons we require $\mathscr{A}(\cdot)$ to map the skew symmetric part to zero such that $\mathscr{A}(M) = \mathscr{A}(M^{\mathrm{t}})$ for an arbitrary $n \times n$ matrix M. We will call y the dual variables and Z and t the dual slack variables. In the case of (SDP) $\mathscr{A}(X)$ is just $\operatorname{diag}(X)$, $a = e$, and s is not used at all. When we add cutting planes, s will be needed as slack variable.

In each step the algorithm computes a Newton step for the primal–dual pair towards a point on the central trajectory. The central trajectory is characterized by primal and dual feasibility, $ZX = \mu I$, and $s \circ t = \mu e$ ($\circ$ denotes the Hadamard-product of matrices). μ is usually referred to as the barrier parameter. Since the full Newton step might violate the cone restrictions a line search is necessary. We do separate line searches for primal and dual update yielding step sizes α_p and α_d. The new μ-value is computed with respect to the new point by

$$\mu = \min\left\{\frac{\langle X,Z\rangle + \langle s,t\rangle}{n+m} \times (0.5 - 0.4\alpha^2), \mu_{\text{old}}\right\},$$

where $\alpha = \min\{\alpha_p, \alpha_d\}$. This choice is empirically efficient for max-cut and leads to a cautious reduction of μ if the last stepsize was rather small.

The computationally expensive part is the construction and factorization of the Newton-system. After some manipulations the system matrix M of size $k \times k$ reads

$$M = \mathscr{A}(Z^{-1}\mathscr{A}^{\mathrm{t}}(\cdot)X) + \mathscr{B}(t^{-1} \circ \mathscr{B}^{\mathrm{t}}(\cdot) \circ s).$$

Here, t^{-1} is short for $(t_1^{-1}, \ldots, t_m^{-1})^{\mathrm{t}}$. To shed some light on this matrix we note that $\mathscr{A}(X)$ can be represented as $(\mathrm{tr}(A_1X), \ldots, \mathrm{tr}(A_kX))^{\mathrm{t}}$ with the A_i being symmetric $n \times n$ matrices. Using this notation we have

$$M_{ij} = \mathrm{tr}(A_iZ^{-1}A_jX).$$

Since there is no hope that either X or Z^{-1} is sparse, we have to exploit any possible structure present in A_i to compute M efficiently. In our case all A_i are of form bb^{t}. The constraint on diagonal element X_{ii} reads $\mathrm{tr}(e_ie_i^{\mathrm{t}}X) = 1$ and the hypermetric inequalities read $\mathrm{tr}(bb^{\mathrm{t}}X) \geqslant 1$. So M_{ij} is best computed by

$$M_{ij} = (b_j^{\mathrm{t}}Xb_i)(b_i^{\mathrm{t}}Z^{-1}b_j).$$

Precomputing Xb_i and $b_i^{\mathrm{t}}Z^{-1}$ for each row leads to $O(n^2k + nk^2)$ arithmetic operations for building M. In practice this can be improved significantly by making use of the sparsity of the vectors.

M is symmetric, positive semidefinite, and dense, see [23]. We use the LDL^{t} factorization to solve this system. Although LDL^{t} decomposition is comparatively fast for solving dense linear systems, about 60% of our computation time are spent in this routine. The predictor–corrector approach significantly reduces the number of factorizations needed, so we employ it whenever $k > 2n$. In contrast to other algorithms we do not compute a new μ on base of the predicted point but stick with our a priori choice. For some strange reasons this leads to a very stable algorithm.

6.2. (SDP) relaxation

To compute (SDP) it remains to specify a starting point. Obviously $X = I$ is the ideal starting point on the primal side, since it is the center of the highly symmetric max-cut polytope and elliptope. It remains to find a feasible assignment for the dual variables Z and y. Since

$$Z = -L + \sum_{i=1}^{n} y_ie_ie_i^{\mathrm{t}}$$

we can choose the y_i such that Z is diagonally dominant and therefore positive definite. There seems to be no need for a more sophisticated choice. We now compute Newton steps till the gap between primal and dual value is small enough,

$$\langle a,y\rangle - \langle L,X\rangle \leqslant \langle a,y\rangle \times 5 \times 10^{-6}.$$

At termination X is hopefully close to some cut product xx^t and contains useful information for constructing good cuts. We use the following heuristic to exploit this information. For each row of X we do the following. We round the values to a $\{-1, 1\}$-vector and continue changing the sign of the element which yields the largest improvement till no increase of the objective can be gained by flipping single nodes. We take the best of all these cuts. For almost all our test examples this was already the optimal cut.

If the gap between lower bound and dual objective is now smaller than one (we assume that the objective yields integer values for cuts) we stop, otherwise we try to find hypermetric inequalities which are violated by X.

6.3. *Detecting violated hypermetric inequalities*

In the case of the max-cut problem it is not difficult to find violated inequalities. In fact, we find a lot more than we are willing to include in our relaxation because adding m inequalities implies that each Newton step will require $O(n+m)^3$. To keep the code efficient it is extremely important to select just a small number of promising inequalities from the vast set of violated inequalities.

The first criterion which comes to mind is the amount of violation. This is a satisfactory criterion as long as triangle inequalities are used exclusively. For general inequalities the following geometric criterion seems to work better. For a violated inequality compute the intersection of the straight line segment between X and I, the center of the elliptope. We prefer inequalities with small distance of this intersection to I.

During the separation process we maintain a heap of the best inequalities with respect to the geometric criterion. The heap offers room for m inequalities ($m = n$ in case of a large add and $m = n/3$ for small adds, see Section 6.4). If the heap is full a new inequality is added to the heap only if it is better than the worst inequality in the heap. In this case the worst is replaced by the new inequality.

We employ three different strategies for separating inequalities which are violated by the current X. Firstly, we enumerate all triangle inequalities. Secondly, we try to extend clique inequalities which have already proven to be important (the corresponding dual cost y_i is sufficiently large) by adding two more nodes. The two nodes and their signs are determined by complete enumeration over the zero components of the old inequality ($O(n^2)$ for each old inequality). The best two nodes with respect to the geometric criterion yield the candidate.

Finally, we employ a very simple heuristic to construct general hypermetric candidates. We are looking for integer vectors b with the following three properties. The sum of the elements of b is odd, b^tXb is close to zero, and $|\langle b, c\rangle| = 1$ where c is the currently best cut vector. The third property is motivated by the idea that the new inequality should be tight for the optimal solution. We start with a vector which satisfies the first and third property (we use e_i). Now we try to decrease

b^tXb maintaining these two properties. Iteratively, we run through indices j from 1 to n. For each index j, we construct for each $k \neq j$ the vector

$$b^k = b + \begin{cases} e_j + e_k & \text{if } 1 \cdot c_j + 1 \cdot c_k = 0, \\ e_j - e_k & \text{if } 1 \cdot c_j + (-1) \cdot c_k = 0. \end{cases}$$

The case distinction guarantees that the third property is maintained. Let $\bar{k}$ be the index such that $(b^k)^tXb^k$ is minimal. If $(b^{\bar{k}})^tXb^{\bar{k}} < b^tXb$ then we replace b by $b^{\bar{k}}$ otherwise b remains unchanged for this j. Continue with the next j. The iteration over all j is repeated till all indices were tried without further progress. If in the end $b^tXb < 1$ this is a violated inequality.

6.4. *Adding and dropping cutting planes*

We have explained how we find violated hypermetric inequalities and which of them we want to include. All we have to do now is to restart from the center $X = I$, compute a new dual starting point, and iterate. On its path from the center towards the solution of this new relaxation X will at some point leave the cut polytope. At this point it pays off to add a few newly violated inequalities, push X back inside a little bit and continue. We do this several times and then solve the improved relaxation exactly again. To distinguish between the process of adding inequalities at the exact solution of the relaxation and adding inequalities while still solving the current relaxation we call the first a *large add* and the second a *small add*. The process of a small add requires some more explanation.

We use the triangle inequalities for triggering small adds, i.e. we check after each iteration for a violated triangle inequality by enumeration. If a violation is detected we do three more Newton steps to allow some more inequalities to become violated and then apply the same separation and selection process as above. To get a new feasible primal point X is pushed back along the straight line segment from X to I such that all added inequalities are satisfied. For the dual variables we try to avoid changes in the hope that the current dual point is also a good counterpart to a just slightly changed X. Therefore we set the y_i-values corresponding to the new inequalities to zero, and the corresponding t_j-values (the counterparts to the slack variables s_j) to one. This results in an infeasible dual point because feasibility requires $t_j = -y_i$. None the less the dual objective function value remains a valid upper bound as long as the y_i do not get positive. Since the t_j cannot get negative and the Newton direction tries to satisfy $t_j = -y_i$, a positive y_i is an unlikely event and has not been observed in practice. As a safeguard we iterate till a full dual Newton step is performed, this removes infeasibilities after roughly three iterations.

Since changes on the primal and dual side are small the new variables form, in general, an acceptable primal–dual pair and the algorithm recovers within the aforementioned three iterations. We can continue with the next small add. Empirically it was our impression that small adds help to generate a representative collection of inequalities which are well distributed over all vertices of the graph and help to reduce

the number of redundant inequalities. A heuristic explanation for this behavior is that due to the underlying barrier method the new central path is pushed away from the newly added inequalities, such that the next set of inequalities is generated with respect to an X which generously satisfies all previous inequalities.

On the negative side small adds inhibit the early recognition of inequalities which have become redundant. This is due to the fact that the barrier parameter μ is in general still too large for the dual variables to converge to zero. To avoid the accumulation of redundant inequalities – and to check whether the relaxation is already good enough – after several small adds we solve the relaxation exactly. The dual costs of the exact solution provide a good indicator for the importance of inequalities. We remove all inequalities i that have dual cost $t_i < \max(t) \times 10^{-4}$.

Summing up, we first compute the solution of (SDP) without any inequalities. For this converged solution a large add, adding n violated inequalities, is performed. After the inequalities have been added the optimization process is restarted from "the center". In optimizing this relaxation each iterate is checked for violated triangle inequalities. Violated triangle inequalities trigger, with a delay of three iterations, a small add adding $n/3$ violated inequalities. A small add allows for (infeasible) restarting in the vicinity of the last iterate, the optimization process is continued and the triggering mechanism remains active. After 10 small adds further iterates are checked no more for violated triangle inequalities, the optimal solution of the current relaxation is computed. We call this combination of one large add and 10 small adds one round or phase of adding inequalities. By means of the dual costs of the optimal solution inequalities with small dual costs are eliminated and the process is iterated starting with the next large add. Both, small and large adds employ all three separation routines.

7. Numerical results

In this section we will report numerical results on six different types of test problems. The first part is devoted to the cutting plane algorithm itself. In the second part we investigate possibilities to utilize this algorithm in a Branch and Bound scheme.

All six types of test problems are randomly generated, four of them are formulated as max-cut problems on graphs, two as quadratic $(0,1)$ programming problems.

The first, called $G_{0.5}$, consists of unweighted graphs with edge probability 1/2. The second type, $G_{-1/0/1}$, is a weighted (complete) graph with edge weights chosen uniformly from $\{-1,0,1\}$.

The class G_{p} was suggested by one of the referees and consists of graphs whose edge sets are the union of two random planar graphs. The instances were generated by the platform independent graph generator rudy that was written by G. Rinaldi. The graphs are completely specified by the command line arguments, these can be found in Table 1. Table 2 gives the arguments for the class $G_{\pm\mathrm{p}}$ which is the weighted

Table 1
Arguments for the graph generator rudy for G_p

-planar	30	100	3001	-planar	30	100	3002	+
-planar	30	100	3011	-planar	30	100	3012	+
-planar	30	100	3021	-planar	30	100	3022	+
-planar	30	100	3031	-planar	30	100	3032	+
-planar	30	100	3041	-planar	30	100	3042	+
-planar	30	100	3051	-planar	30	100	3052	+
-planar	30	100	3061	-planar	30	100	3062	+
-planar	30	100	3071	-planar	30	100	3072	+
-planar	30	100	3081	-planar	30	100	3082	+
-planar	30	100	3091	-planar	30	100	3092	+
-planar	40	100	4001	-planar	40	100	4002	+
-planar	40	100	4011	-planar	40	100	4012	+
-planar	40	100	4021	-planar	40	100	4022	+
-planar	40	100	4031	-planar	40	100	4032	+
-planar	40	100	4041	-planar	40	100	4042	+
-planar	40	100	4051	-planar	40	100	4052	+
-planar	40	100	4061	-planar	40	100	4062	+
-planar	40	100	4071	-planar	40	100	4072	+
-planar	40	100	4081	-planar	40	100	4082	+
-planar	40	100	4091	-planar	40	100	4092	+
-planar	50	100	5001	-planar	50	100	5002	+
-planar	50	100	5011	-planar	50	100	5012	+
-planar	50	100	5021	-planar	50	100	5022	+
-planar	50	100	5031	-planar	50	100	5032	+
-planar	50	100	5041	-planar	50	100	5042	+
-planar	50	100	5051	-planar	50	100	5052	+
-planar	50	100	5061	-planar	50	100	5062	+
-planar	50	100	5071	-planar	50	100	5072	+
-planar	50	100	5081	-planar	50	100	5082	+
-planar	50	100	5091	-planar	50	100	5092	+
-planar	60	100	6001	-planar	60	100	6002	+
-planar	60	100	6011	-planar	60	100	6012	+
-planar	60	100	6021	-planar	60	100	6022	+
-planar	60	100	6031	-planar	60	100	6032	+
-planar	60	100	6041	-planar	60	100	6042	+
-planar	70	100	7001	-planar	70	100	7002	+
-planar	70	100	7011	-planar	70	100	7012	+
-planar	70	100	7021	-planar	70	100	7022	+
-planar	80	100	8001	-planar	80	100	8002	+
-planar	80	100	8011	-planar	80	100	8012	+
-planar	90	100	9001	-planar	90	100	9002	+
-planar	100	100	10001	-planar	100	100	10002	+

version of G_p, i.e., the edges have random weights ± 1. In fact, the graph instances of $G_{\pm p}$ are the same as for G_p, the only difference are the random weights ± 1 on the edges.

The fifth class, Q_{100}, goes back to [44] and is formulated in quadratic $(0, 1)$ programming terms. Its most natural setting with respect to our definition of (QP) is to set all elements q_{ij} of Q with $j < i$ to zero and choose the others uniformly from

Table 2
Arguments for the graph generator rudy for $G_{\pm p}$

-planar	30	100	3001	-planar	30	100	3002	+	-random	0	1	300	-times	2	-plus -1
-planar	30	100	3011	-planar	30	100	3012	+	-random	0	1	301	-times	2	-plus -1
-planar	30	100	3021	-planar	30	100	3022	+	-random	0	1	302	-times	2	-plus -1
-planar	30	100	3031	-planar	30	100	3032	+	-random	0	1	303	-times	2	-plus -1
-planar	30	100	3041	-planar	30	100	3042	+	-random	0	1	304	-times	2	-plus -1
-planar	30	100	3051	-planar	30	100	3052	+	-random	0	1	305	-times	2	-plus -1
-planar	30	100	3061	-planar	30	100	3062	+	-random	0	1	306	-times	2	-plus -1
-planar	30	100	3071	-planar	30	100	3072	+	-random	0	1	307	-times	2	-plus -1
-planar	30	100	3081	-planar	30	100	3082	+	-random	0	1	308	-times	2	-plus -1
-planar	30	100	3091	-planar	30	100	3092	+	-random	0	1	309	-times	2	-plus -1
-planar	40	100	4001	-planar	40	100	4002	+	-random	0	1	400	-times	2	-plus -1
-planar	40	100	4011	-planar	40	100	4012	+	-random	0	1	401	-times	2	-plus -1
-planar	40	100	4021	-planar	40	100	4022	+	-random	0	1	402	-times	2	-plus -1
-planar	40	100	4031	-planar	40	100	4032	+	-random	0	1	403	-times	2	-plus -1
-planar	40	100	4041	-planar	40	100	4042	+	-random	0	1	404	-times	2	-plus -1
-planar	40	100	4051	-planar	40	100	4052	+	-random	0	1	405	-times	2	-plus -1
-planar	40	100	4061	-planar	40	100	4062	+	-random	0	1	406	-times	2	-plus -1
-planar	40	100	4071	-planar	40	100	4072	+	-random	0	1	407	-times	2	-plus -1
-planar	40	100	4081	-planar	40	100	4082	+	-random	0	1	408	-times	2	-plus -1
-planar	40	100	4091	-planar	40	100	4092	+	-random	0	1	409	-times	2	-plus -1
-planar	50	100	5001	-planar	50	100	5002	+	-random	0	1	500	-times	2	-plus -1
-planar	50	100	5011	-planar	50	100	5012	+	-random	0	1	501	-times	2	-plus -1
-planar	50	100	5021	-planar	50	100	5022	+	-random	0	1	502	-times	2	-plus -1
-planar	50	100	5031	-planar	50	100	5032	+	-random	0	1	503	-times	2	-plus -1
-planar	50	100	5041	-planar	50	100	5042	+	-random	0	1	504	-times	2	-plus -1
-planar	50	100	5051	-planar	50	100	5052	+	-random	0	1	505	-times	2	-plus -1
-planar	50	100	5061	-planar	50	100	5062	+	-random	0	1	506	-times	2	-plus -1
-planar	50	100	5071	-planar	50	100	5072	+	-random	0	1	507	-times	2	-plus -1
-planar	50	100	5081	-planar	50	100	5082	+	-random	0	1	508	-times	2	-plus -1
-planar	50	100	5091	-planar	50	100	5092	+	-random	0	1	509	-times	2	-plus -1
-planar	60	100	6001	-planar	60	100	6002	+	-random	0	1	600	-times	2	-plus -1
-planar	60	100	6011	-planar	60	100	6012	+	-random	0	1	601	-times	2	-plus -1
-planar	60	100	6021	-planar	60	100	6022	+	-random	0	1	602	-times	2	-plus -1
-planar	60	100	6031	-planar	60	100	6032	+	-random	0	1	603	-times	2	-plus -1
-planar	60	100	6041	-planar	60	100	6042	+	-random	0	1	604	-times	2	-plus -1
-planar	70	100	7001	-planar	70	100	7002	+	-random	0	1	700	-times	2	-plus -1
-planar	70	100	7011	-planar	70	100	7012	+	-random00		1	701	-times	2	-plus -1
-planar	70	100	7021	-planar	70	100	7022	+	-random	0	1	702	-times	2	-plus -1
-planar	80	100	8001	-planar	80	100	8002	+	-random	0	1	800	-times	2	-plus -1
-planar	80	100	8011	-planar	80	100	8012	+	-random	0	1	801	-times	2	-plus -1
-planar	90	100	9001	-planar	90	100	9002	+	-random	0	1	900	-times	2	-plus -1
-planar	100	100	10001	-planar	100	100	10002	+	-random	0	1	1000	-times	2	-plus -1

$\{-100, \ldots, 100\}$. Since we have already specified weights on the diagonal of Q we do not need the linear term q. We include this last type for comparison with [3]. We will also give examples of Q_{100} with density of 20% to demonstrate that our algorithm is insensitive to different densities. We will call this latter class $Q_{100,0.2}$.

All our experiments were computed on an HP 9000/715.

7.1. Numerical results for (SDP) with hypermetric inequalities

As described in Section 6 the algorithm first computes (SDP) and iteratively performs a large add followed by 10 small adds till the gap between upper bound and best known solution is less than one. A second stopping rule used here is that after 5 h the algorithm is not allowed to add any more inequalities. However, we wait till the current relaxation is solved. This may take considerably longer than 5 h.

We illustrate the typical behavior of the algorithm on an example of class $G_{0.5}$ in Table 3. The dimension of the graph is 70. The maximum cut has value 708. The first column gives the number of Newton steps, the second computation time, the third the development of the upper bound, the fourth the relative gap in percent (= (ubd − lbd)/ubd × 100), the fifth the maximal violation of the triangle inequalities, the sixth the number of constraints, which corresponds to k as used in Section 6.1. The last three columns give the number of triangle facets, larger clique facets, and general hypermetric inequalities included in the relaxation. Each line gives the current figures after one round of adding inequalities and solving the resulting relaxation exactly. In particular the first line is the (SDP) solution, the second the solution after one large and 10 small adds, etc.

During the adding phase of the last line computation time exceeded 5 h. Although no more inequalities were added thereafter it took more than 1 hour to solve this last relaxation in just 20 Newton steps. Note that computation time per line increases rapidly as more and more inequalities are added, whereas the number of iterations per line remains almost constant. Furthermore we can observe a tailing off effect after the first round of adding inequalities: after about 0.28% (1 min) of the total computation time we have already reached about 70% of the total improvement. We will make use of this fact in the Branch and Bound approach.

Due to this special structure of the algorithm computation times vary largely between different problems of the same dimension. Whenever we reach the exact solution of the running relaxation and the bound is not yet good enough another round of adding inequalities is needed and this increases computation times drastically. We ask the reader to keep this in mind when looking at average results given in Table 4.

Table 3
Example of $G_{0.5}$, $n = 70$, maximum cut value is 708

Iter	h:mm:ss	ubd	%	viol	#constr	#tri	#cli	#hyp
11	1	726.71	2.57	0.97	70	0	0	0
55	56	715.47	1.04	0.62	370	300	0	0
103	8:46	713.10	0.72	0.46	658	509	60	19
152	31:43	711.91	0.55	0.26	893	595	165	63
199	1:18:26	711.36	0.47	0.24	1112	608	332	102
251	2:55:28	711.02	0.42	0.15	1326	629	497	130
299	4:50:56	710.79	0.39	0.13	1435	564	645	156
319	6:04:32	710.74	0.39	0.08	1487	574	681	162

Table 4
Average results by using cutting planes with a time limit of 5 h

n	nr	h:mm:ss	solved	gap	φ-gap	h:mm:ss	solved	gap	φ-gap
		$G_{0.5}$				$G_{-1/0/1}$			
30	10	1	10	0	2.36	1	10	0	11.57
40	10	33	10	0	2.67	27	10	0	13.53
50	10	3:39	10	0	2.44	4:18	10	0	11.82
60	5	1:44:00	4	0.38	2.29	1:13:52	4	1.133	13.08
70	3	4:03:10	1	0.28	2.26	2:11:11	2	1.306	13.85
80	2	3:40:28	1	0.39	2.13	5:34:48	0	1.556	12.90
90	1	5:57:00	0	0.46	2.39	5:37:11	0	5.343	19.01
100	1	5:02:05	0	0.78	2.53	5:38:25	0	5.299	16.04
		G_{p}				$G_{\pm\mathrm{p}}$			
30	10	1	10	0	2.66	0	10	0	11.73
40	10	14	10	0	3.02	5	10	0	10.24
50	10	1:34	10	0	3.16	20	10	0	11.99
60	5	6:13	5	0	3.27	2:21	5	0	12.08
70	3	9:25	3	0	3.21	6:46	3	0	13.41
80	2	1:06:00	2	0	3.22	1:05	2	0	13.95
90	1	13:53	1	0	2.79	35:47	1	0	11.99
100	1	5:27:26	0	0.87	4.03	1:12:16	1	0	11.92
		Q_{100}				$Q_{100,0.2}$			
30	10	5	10	0	4.13	3	10	0	4.94
40	10	1:29	10	0	7.15	1	10	0	3.92
50	10	3:56	10	0	7.01	34	10	0	4.96
60	5	6:35	5	0	6.44	1:01	5	0	7.40
70	3	15:25	3	0	6.35	9:04	3	0	7.60
80	2	36:08	2	0	8.03	31	2	0	6.38
90	1	12:49	1	0	3.98	12:31	1	0	5.70
100	1	3:01:24	1	0	5.08	16:10	1	0	4.30

Table 4 lists computational results for the six classes of random test problems described above. The first column gives the dimension of the problems, the second the number of instances computed. This is followed by the average computation time. Column *solved* gives the number of instances where the algorithm could prove the lower bound to be optimal. For those which could not be solved to optimality within the given time limit of "5 h", *gap* gives the average relative gap (ubd-lbd)/ubd in percent. The last column gives the average relative gap of (SDP) without any additional cutting planes.

All problems of type $G_{\pm\mathrm{p}}$, Q_{100}, and $Q_{100,0.2}$ were solved to optimality within the given time limit. To compare these results to Table 3 of [3] we should first point out that computation times are not comparable and that Barahona et al. restricted themselves to a time limit of 10 min. In Table 3 they give results for instances with density 20% up to $n = 80$ and for instances of density larger than 70% up to $n = 30$. Even considering serious speedup due to improved computer technology it seems to

be fair to say that for dense problems (SDP) with cutting planes is by far superior. For sparse problems the linear approach will in general be more attractive, since we cannot exploit sparsity and thus our approach is limited to a rather small number of nodes.

Obviously, examples of type G_p, $G_{\pm p}$, Q_{100}, and $Q_{100,0.2}$ are much easier to solve than the instances of $G_{0.5}$ and $G_{-1/0/1}$. Here the algorithm is quite successful for instances with up to 60 nodes. Besides from enormous increase in computation time we expect that for examples of size larger than 90 more sophisticated separation routines will be needed to prove optimality.

We illustrate the effect of the current separation routines on the examples of G_p. For these almost planar graphs one would expect that the triangle inequalities suffice to prove optimality. In Table 5 we consider three configurations of the separation routines. Triangle inequalities are separated in all of them. They are also the only kind of inequalities separated in the first configuration. In the second configuration the clique extension heuristic is added. In the third configuration clique extension is switched off again and the hypermetric heuristic is used instead.

Both additional separation routines speed up the algorithm significantly and make instances solvable that were not solvable using triangle inequalities exclusively. On many occasions the combination triangle inequalities plus clique extension heuristic beats the standard setup using all three separation routines by a little (in terms of computation time). However, on some examples, e.g. for $n = 80$, the standard setup is considerably faster than any other combination.

In view of their simplicity the success of the additional separation routines is astonishing. However, significant additions to the triangle inequalities are typically produced by these heuristics only after one or two phases of adding triangle inequalities exclusively. In a Branch and Bound approach with only one phase of adding inequalities it may therefore pay to drop the two heuristics to obtain somewhat shorter computation times. To avoid confusion we will not make use of this possibility and stay with the standard constellation with all three separation routines working.

Table 5
Average results by using different separation strategies for the examples of G_p with a time limit of 5 h

G_p n	nr	only triangle			triangle + cliques			triangle + hypermet.		
		h:mm:ss	solved	gap	h:mm:ss	solved	gap	h:mm:ss	solved	gap
30	10	0	10	0	0	10	0	0	10	0
40	10	25	9	0.8601	14	10	0	24	10	0
50	10	5:44	9	0.8324	1:36	10	0	2:38	10	0
60	5	38:20	3	0.7251	5:08	5	0	18:25	5	0
70	3	48:01	2	0.615	8:54	3	0	12:02	3	0
80	2	3:20:58	0	0.8173	2:03:30	2	0	2:16:32	2	0
90	1	2:07:41	1	0	13:58	1	0	43:11	1	0
100	1	5:24:24	0	1.125	5:35:15	0	0.8751	5:29:42	0	1.095

Table 6
Larger examples for (SDP) and one phase of adding cutting planes

n	lbd	mm:ss	φ-ubd	φ-gap	h:mm:ss	ubd	gap
$G_{0.5}$							
150	3168	16	3237.3	2.14	14:23	3209.2	1.28
200	5514	52	5618.9	1.94	38:58	5581.8	1.22
250	8574	2:00	8725.1	1.73	1:27:20	8673.4	1.15
300	12147	4:34	12368.5	1.80	2:35:53	12306.4	1.29
350	16390	8:10	16679.6	1.74	4:41:20	16613.4	1.34
400	21479	12:21	21811.0	1.52	7:14:38	21736.5	1.18
450	27144	25:00	27538.9	1.47	9:29:51	27450.5	1.12
$G_{-1/0/1}$							
150	526	18	639.1	17.7	14:11	587.6	10.5
200	865	52	1039.0	17.2	32:00	986.4	12.3
250	1135	2:14	1389.8	18.7	1:26:36	1312.3	13.5
300	1654	4:45	1993.2	17.0	2:24:21	1897.4	12.8
350	2126	8:32	2576.0	17.5	5:55:03	2467.9	13.9
400	2518	15:14	3074.0	18.1	8:06:29	2948.6	14.6
450	2794	29:06	3450.6	19.0	12:24:43	3312.0	15.6

To explore the limits of our code we give some examples of larger sizes for problems of type $G_{0.5}$ and $G_{-1/0/1}$ in Table 6. Only one example is computed for each dimension n. lbd gives the best solution found. The computation time given in the next column refers to the time needed to compute φ, φ-ubd is the corresponding upper bound with relative gap φ-gap. The last three columns give the total computation time after one phase of adding inequalities with the resulting upper bound and relative gap.

As stated in Theorem 4.2 φ is constantly getting better for the class $G_{0.5}$ as n increases. On the other hand the upper bound φ is constantly getting worse for $G_{-1/0/1}$. In both cases the improvement gained by one phase of adding inequalities is still considerable but the relative improvement is decreasing. Computing times do not allow for adding more inequalities and are, in fact, prohibitive for these large examples.

7.2. Branch and Bound

In [9] a Branch and Bound code was implemented using (SDP) as bounding procedure. The results were good yet below our expectations. Using (SDP) with cutting planes, however, seems to be worth the trouble in spite of the rather large computational costs involved.

Typically max-cut Branch and Bound codes branch by fixing the relation between two vertices. This is usually called branching on an edge. Either both end points are put into the same set (the edge is not in the cut) or the endpoints must go into different sets (the edge is in the cut). Both cases can be modeled by replacing the two vertices by one new vertex, thereby reducing the dimension of the cost matrix by

one. We illustrate this for the case that $x_{n-1,n}$ is set to one ($n-1$ and n go into the same set). All feasible solutions with $x_{n-1,n}=1$ satisfy $x_{i,n-1}=x_{i,n}$ for $1\leqslant i<n-1$ and therefore one can eliminate the variables having at least one index n. The new symmetric cost matrix $\bar{C}$ of order $n-1$ is constructed from the old cost matrix C by (only the upper triangle is specified)

$$\bar{c}_{ij}=\begin{cases} c_{ij} & \text{for } 1\leqslant i\leqslant j<n-1,\\ c_{i,n-1}+c_{i,n} & \text{for } 1\leqslant i<n-1,\ j=n-1,\\ c_{n-1,n-1}+2c_{n-1,n}+c_{n,n} & \text{for } i=j=n-1. \end{cases}$$

As we have mentioned in Section 7.1 the first round of adding inequalities typically yields substantial improvement of the bound, and afterwards we observe a strong tailing off effect. Therefore we restricted the bounding procedure to just one phase of adding inequalities (one large and ten small adds). Since we do not yet know how to fully exploit the dual information for variable fixing we keep branching on edges till the gap between upper bound and lower bound is closed to our satisfaction.

Branching rules: The success of Branch and Bound algorithms depends very much on the choice of the edge to branch on next. Numerous different branching rules of arbitrary complexity can be thought of but it is quite impossible to predict their performance on an arbitrary cost function. We will report our experience with a few very simple rules in the following.

Probably the first strategy which comes to mind is to select the edge ij with $|x_{ij}|$ maximal. Separating or contracting the vertices i and j as suggested by the sign of x_{ij} will not change the problem substantially but setting x_{ij} opposite to its current sign should lead to a sharp drop of the optimal solution in the corresponding subtree. If the bound also drops as fast we can hope that this subtree will be cut off quickly. We will call this rule R_1.

With the complete primal X matrix available deciding upon only one edge seems to be a waste of information. Rule R_2 will choose i and j such that their rows are "closest" to a $\{-1,1\}$ vector, i.e. they minimize $\sum_{k=1}^{n}(1-|x_{ik}|)^2$. Using this rule we hope for the same effect as above assuming that for two very well articulated rows i and j $|x_{ij}|$ will also be quite large.

In rule R_3 we follow the opposite idea. The quality of the bound should get better fast if we fix the most difficult decisions. Therefore we branch on edge ij which minimizes $|x_{ij}|$.

Probably it might be better to link a vertex which is hard to fix to a vertex which is quite sure about its position. We try to follow this strategy in rule R_4 by choosing vertex i to be the vertex minimizing $\sum_{k=1}^{n}(1-|x_{ik}|)^2$ and vertex j to be the vertex minimizing $\sum_{k=1}^{n}x_{jk}^2$.

Finally there are the triangle inequalities which may offer useful information. If in an active triangle inequality the coefficient of an edge is $+1$ we assume that the value of this edge tends too much towards -1. If on the other hand the coefficient is -1 the value of the edge tends too much towards $+1$. We extend this interpretation to hypermetric inequalities in general. Edges being covered by several inequalities, all of

them indicating the same behavior, should be an object of interest. In rule R_5 for each edge we sum up – over all active hypermetric inequalities – the product of the dual cost of the inequality times the coefficient of this edge. We pick the edge with maximal absolute value of this sum.

Experimental Comparison: We compare these rules for the class $G_{-1/0/1}$ using the same instances as in Table 4, but omit examples of size 30 and 40 which are usually solved in the root node. The results are given in Table 7. To facilitate the interpretation of these results we also describe the typical structure of the Branch and Bound trees for the branching rules.

R_1 and R_2 indeed show the behavior predicted. Branching against the variable x_{ij} leads to much smaller optimal cuts for the subtree which is consequently cut off most of the time. Branching with x_{ij} does not change the optimal solution but does not improve the bound by much either. This usually leads to long chains in the branch tree. Especially if the bound is not good enough to cut off the opposite subtree immediately, the same behavior is observed again for the subtree. Obviously, difficult decisions are postponed. This branching rule may lead to a high number of branch-

Table 7
Comparison of branching rules R_1–R_5

	n	nr	h:mm:ss	min-time	max-time	nodes	min	max
	50	10	3:11	12	6:29	25	1	57
	60	5	12:20	28	46:04	64	1	209
R_1	70	3	39:42	13:18	1:17:00	109	31	237
	80	2	3:43:15	3:21:38	4:04:52	411	303	519
	90	1	176:17:56			11279		
	50	10	2:42	12	6:20	19	1	49
	60	5	8:53	28	23:45	36	1	99
R_2	70	3	28:01	11:14	53:03	69	25	147
	80	2	2:17:12	2:16:21	2:18:04	213	187	239
	90	1	41:33:43			2339		
	50	10	2:32	12	7:07	10	1	31
	60	5	11:26	28	38:07	31	1	97
R_3	70	3	45:54	4:22	1:54:16	67	5	173
	80	2	1:56:38	6:14	3:47:03	367	211	523
	90	1	115:56:40			5033		
	50	10	1:56	12	7:01	12	1	37
	60	5	13:10	28	40:21	31	1	101
R_4	70	3	50:21	5:41	2:03:27	73	7	183
	80	2	7:40:25	4:44:30	10:36:20	394	267	521
	90	1	160:23:59			7143		
	50	10	2:29	12	6:24	12	1	33
	60	5	9:05	28	29:26	33	1	97
R_5	70	3	37:07	5:21	1:23:47	69	7	167
	80	2	4:00:23	2:25:59	5:34:47	294	167	421
	90	1	47:24:48			2641		

ing nodes but as the tree is not very dense, many of them are solved for small dimensions, thus resulting in faster overall computation times than rules with fewer nodes in higher dimensions. The more global point of view of rule R_2 in selecting the edge seems to pay off.

R_3 and R_4 also match our expectations. For these rules both branches are equally difficult to solve but the bound improves fast. This results in very dense trees which are not very deep. This time R_3 seems to be superior to R_4. Although for some problems these two rules result in fewer branching nodes they are quite a bit slower than R_2 since all problems are solved for large dimensions.

There is no typical structure of the Branch and Bound tree for rule R_5. Looking at the computational results R_5 seems quite attractive and is second best after rule R_2.

Experience with R_2: We conclude our section on numerical results by using rule R_2 to solve the same problems as of Table 4 and to solve a few larger examples of the class $G_{0.5}$ with a performance guarantee of 1%.

Comparing the results of Table 8 with Table 4 we see that for small sizes up to 40 the direct approach is likely to be faster, but for larger sizes the Branch and Bound approach is more attractive (keep in mind that in Table 4 the algorithm was stopped after approximately 5 h). Reasonable computing times can be expected for sizes up to 80.

For the easier classes G_p, $G_{\perp p}$, Q_{100} and $Q_{100,0.2}$ Tables 9 and 10 indicate that the direct approach is often more efficient. Obviously, we branch before the bound is good enough. As we explained above, branching by Rule R_2 does not improve the bound significantly for one of the two subproblems. So even for easy problems

Table 8
Average branch and bound results for the instances of Table 4, part 1

n	nr	h:mm:ss	min-time	max-time	nodes	min	max
$G_{0.5}$							
30	10	1	0	3	1	1	1
40	10	34	0	2:14	7	1	27
50	10	2:31	0	4:36	16	1	33
60	5	11:50	28	35:57	56	1	147
70	3	54:37	17:38	1:50:41	139	65	253
80	2	1:46:48	36:06	2:57:30	154	77	231
90	1	49:17			613		
100	1	131:01:24			5741		
$G_{-1/0/1}$							
30	10	1	1	2	1	1	1
40	10	42	0	1:37	9	1	21
50	10	2:42	12	6:20	19	1	49
60	5	8:53	28	23:45	36	1	99
70	3	28:01	11:14	53:03	69	25	147
80	2	2:17:12	2:16:21	2:18:04	213	187	239
90	1	41:33:43			2339		
100	1	78:41:28			3369		

Table 9
Average branch and bound results for the instances of Table 4, part 2

n	nr	h:mm:ss	min-time	max-time	nodes	min	max
G_p							
30	10	1	0	4	1	1	1
40	10	22	2	1:23	4	1	15
50	10	1:57	15	5:38	10	1	37
60	5	7:32	36	22:30	28	5	71
70	3	18:19	7:37	32:57	33	11	65
80	2	1:51:18	1:37:22	2:05:15	121	101	141
90	1	1:18:19			59		
100	1	93:06:00			3557		
$G_{\pm p}$							
30	10	0	0	2	1	1	1
40	10	5	3	22	1	1	3
50	10	29	3	1:50	2	1	9
60	5	2:44	22	11:55	11	1	51
70	3	16:30	1:03	28:34	29	1	55
80	2	16:01	1:36	30:27	21	1	41
90	1	1:10:41			67		
100	1	4:10:43			137		

Table 10
Average branch and bound results for the instances of Table 4, part 3

n	nr	h:mm:ss	min-time	max-time	nodes	min	max
Q_{100}							
31	10	8	2	38	3	1	23
41	10	1:22	30	2:15	24	5	51
51	10	3:52	18	5:01	37	1	57
61	5	3:56	19	10:07	52	33	65
71	3	19:57	19:34	20:22	76	73	79
81	2	35:43	35:01	36:25	81	75	87
91	1	52:30			79		
101	1	1:32:00			137		
$Q_{100,0.2}$							
31	10	3	2	4	1	1	1
41	10	1	0	10	1	1	1
51	10	24	15	1:01	1	1	5
61	5	2:52	28	11:28	14	1	67
71	3	22:03	11:03	32:42	65	21	87
81	2	25:38	15:56	35:21	45	19	71
91	1	55:49			79		
101	1	1:31:25			127		

one branch of the tree – the branch running in line with the optimal solution – will extend over many levels.

We also tried to compute solutions with a given performance guarantee for the examples given in Table 6. For the class $G_{0.5}$ a gap of 1% between upper and lower

Table 11
"Limits" of the branch and bound approach, instances as in Table 5

n	h:mm:ss	Nodes
$G_{0.5}$, 1% gap		
150	16:33	51
200	16:55:39	35
250	29:29:28	23
300	>11 days	>150
$G_{-1/0/1}$, 5% gap		
150	303:40:34	1845

bound seems reasonable. For $G_{-1/0/1}$ we decided to go for a gap of 5%. As we can see in Table 11 there is some hope for $G_{0.5}$ examples even for relatively large sizes if a few branching nodes suffice. If this is not the case the high cost of the bounding procedure gets prohibitive immediately. Obviously there is no hope to get reasonable results for large examples of type $G_{-1/0/1}$ with this approach.

8. Concluding remarks

In view of the computational results presented in the previous sections, we offer the following conclusions.

- The (SDP) approach is very robust and consistently yields bounds where the φ-gap on graphs with nonnegative weights is typically much below the worst case error of 13.9% predicted by Theorem 4.1.
- Combined with a small selection of hypermetric inequalities, the (SDP) approach approximates the max-cut problem quite well, and is feasible for graphs with up to several hundred vertices ($n \approx 400$).
- For smaller problems ($n \leqslant 50$), the cutting plane approach applied at the root node of the branching tree is sufficient to solve the problem, otherwise branching is necessary. The number of nodes in the branching tree is rather small, but the computation time per node can be quite large.
- To generate good feasible solutions it is in general sufficient to compute the optimal solution X of the elliptope. The rows of X form good starting vectors for local improvement heuristics.
- Since solving the relaxation (SDP) is only moderately time consuming this leads to reasonable solutions for arbitrary cost matrices up to dimensions of 1000.
- The algorithm does not make use of any special structure in the input data. For sparse cost matrices the linear programming approach will be more efficient but may require very precise knowledge about the underlying structure.

In summary, we consider the combination of (SDP) with polyhedral methods a strong new tool to solve general quadratic (0,1) problems.

References

[1] F. Barahona, The max-cut problem in graphs not contractible to K_5, Operations Research Letters 2 (1983) 107–111.

[2] F. Barahona, Ground-state magnetization of Ising spin glasses, Physical Reviews B 49 (18) (1994) 12864–12867.

[3] F. Barahona, M. Jünger, G. Reinelt, Experiments in quadratic 0-1 programming, Mathematical Programming 44 (1989) 127–137.

[4] F. Barahona, H. Titan, Max mean cuts and max cuts, Combinatorial Optimization in Science and Technology, 1991, pp. 30–45.

[5] E. Boros, Y. Crama, P.L. Hammer, Chvátal cuts and odd cycle inequalities in quadratic 0-1 optimization, SIAM Journal on Discrete Mathematics 5 (1992) 163–177.

[6] E. Boros, P.L. Hammer, The max-cut problem and quadratic 0-1 optimization; polyhedral aspects, relaxations and bounds, Annals of Operations Research 33 (1991) 151–180.

[7] E. Boros, P.L. Hammer, cut polytopes, boolean quadric polytopes and nonnegative pseudo-boolean functions, Mathematics of Operations Research 18 (1993) 245–253.

[8] E. Boros, P.L. Hammer, X. Sun, The DDT method for quadratic 0-1 minimization, Technical Report RRR 39-89, Rutgers Unversity, 1989.

[9] M. Burkard, An interior point algorithm for solving max-cut problems, Diploma Thesis, Technical University of Graz, 1994.

[10] M.W. Carter, The indefinite zero-one quadratic problem, Discrete Applied Mathematics 7 (1984) 23–44.

[11] C. Delorme, S. Poljak, Complexity of a max cut approximation, European Journal of Combinatorics 14 (1993) 313–333.

[12] C. Delorme, S. Poljak, Laplacian eigenvalues and the maximum cut problem, Mathematical Programming 62 (1993) 557–574.

[13] C. De Simone, The cut polytope and the boolean quadric polytope, Discrete Applied Mathematics 79 (1989) 71–75.

[14] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, G. Rinaldi, Exact ground states of ising spin glasses: new experimental results with a branch-and-cut algorithm, Journal of Statistical Physics 80 (1/2) (1995) 487–496.

[15] C. De Simone, G. Rinaldi, A cutting plane algorithm for the max-cut problem, Optimization Methods and Software 3 (1994) 195–214.

[16] M. Deza, M. Grishuchin, M. Laurent, The hypermetric cone is polyhedral, Combinatorica 13 (1993) 397–411.

[17] M. Deza, M. Laurent, Applications of cut polyhedra I, Journal of Computational and Applied Mathematics 55 (1994) 191–216.

[18] M. Deza, M. Laurent, Applications of cut polyhedra II, Journal of Computational and Applied Mathematics 55 (1994) 217–247.

[19] M. Deza, M. Laurent, Geometry of Cuts and Metrics, Algorithms and Combinatorics, vol. 15, Springer, Berlin, 1997.

[20] M.X. Goemans, D.P. Williamson, 0.878-Approximation algorithms for Max-Cut and Max 2SAT, Proceedings of 26th Annual ACM Symposium on Foundations of Computer Science, Computer Science Press, Rockville, MD, 1994, pp. 2–13; see also Journal ACM 42 (1995) 1115–1145.

[21] P.L. Hammer, Some network flow problems solved with pseudo-boolean programming, Operations Research 13 (1965) 388–399.

[22] P.L. Hammer, P. Hansen, B. Simeone, Roof duality, complementation and persistency in quadratic 0-1 optimization, Mathematical Programming 28 (1984) 121–155.

[23] C. Helmberg, An interior point method for semidefinite programming and max-cut bounds, Doctoral Dissertation, University of Technology Graz, 1994.

[24] C. Helmberg, Fixing Variables in Semidefinite Relaxations, Preprint SC 96-43, Konrad-Zuse-Zentrum Berlin, Takustraße 7, D-14195 Berlin, Germany, 1996.

[25] C. Helmberg, S. Poljak, F. Rendl, H. Wolkowicz, Combining semidefinite and polyhedral relaxations for integer programs, in: E. Balas, J. Clausen (Eds), Proceedings of IPCO 4, Lecture Notes in Computer Science 920, 1995, pp. 124–134.

[26] C. Helmberg, F. Rendl, R.J. Vanderbei, H. Wolkowicz, An interior point method for semidefinite programming, SIAM Journal on Optimization 6 (1996) 342–361.
[27] B. Kalantari, A. Bagchi, An algorithm for quadratic zero-one programs, Naval Research Logistics 37 (1990) 527–538.
[28] F. Körner, An efficient branch and bound algorithm to solve quadratic integer programming problem, Computing 30 (1983) 253–260.
[29] M. Laurent, S. Poljak, The metric polytope, in: E. Balas, G. Cornuejols, R. Kannan (Eds.), Proceedings of IPCO 1992, 1992, pp. 274–286.
[30] M. Laurent, S. Poljak, On a positive semidefinite relaxation of the cut polytope, Linear Algebra and Applications 223/224 (1995) 439–461.
[31] M. Laurent, S. Poljak, On the facial structure of the set of correlation matrices, SIAM Journal on Matrix Analysis and Applications 17 (1996) 530–547.
[32] M. Laurent, S. Poljak, F. Rendl, Connections between semidefinite relaxations of the max-cut and stable set problems, Mathematical Programming 77 (1997) 225–246.
[33] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley, Chichester, 1990.
[34] B. Mohar, S. Poljak, Eigenvalues and the max-cut problem, Czechoslovak Mathematical Journal 40 (115) (1990) 343–352.
[35] P.M. Pardalos, G.P. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming, Computing 45 (1990) 131–144.
[36] P.M. Pardalos, G.P. Rodgers, Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture, Annals of Operations Research 22 (1990) 271–292.
[37] S. Poljak, Polyhedral and eigenvalue approximations of the max-cut problem, in: D. Miklós, G.Halász, L. Lovász, T. Szönyi (Eds.), Sets, Graphs and Numbers, North-Holland, Amsterdam, 1992, pp. 568–581.
[38] S. Poljak, Z. Tuza, On the expected relative error of the polyhedral approximation of the max-cut, Operations Research Letters 16 (1994) 191–198.
[39] S. Poljak, Z. Tuza, Maximum cuts and large bipartite subgraphs, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 20 (1995) 181–244.
[40] S. Poljak, F. Rendl, Node and edge relaxations of the max-cut problem, Computing 52 (1994) 123–137.
[41] S. Poljak, F. Rendl, Nonpolyhedral relaxations of graph bisection problems, SIAM Journal on Optimization 5 (1995) 467–487.
[42] S. Poljak, F. Rendl, Solving the max-cut problem using eigenvalues, Discrete Applied Mathematics 62 (1995) 249–278.
[43] A. Schrijver, personal communication, 1992.
[44] A.C. Williams, Quadratic 0-1 programming using the roof dual with computational results, RUTCOR Research Report 8-85, Rutgers Unversity, 1985.