Solving Max-Cut to Optimality by Intersecting Semidefinite and Polyhedral Relaxations

F. Rendl* G. Rinaldi† A. Wiegele‡

May 6, 2008

Abstract

We present a method for finding exact solutions of Max-Cut, the problem of finding a cut of maximum weight in a weighted graph. We use a Branch-and-Bound setting, that applies a dynamic version of the bundle method as bounding procedure. This approach uses Lagrangian duality to obtain a "nearly optimal" solution of the basic semidefinite Max-Cut relaxation, strengthened by triangle inequalities. The expensive part of our bounding procedure is solving the basic semidefinite relaxation of the Max-Cut problem, which has to be done several times during the bounding process.

We review other solution approaches and compare the numerical results with our method. We also extend our experiments to instances of unconstrained quadratic 0-1 optimization and to instances of the graph equipartition problem.

The experiments show, that our method nearly always outperforms all other approaches. In particular, for dense graphs, where linear programming based methods fail, our method performs very well. Exact solutions are obtained in a reasonable time for any instance of size up to $n = 100$, independent of the density. For some problems of special structure we can solve even larger problem classes. We could prove optimality for several problems of the literature where, to the best of our knowledge, no other method is able to do so.

1 The Max-Cut Problem

The Max-Cut problem is one of the basic NP-hard combinatorial optimization problems and has attracted scientific interest from both the discrete (see, e.g.,

*Alpen-Adria-Universität Klagenfurt, Institut für Mathematik, Universitätsstr. 65-67, 9020 Klagenfurt, Austria, (franz.rendl@uni-klu.ac.at)

†Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" del CNR, viale Manzoni 30, 00185 Roma, Italy, (rinaldi@iasi.cnr.it)

‡Alpen-Adria-Universität Klagenfurt, Institut für Mathematik, Universitätsstr. 65-67, 9020 Klagenfurt, Austria, (angelika.wiegele@uni-klu.ac.at)

[17, 33, 35]) and the nonlinear optimization community (see, e.g., [14, 10, 28]).

The purpose of this paper is twofold. We first give an overview of all relevant exact solution approaches for the Max-Cut problem in Section 3. This part is of survey type, providing a literature overview and a discussion of the practical limitations of these approaches. Secondly, we provide some computational experience with a Branch-and-Bound based method, that solves large size instances of this problem to optimality. Here we call "large" an instance whose size makes it very difficult to solve with the current state-of-the-art methods. The technical details of our approach are given in Sections 4 and 5. Sections 6 and 7 contain a description of the data sets and computational results on a variety of test data. A preliminary version of this work is presented in [42].

There are two equivalent formulations to the problem.

Max-Cut in a graph: Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges with edge weights w_e for $e \in E$. Any bipartition (S, T) of the node set V, where S or T are allowed to be empty, defines a *cut* of G, i.e., the edge set $(S : T) = \{\{i, j\} \in E \mid i \in S, j \notin S\}$. The weight $w(S : T)$ of the cut is defined as

$$w(S : T) = \sum_{e \in (S:T)} w_e.$$

The *Max-Cut problem* calls for a bipartition (S^*, T^*) for which $w(S^* : T^*)$ is maximal. It will be convenient to use matrix notation and introduce the weighted adjacency matrix $A = (a_{ij})$ with

$$a_{ij} = a_{ji} = w_e$$

for edge $e = \{i, j\} \in E$, and $a_{ij} = 0$ if $\{i, j\} \notin E$. We also introduce the Laplacian matrix L associated with A, defined by

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

where e denotes the vector of all ones and $\mathrm{Diag}(v)$ is the operator that maps a n-dimensional vector v into the n-dimensional matrix M, where $M_{ii} = v_i$ and all the off-diagonal components are zero.

If we represent a bipartition (S, T) by its incidence vector $x \in \{-1, 1\}^n$, with $x_i = 1$ if $i \in S$ and $x_i = -1$ otherwise, then it is easy to show that

$$w(S : T) = \frac{1}{4} x^T L x.$$

Hence finding a cut in a graph with maximum weight is equivalent to solving the following quadratic optimization problem.

$$\text{(MC)} \qquad z_{MC} = \max\{x^T L x : \ x \in \{-1, 1\}^n\}.$$

Quadratic 0-1 minimization: Given a matrix Q of order n and an n-dimensional vector c, let

$$q(y) := y^T Q y + c^T y. \tag{1}$$

We consider the following problem.

$$\text{(QP)} \qquad \min\{q(y) : \ y \in \{0,1\}^n\}.$$

It is not difficult to show that solving (QP) is equivalent to solving (MC) (see, for instance, [6]).

To this purpose, define

$$W = \begin{pmatrix} 0 & -(Qe+c)^T \\ -(Qe+c) & Q \end{pmatrix}$$

and consider W to be the adjacency matrix of a graph with node set $V = \{0, 1, \ldots, n\}$. Let x denote the incidence vector of a cut of this graph with value k. Without loss of generality we can assume $x_0 = -1$. Then y, defined as

$$y_i = \frac{1}{2}(x_i + 1), \quad 1 \le i \le n,$$

is a solution of (QP) with objective value $-k$.

Conversely, let L be the Laplace matrix of a given graph and

$$L = \begin{pmatrix} l_{11} & L_{12}^T \\ L_{12} & L_{22} \end{pmatrix}$$

where L_{22} is a $(n-1) \times (n-1)$ matrix. Let y be a solution of (QP) with $Q = -L_{22}$ and $c = L_{12} + L_{22}e$ with value $q(y)$. Then, x defined by

$$x_1 = -1, \ x_i = 2y_{i-1} - 1, \quad 2 \le i \le n$$

is a solution of (MC) with objective value $-q(y)$.

2 A Generic Branch-and-Bound Scheme for (MC)

Even though the description of a Branch-and-Bound (B&B) scheme is rather simple, and can be found in virtually any textbook on enumerative methods for NP-hard problems, we summarize the key features of this procedure. We do this because we are going to compare several different B&B approaches to solve (MC). In order to explain how these approaches differ from each other, we need to describe the basic principle as it is relevant for our purposes. The development of the machinery for linear optimization based B&B has reached a high degree of sophistication, see for instance [2] for a recent summary on branching rules. There are also efficient implementations available, for instance ABACUS [18] and SCIP [1].

The basic step of a Branch-and-Bound procedure is to decompose an instance into two smaller ones where one of the binary decisions involved in the problem

is already taken. In particular, to make such a decomposition, we follow a very simple approach and subdivide the set of feasible solutions $\mathcal{S} = \{-1,1\}^n$ by selecting two vertices i and j and considering the two sub problems

$$\begin{aligned} \mathcal{S}_{join} &:= \{x \in \{-1,1\}^n : \ x_i - x_j = 0\}, \\ \mathcal{S}_{split} &:= \{x \in \{-1,1\}^n : \ x_i + x_j = 0\}. \end{aligned}$$

In the first case, i and j are forced to stay in the same set of the partition (we will say that i and j are forced to be *joined*), while in the second case they must be in separate sets (we will say that i and j are forced to be *split*). A nice feature of Max-Cut allows us to optimize over both these sets by solving again a Max-Cut problem but this time on two graphs of size $n-1$. In the first case the problem is equivalent to finding a maximum cut in a graph obtained from the original one by identifying the two nodes i and j, by removing the loop edge possibly created, and by replacing each pair of parallel edges e and f resulting from the identification with a single edge having weight $w_e + w_f$. The new graph has $n-1$ nodes. Each of its cuts corresponds to exactly one cut of the original graph and the two cuts have the same weight.

In the second case we first apply the following linear transformation to the problem:

$$\tilde{x} = N_j x, \tag{2}$$

where N_j is the identity matrix with the sign of the j-th diagonal component changed to -1. After the transformation, Problem (MC) becomes

$$\max\{\tilde{x}^T \tilde{L} \tilde{x} \mid \tilde{x} \in \{-1,1\}^n\}, \tag{3}$$

where $\tilde{L} = N_j L N_j$. It is easy to verify that $\tilde{x}^T \tilde{L} \tilde{x} = x^T L' x + 4s$, where L' is the Laplacian matrix corresponding to the weighted adjacency matrix $N_j A N_j$ and s is the weight of the cut defined by the bipartition $(\{j\} : V \setminus \{j\})$, i.e., $s = \sum_{kj \in E} a_{kj}$. Therefore, solving (3) amounts to solving a Max-Cut instance on a graph obtained from the original graph by flipping the sign of the weights of the edges incident with node j. After having applied the transformation (2), the set $\mathcal{S}_{split}$ is represented by $\mathcal{S}_{split} := \{\tilde{x} \in \{-1,1\}^n : \ \tilde{x}_i - \tilde{x}_j = 0\}$, so like in the first case, the problem is equivalent to finding a maximum cut in a graph obtained from the original one (but with the sign of some edge weights flipped) by identifying the two nodes i and j.

Every node of the enumeration tree associated with the B&B process is fully characterized by two sets F_{join} and F_{split} containing the pairs $\{i,j\}$ of nodes for which the constraints $x_i - x_j = 0$ and $x_i + x_j = 0$, respectively, are active. Therefore, by (L, F_{join}, F_{split}) we denote the problem

$$(L, F_{join}, F_{split}) \qquad \begin{array}{ll} \max & x^T L x \\ \text{s.t.} & x_i - x_j = 0, \ \text{for } \{i,j\} \in F_{join} \\ & x_i + x_j = 0, \ \text{for } \{i,j\} \in F_{split}. \end{array}$$

Note that the edges whose endpoints are the pairs in $F_{join} \cup F_{split}$ form an acyclic graph.

In order to complete the definition of the B&B process, given an instance of (MC) through the matrix L and the two sets F_{join} and F_{split}, we need the following three procedures.

1. *Upper Bound.* The bounding procedure

$$z_{ub} = \text{upper-bound}(L, F_{join}, F_{split})$$

   determines an upper bound z_{ub} on the optimal value z_{MC} of (MC) for cost matrix L under the constraints defined by the two sets F_{join} and F_{split} (we have seen that this problem is a (MC) on a graph with $n - |F_{join}| - |F_{split}|$ nodes).

2. *Heuristic.* We also need to be able to produce some (good) feasible solution x_f with value $z_f = x_f^T L x_f$ (hopefully) close to z_{MC} for the problem of the previous point:

$$x_f = \text{feasible-solution}(L, F_{join}, F_{split})$$

3. *Branching Pair Selection.* Finally, a branching strategy to determine a pair $\{i, j\}$ of vertices to be split or joined is needed.

Using these three ingredients, we can summarize a generic B&B approach for (MC) in the following scheme.

Generic Branch-and-Bound Algorithm for (MC)

input: L symmetric $n \times n$ matrix
output: $x \in \{-1, 1\}^n$ optimal solution
initialize:
  $z_{ub} = \text{upper-bound}(L, \emptyset, \emptyset)$; (initial upper bound)
  $x_{bk} = (1, \ldots, 1)^T$; (initial cut vector)
  $z_{bk} = x_{bk}^T L x_{bk}$; (best known (bk) solution value)
  $\mathcal{Q} = \{(z_{ub}, (L, \emptyset, \emptyset))\}$; (problem list)
while $\mathcal{Q} \neq \emptyset$
  remove problem $(z, (L, F_{join}, F_{split}))$ from $\mathcal{Q}$ having z maximal;
  compute:
    $z_{ub} = \text{upper-bound}(L, F_{join}, F_{split})$;
    $x_f = \text{feasible-solution}(L, F_{join}, F_{split})$;
  if $x_f^T L x_f > z_{bk}$ **then**
    update (z_{bk}, x_{bk});
    remove all problems $(z', (L, F'_{join}, F'_{split}))$ from $\mathcal{Q}$ with $z' < z_{bk}$;
  if $z_{bk} < z_{ub}$ **then**
    determine the branching pair $\{i, j\}$ for the problem $(z, (L, F_{join}, F_{split}))$;
    set $F'_{join} = F_{join} \cup \{\{i, j\}\}$ and $F'_{split} = F_{split} \cup \{\{i, j\}\}$;
    add $(z_{ub}, (L, F'_{join}, F_{split}))$ and $(z_{ub}, (F_{join}, F'_{split}))$ to $\mathcal{Q}$;
endwhile
return x_{bk};

Contrary to linear optimization based B&B, where branching is a critical issue (see for instance [2]), we concentrate on the bounding procedure as this will turn out to be the only nontrivial part. This is also the distinguishing feature of most of the existing methods to solve (MC), so we first summarize the various ways to get bounds on (MC) in the following section. The bounding procedure used in our algorithm will then be described in detail in Section 4.

Finding good feasible solutions can be done routinely in our case, as any vector $x \in \{-1, 1\}^n$ is feasible. We will briefly touch this issue in Section 5.2. The branching strategy will be described in some detail in Section 5.1. Here we follow essentially the ideas outlined in [27].

3 Some Solution Approaches and their Limits

In this section we recall the most popular and the most recent methods for solving our problem of interest. We sketch the algorithms and summarize their limits. A survey of techniques developed before 1980 can be found in [24].

3.1 LP based relaxations

The solution of (MC) can be found, in principle, by solving a linear program. To express the objective with a linear function we have to represent every solution of the problem with the incidence vector of a cut. In particular, if K is a cut of G, its incidence vector $\chi^K \in \{0, 1\}^m$ is such that its component χ^K_e is equal to 1 if $e \in K$ and is equal to zero otherwise. The *cut polytope* $\mathrm{CUT}(G)$ associated with G is the convex hull of the incidence vectors of all cuts of G. Problem (MC) can now be written as

$$\text{(MC)} \qquad z_{MC} = \max\{4w^T s : s \in \mathrm{CUT}(G)\}, \tag{4}$$

where w is the vector of the weights on the edges. Unfortunately, the constraint set of (4) has a number of inequalities that is way too large to be solved with current LP technology even for graphs of small size. However, by replacing $\mathrm{CUT}(G)$ in (4) with an *LP relaxation*, i.e., by any (more manageable) polytope P containing $\mathrm{CUT}(G)$, we get an upper bound on z_{MC} that can be used in a B&B scheme. Usually, it is required that an integral point contained in P is always the incidence vector of a cut of G. If this is the case, we call P a *valid IP formulation* and (4) can be replaced by the following integer linear program:

$$z_{MC} = \max\{4w^T s : s \in P,\ s \text{ integral}\}.$$

The most used relaxation of $\mathrm{CUT}(G)$ is given by the following system of inequalities defining the so-called *semimetric polytope* $\mathcal{M}(G)$ of the graph G:

$$\begin{array}{ll} \sum_{e \in F} s_e - \sum_{e \in C \setminus F} s_e \leq |F| - 1 & C \in \mathcal{C}, F \subseteq C, |F| \text{ odd} \\ 0 \leq s_e \leq 1 & e \in E, \end{array} \tag{5}$$

where $\mathcal{C}$ is the set of all cycles of G, see for instance [5] or [17]. The inequalities of the first set are called the *cycle inequalities* and are non-redundant (and actually

facet defining) if for $\mathcal{C}$ we take the set of all chordless cycles. The inequalities of the second set are non-redundant (and facet defining) if E is replaced by the set of all edges of G that are not contained in a triangle.

If the graph is complete, the (non-redundant) system (5) becomes

$$\left.\begin{array}{r} s_{ij} + s_{ik} + s_{jk} \leq 2 \\ s_{ij} - s_{ik} - s_{jk} \leq 0 \\ -s_{ij} + s_{ik} - s_{jk} \leq 0 \\ -s_{ij} - s_{ik} + s_{jk} \leq 0 \end{array}\right\} \quad \text{for all triangles } i < j < k \text{ of } G \tag{6}$$

While there are $O(n^3)$ triangles, there are in general exponentially many cycle inequalities. However, in [5] a polynomial time algorithm is given that solves the separation problem for (5), i.e., the problem to find, for any given point $\bar{s} \in \mathbb{R}^m$, a cycle inequality violated by $\bar{s}$ or show that no such an inequality exists.

The number of inequalities of (5) (or of (6)) is too large to be explicitly represented in the LP formulation needed to compute the upper bound. The polynomial time separation algorithm, however, can be effectively used in the following cutting-plane scheme:

Cutting-Plane Algorithm for (MC)

initialize:
  $\mathcal{L} = \{(\ell, \ell_0)\}$ = initial set of inequalities (typically the bounds on the variables);
repeat
  solve $\max\{w^T s : \ell^T s \leq \ell_0 \text{ for } (\ell, \ell_0) \in \mathcal{L}\}$;
  let $\bar{s}$ be the optimal solution;
  find a cycle inequality $\ell^T s \leq \ell_0$ with $\ell^T \bar{s} > \ell_0$;
  if successful **then** add (ℓ, ℓ_0) to $\mathcal{L}$;
until no cycle inequalities are generated;

The optimal solution s^* of an LP relaxation is used for selecting the branching pair $\{i, j\}$, as well as for finding a feasible (good) solution of (MC). A typical heuristic that achieves this result amounts to finding a maximum weight spanning tree in the graph G where each edge e is assigned the weight $|s^* - 0.5|$. The edges of the optimal tree with $s^*_e > 0$ are assigned to the resulting cut, while the other edges are assigned to the complement of the cut. The assigned edges unambiguously determine a bipartition $(S : T)$ and hence a feasible cut.

The optimal solution of an LP relaxation can also be used for *fixing variables*. If $s^*_e = 0$ and $w^T s^* - d_e < z_{bk}$, where z_{bk} is the value of the best known cut in G and $d \in \mathbb{R}^{E(G)}$ is the reduced cost vector, then the variable s_e has value 0 also in the optimal solution, consequently s_e can be fixed to 0. Similarly, if $s_e = 1$ and $w^T s^* + d_e < z_{bk}$, we can fix the variable s_e to 1. Furthermore, we can also fix the variables associated with all the edges that belong to a subgraph induced by the end-nodes of the edges fixed to 0 or 1 as, by the cycle inequalities, their value is readily determined.

The techniques described in this section are the basic tools that have been used, within a B&B scheme of the type described before, in all the computational studies based on LP relaxations (see, e.g., [35] for a recent survey on these methods). The bound obtained by optimizing over the semimetric polytope is, in general, not very strong. On the other hand, the cut polytope has been extensively studied and several families of valid and facet defining inequalities have been characterized (see, e.g., [17] for an extensive survey). Nevertheless most of these results concern the case when the graph G is complete. For the case of a general graph only few facet inducing inequalities are known besides (5). Moreover, it seems very difficult to extend the inequalities for a complete graph to the general case, as complex projection operations would be involved. A possible solution to overcome some of these difficulties, at least for very sparse graphs, is suggested in [30] where the separation procedures are applied to a projection of the fractional point and then the violated inequalities found are lifted up to the original space. This technique, however, was never experimented with so far in actual computations.

The most successful results of the LP relaxations have been obtained for the computation of the state of minimum energy for spin glasses described by the Ising model. The graphs related to these problems are toroidal grids, thus very sparse. For these type of instances the LP based approach appears to be, by far, the method of choice. Choosing the weights randomly from a Gaussian distribution, in [15] the solutions of instances up to 22 500 nodes are reported. For instances with ± 1 objective function coefficients drawn from a uniform distribution, solutions are reported for sizes up to 12 100 nodes in [4].

Limits of this method. The computational results presented in [6] and [4] show, that graphs of any density up to $n = 30$ nodes can be computed in reasonable time. But with an increasing number of nodes, the limits on the density of the graphs decreases rapidly. Graphs with $n = 100$ nodes can only be solved, if the edge density is at most 20%.

Besides the lack of additional inequalities that would strengthen the semimetric relaxation, there is another drawback that prevents LP based methods to attack graphs of moderate density. When in the above cutting-plane algorithm a simplex or a barrier algorithm is used to solve each LP, computation times can really blow up: for example, [20] reports computation times of more than one hour for complete graphs of 150 nodes just to solve the LP relaxation. A different way to compute the relaxation is to keep, as explicit constraints, only a small subset of the inequalities (5) and to dualize all the others that would be used by the cutting-plane algorithm. In [4] only the box constraints of the variables are kept as explicit constraints and the *volume* algorithm is used to solve the Lagrangian dual.

Yet another approach is proposed in [20]. It is assumed that the graph has a node r adjacent to all other nodes. If this is not the case, 0-weighted edges are added to the graph to meet the assumption. Then the following subset of (5) is

considered:

$$\left.\begin{array}{r} s_{rj}+s_{rk}+s_{jk}\leq 2\\ s_{rj}-s_{rk}-s_{jk}\leq 0\\ -s_{rj}+s_{rk}-s_{jk}\leq 0\\ -s_{rj}-s_{rk}+s_{jk}\leq 0\end{array}\right\}\quad \text{for all } \{j,k\}\in E,\ r\neq j<k\neq r. \tag{7}$$

These inequalities define the *r-rooted metric polytope* of G ($\mathcal{M}^r(G)$). The optimization of a linear function over $\mathcal{M}^r(G)$ can be done efficiently, as it amounts to solving a maximum flow problem in a graph derived from G. Therefore, the inequalities (7) are kept explicitly in the constraint set while all the other cycle inequalities are dualized. The Lagrangian dual is then solved with a bundle method pretty much in the same way as described in Section 4.1. In [20] time savings of up to two orders of magnitude with respect to simplex or barrier based cutting-plane algorithms are reported for graphs of up to 150 nodes. The primal and dual infeasibilities of the solutions are comparable with those obtained with the simplex or the barrier algorithm. Experiments to obtain exact solutions for (MC) using this approach have not been carried out. However, despite the remarkable improvements in the computation time of the LP bound, on dense graphs it is not expected that, at present, LP based relaxations can compete with the techniques described in the following sections.

3.2 The basic SDP relaxation

An equivalent formulation of (MC) is given by

$$z_{MC}=\max\{\text{tr } LX : \text{diag}(X)=e,\ \text{rank}(X)=1,\ X\succeq 0\}, \tag{8}$$

(see, e.g., [41]), where X is an $n\times n$ real matrix, tr$A$ denotes the *trace* of the matrix A, i.e., the sum of its diagonal elements, and diag(A) maps a matrix A of order n into the n-dimensional vector made of its diagonal components. It is easy to see that if x is the incidence vector of an $(S:T)$ bipartition, then the matrix xx^T satisfies all the constraints in (8). (Remember that x is a ± 1 incidence vector.) Moreover, if $s\in\mathbb{R}^m$ is the incidence vector of the cut defined by $(S:T)$ and X_{ij} denotes $[xx^T]_{ij}$, then we have

$$s_{ij}=\frac{1-X_{ij}}{2}. \tag{9}$$

By dropping the constraint that imposes X to have rank one, we obtain the following semidefinite relaxation of (MC):

$$z_{SDP}=\max\{\text{tr } LX : \text{diag}(X)=e,\ X\succeq 0\}. \tag{10}$$

This is a semidefinite program (SDP) in the matrix variable X of order n, and n equality constraints. Its dual form

$$\min\{e^Tu : \text{Diag}(u)-L\succeq 0\} \tag{11}$$

was introduced by Delorme and Poljak [16] as the (equivalent) eigenvalue optimization problem

$$\min\{n\lambda_{\max}(L - \mathrm{Diag}(u)) : u \in \mathbb{R}^n, u^T e = 0\}, \tag{12}$$

where $\lambda_{\max}(A)$ is the largest eigenvalue of the matrix A. The primal version (10) can be found in [40]. In [23] it is shown that this relaxation has an error of no more than 13.82%, i.e.,

$$\frac{z_{SDP}}{z_{MC}} \leq 1.1382,$$

provided there are non-negative weights on the edges.

The model (12) is used in [39] as the bounding procedure in a Branch-and-Bound framework.

Limits of this method. This basic SDP bound can be computed efficiently by interior point methods. However, the bound is too weak to be successfully used within a Branch-and-Bound framework, since the progress at each node is disappointingly small and the number of B&B nodes becomes rather large, already for medium sized problems. Graphs up to $n = 50$ nodes can be solved quite efficiently to optimality, but for larger n a solution in reasonable time can only be obtained, for instances where the initial gap is already very small.

3.3 Convex quadratic relaxations

Billionnet and Elloumi [11] consider the following relaxation of (QP). Define for any vector $u \in \mathbb{R}^n$ the Lagrangian

$$q_u(y) := q(y) + \sum_i u_i(y_i - y_i^2) = y^T(Q - \mathrm{Diag}(u))y + (c + u)^T y$$

An equivalent problem to (QP) is

$$(\mathrm{QP}_u) \quad \min\{q_u(y) : y \in \{0,1\}^n\}.$$

Relaxing the integrality constraints in (QP_u), gives the lower bound $\beta(u)$,

$$\beta(u) = \min\{q_u(y) : y \in [0,1]^n\}.$$

If the vector u is chosen, such that $Q - \mathrm{Diag}(u)$ is positive semidefinite, $\beta(u)$ is obtained by solving a convex quadratic problem, which can be done efficiently. Now, u^* is chosen to maximize $\beta(u)$. This gives the "optimal" lower bound β^*, i.e.,

$$\beta^* = \beta(u^*) = \max\{\beta(u) : (Q - \mathrm{Diag}(u)) \succeq 0, u \in \mathbb{R}^n\}.$$

In [11] it is observed that the dual to this SDP is essentially equivalent to the basic Max-Cut relaxation (10), see Section 3.2.

The solution of problem (QP_u) (or (QP_{u^*}), respectively) can be derived by using a solver for convex quadratic 0-1 problems, i.e., a Branch-and-Bound algorithm using $\beta(u)$, the continuous relaxation of QP_u, as a bound.

The computational effort for this algorithm can be summarized as follows:

1. Preprocessing phase: solve an SDP to obtain a vector u^* and a bound β^*.
2. Use a mixed integer quadratic problem solver for solving problem (QP_{u^*}). Even though the computation of the bounds is very cheap, the number of nodes in the Branch-and-Bound tree typically exceeds 100 000 for problems of $n = 100$ variables.

Limits of this method. Quadratic problems with some special structure can be solved up to $n = 100$ variables. But the method is not capable of solving certain classes of Max-Cut instances of this size (for example, graphs with edge weights chosen uniformly from $\{-1, 0, 1\}$).

3.4 SDP with cutting planes

The SDP relaxation introduced in Section 3.2 can be strengthened by requiring X to satisfy the triangle inequalities. By applying (9) to the triangle inequalities (6) we obtain

$$\left.\begin{array}{r} X_{ij} + X_{ik} + X_{jk} \geq -1 \\ X_{ij} - X_{ik} - X_{jk} \geq -1 \\ -X_{ij} + X_{ik} - X_{jk} \geq -1 \\ -X_{ij} - X_{ik} + X_{jk} \geq -1 \end{array}\right\} \quad \text{for all } i < j < k.$$

We collect these inequalities symbolically as

$$\mathcal{A}(X) \leq b,$$

where $\mathcal{A}$ is an operator mapping symmetric matrices of dimension n into $\mathbb{R}^m$, $m = 4\binom{n}{3}$ with the adjoint operator $\mathcal{A}^T$. Hence we get the *strengthened SDP relaxation*

$$z_{mc-met} = \max\{\mathrm{tr}\ LX : \mathrm{diag}(X) = e,\ \mathcal{A}(X) \leq b,\ X \succeq 0\}. \tag{13}$$

This is again a semidefinite program, but it has $4\binom{n}{3}$ triangle inequalities in addition to the n equations fixing the main diagonal of X to e. The computational effort to solve this problem is nontrivial, even for small n like $n \approx 100$, see, e.g., [27].

Helmberg and Rendl [27] apply this semidefinite relaxation with cutting planes (solved by an interior point code) in a Branch-and-Bound scheme. They consider the basic semidefinite relaxation (10), strengthened by some hypermetric inequalities. Inequalities are added while solving the relaxation (i.e., after some Newton steps), as well as after the exact solution of the relaxation has been obtained. Then the optimization process is restarted again.

Later on, Helmberg [25] improved this algorithm by fixing variables. In [26] this approach is further refined.

Although the relaxation produced very tight bounds, the results of the Branch-and-Bound code remained below the expectations of the authors. The number of nodes in the B&B tree is very small, but the computation time per node may be rather large.

Limits of this method. Most graphs with up to $n = 50$ nodes can be solved in the root node of the Branch-and-Bound tree. Instances up to the size $n = 100$ can still be solved, but the computational effort may be very high. For graphs with more than 100 nodes this algorithm is too slow to be practical.

3.5 Further approaches

Branch-and-Bound with second-order cone programming. Kim and Kojima [32], and later on Muramatsu and Suzuki [36] use a second-order cone programming (SOCP) relaxation as bounding procedure in a Branch-and-Bound framework to solve Max-Cut problems. However, the basic SDP relaxation (see Section 3.2) performs better than their SOCP relaxation and the method works only for sparse graphs.

Limits of this method. The algorithm is capable of solving very sparse instances only. The largest graphs for which solutions are reported are random graphs (weights between 1 and 50) of $n = 120$ nodes and density $d = 2\%$, and graphs that are the union of two planar graphs up to $n = 150$, $d = 2\%$.

Branch-and-Bound with preprocessing. Pardalos and Rodgers [37], [38] solve (QP) by Branch-and-Bound using a preprocessing phase where they try to fix some of the variables. The test on fixing the variables is based on the gradient of (1), which reads $2Qy+c$, and exploits the fact that if y^* is the global solution of (QP), then y^* is also optimal for the linear program

$$\min\{(2Qy^* + c)^T y : y \in \{0,1\}^n\}.$$

Limits of this method. Similar to the cutting-plane technique in [6], dense instances up to $n = 30$ and sparse instances up to $n = 100$ can be computed. Special classes of instances can be solved efficiently up to $n = 200$. These instances have off-diagonal elements in the range $[0, 100]$ and diagonal elements lying in the fixed interval $[-I, 0]$, for the case $I = 63$ (the density is 100%). For other values of I, the problem may become much more difficult to solve. However, the method fails for general dense problems already with $n = 50$ variables.

4 The Bounding Procedure in our Branch-and-Bound Framework

In the previous section we described the most popular bounding procedures for Max-Cut, along with their limitations. In this section we explain the bounding procedure used in our approach. The explanation of step 2 (heuristics) and step 3 (branching rules) can be found in the subsequent section.

We are going to use the semidefinite relaxation (13). Instead of solving this relaxation with a limited number of inequality constraints by interior point methods, as done in [27], we use the bundle approach suggested in [19].

The basic semidefinite relaxation (10) can be solved with reasonable effort for rather large problem sizes. However, using this relaxation as bounding procedure in a Branch-and-Bound framework turns out to be too weak. On the other hand, solving the strengthened relaxation (13) directly is intractable for problems of size $n \geq 100$, since the number of inequalities is roughly $\frac{2}{3}n^3$.

As already mentioned in Section 3.4, [27] developed a machinery for getting solutions of this relaxation by using an interior point algorithm applied to a limited number of triangle inequalities. The number of inequalities to be included, say m, strongly affects the computational effort, since a dense matrix of order m has to be stored and factorized throughout the algorithm. This puts a severe limit to the number m of triangle inequalities to be included explicitly.

Although the decrease of the bound after the inclusion of some of the triangle inequalities is significant, the computational overhead is prohibitive for larger instances. Subsection 4.2 describes the situation and compares it to the method used in the Branch-and-Bound framework of our algorithm.

Instead of maintaining a limited set of triangle inequalities explicitly in the SDP, [19] apply the bundle method to the Lagrangian dual, obtained by dualizing the triangle constraints. We briefly describe the relevant details of this approach, and our modifications for use in the Branch-and-Bound setting.

4.1 Using Lagrangian duality for solving the strengthened SDP relaxation

The set $\mathcal{E} := \{X : \mathrm{diag}(X) = e,\ X \succeq 0\}$ defines the feasible region of (10). Therefore, (13) can be written compactly as

$$z_{mc-met} = \max\{\langle L, X\rangle : X \in \mathcal{E}, \mathcal{A}(X) \leq b\}. \tag{14}$$

Let us introduce the Lagrangian

$$\mathcal{L}(X, \gamma) := \langle L, X\rangle + \gamma^T(b - \mathcal{A}(X)) \tag{15}$$

and the dual functional

$$f(\gamma) := \max_{X\in\mathcal{E}} \mathcal{L}(X, \gamma) = b^T\gamma + \max_{X\in\mathcal{E}} \langle L - \mathcal{A}^T(\gamma), X\rangle. \tag{16}$$

The problem now consists in minimizing f over $\gamma \geq 0$:

$$z_{mc-met} = \min_{\gamma \geq 0} f(\gamma).$$

The function f is well-known to be convex but non-smooth. Evaluating f for some $\gamma \geq 0$ amounts to solving a problem of type (10), which can be done easily for problem sizes of our interest. We use a primal-dual interior-point method to solve it, which also provides an optimality certificate X_γ, u_γ (optimal solutions to (10) and (11)). The primal matrix X_γ will turn out to be useful in our algorithmic setup. We have, in particular that

$$f(\gamma) = \mathcal{L}(X_\gamma, \gamma).$$

Moreover, a subgradient of f at γ is given by $b - \mathcal{A}(X_\gamma)$.

Dualizing all triangle constraints would result in a dual problem of dimension roughly $\frac{2}{3}n^3$, we prefer a more economical approach where inequalities are included only if they are likely to be active at the optimum.

Let I be a subset of the triangle inequalities, hence $\mathcal{A}_I(X) \leq b_I$. We also write γ_I for the variables dual to the inequalities in I. Setting the dual variables not in I to zero, it is clear that for any I and any $\gamma_I \geq 0$, we have

$$f(\gamma_I) \geq z_{mc-met} \geq z_{MC},$$

hence $f(\gamma_I)$ is an upper bound on the optimal value of Max-Cut.

Approximating the value z_{mc-met} therefore breaks down into the following two independent tasks:

1. Identify a subset I of triangle inequalities.
2. For a given set I of inequalities, determine an approximate minimizer $\gamma_I \geq 0$ of f.

The second step can be carried out with any of the subgradient methods for convex non-smooth functions. For computational efficiency we use the bundle method with a limit on the number of function evaluations.

Carrying out the first step is less obvious. We are interested in constraints which are active at the optimum, but this information is in general not available. Therefore we use the optimizer X_{γ_I}, corresponding to an approximate minimizer γ_I of f, and add to the current set I of constraints the t triangle inequalities most violated by X_{γ_I}. (Here t is a parameter which is dynamically chosen.) Thus we can identify promising new inequalities to be added to I.

A dual multiplier close to zero is an indication that the constraint is unlikely to be binding at the optimal solution. Therfore, we remove any constraint from I where the dual multiplier is close to zero. We iterate this process of selecting and updating a set of triangle inequalities, and then solving the respective relaxation, as long as the decrease of the upper bound is sufficiently large. An informal description of our bounding procedure therefore goes as follows.

The Bounding Procedure

solve (10) yielding X and an upper bound $f(0)$;
select a set I of triangle inequalities violated by X;
while upper bound decreases significantly
  use the bundle method to obtain an approximate minimizer γ_I of f on I;
  remove constraints from I where $\gamma_i \approx 0$;
  add new constraints to I, violated by X_{γ_I};
endwhile

Remarks:

- In addition to the triangle inequalities, other constraints could be used to tighten the relaxation (see [27]). However, using only triangle inequalities led to a satisfactory behavior of our algorithm and therefore we abandoned the option of considering other inequalities in favor of computational efficiency and simplicity.

- The (exact) minimizer $\gamma^* \geq 0$ of $f(\gamma)$ is difficult to reach using the bundle method. This is illustrated in Figure 1 and will be explained in more detail in Section 4.2. In particular, see also [19], the improvement in the minimization process of f is biggest in the first iterations, with a strong tailing-off effect. Detecting this phenomenon is useful in a Branch-and-Bound setting, where early termination of the bound computation can provide a substantial overall speed-up.

- Finally, we point out again that the main computational effort in the bounding process is solving (10).

4.2 Comparing two methods for solving the strengthened SDP relaxation

As reported in [19], this approach provides the currently strongest bounds at reasonable computational cost for Max-Cut. The number of function evaluations (i.e., solving (10)) is surprisingly small.

To illustrate the practical behavior of this algorithm compared to the SDP based method of [27], we plot in Figure 1 the decrease of the bound over time for both approaches. We took an instance from the Beasley collection (beasley250-6) with $n = 250$, see Section 7.2. The optimal value is $z_{MC} = 43\,931$ (bottom line in the figure). We also computed z_{mc-met} and found that $z_{mc-met} \leq 44\,095$ (dashed horizontal line). The topmost curve in the figure shows the progress of the upper bound as more and more constraints are added and the resulting SDP is solved by interior-point methods. The second curve shows the development of the upper bound during the iterative scheme described above. It should be clear that this variant is substantially more efficient in approaching z_{mc-met}.

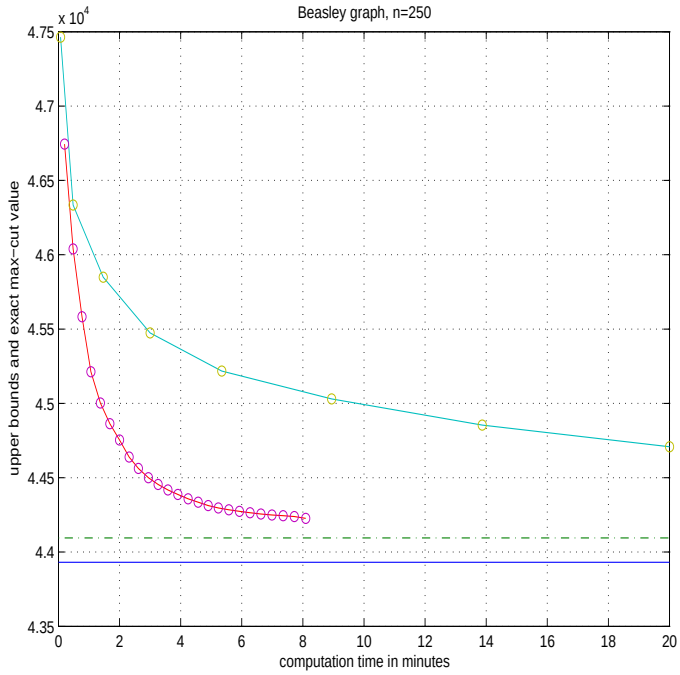

Figure 1: Beasley graph with $n = 250$. The optimal value of Max-Cut is 43 931, $z_{mc-met} \approx 44\,095$.

The figure also suggests that the true value of z_{mc-met} is hard to reach for both methods. Finally, we also see that there is some room to experiment with early termination of the iterations: in case it is clear that the value of the best cut can not be reached (in a subproblem of the Branch-and-Bound process), it may be worthwhile to stop iterating and generate new subproblems (thereby saving computation time). On the other hand, carrying on with the iterative process may be useful in case the current progress of the bound suggests that the value of the best known cut is within reach, so that the branching node under consideration may be eliminated without generating additional branching nodes. In our implementation, we applied a dynamic strategy to control the number of iterations.

5 Branching Rules and Heuristics

In this section we explain how we carry out step 2 (heuristics) and step 3 (branching rules) of the generic Branch-and-Bound algorithm of Section 2.

5.1 Branching strategies

There are several natural choices for a pair i, j of vertices for branching.

Easy first: A first idea is to branch on node pairs where the decision seems to be obvious. We choose i and j such that their rows of X are 'closest' to a $\{-1, 1\}$ vector, i.e., they minimize $\sum_{k=1}^{n}(1 - |X_{ik}|)^2$. We may assume, that for these two very well articulated nodes the value $|X_{ij}|$ is also very large. Setting X_{ij} opposite to its current sign should lead to a sharp drop of the optimal solution in the corresponding sub tree. Hoping that the bound also drops as fast, we should be able to cut off this sub tree quickly. This rule has been introduced in [27] and called R2.

Difficult first: Another possibility for branching is, to fix the hard decisions first. We branch on the node pair $\{i, j\}$ which minimizes $|X_{ij}|$. This means that we fix the most difficult decisions and hope that the quality of the bound gets better fast and that the sub problems become easier. Following [27] we call this rule R3.

Strong branching: Motivated from linear programming based Branch-and-Bound, one can also experiment with *strong branching*, meaning that we do a forecast on potential branching edges in order to branch on the edge that seems to bring the best progress, see for instance [2]. Ideally, we might even be able to fathom a node (before we actually added it to the queue $\mathcal{Q}$), i.e., we would only have to add the other node to the branching tree.

If we consider the decision to join nodes i and j, we would have to solve the SDP

$$z_{i\sim j} = \max\{\mathrm{tr}LX : X \in \mathcal{E}, \mathcal{A}(X) \leq b, X_{ij} = 1\}. \tag{17}$$

It turns out that this problem can be approximately solved using our general bounding procedure by simply taking the Lagrangian dual with respect to the new equation $X_{ij} = 1$. Unfortunately, the computational effort to restart the optimization is substantially bigger than in LP-based B&B. Our various computational experiments with strong branching showed that the overall number of nodes could be reduced, but the overhead in computation time led to an overall inferior performance.

Depending on the class of problems, either rule R2 or R3 was more efficient.

5.2 Generating feasible solutions

Generating feasible solutions is done iteratively, basically in three steps:

1. Apply the Goemans-Williamson hyperplane rounding technique [23] to the primal matrix X obtained from solving the SDP during the bundle iterations. This gives a bipartition-vector $\bar{x}$.

2. The cut $\bar{x}$ is locally improved by checking all possible moves of a single vertex to the opposite set of the partition. This gives a new bipartition-vector $\tilde{x}$.

3. Move the rounding matrix X towards a 'good vertex' by using a convex-combination of X and $\tilde{x}\tilde{x}^T$. With this new matrix go to 1, and repeat as long as a better cut is found.

The last step of biasing X towards a good cut matrix $\tilde{x}\tilde{x}^T$ turned out to be quite helpful in practise. Using this heuristic, the optimal cut is already found at the root node of the B&B tree for most of the instances. The computational effort of this step is neglegible, compared to the bound computation.

6 Random Data for (MC) and (QP)

In this section some random data for presenting numerical results of our algorithm are specified. All the data sets can be downloaded from

`http://biqmac.uni-klu.ac.at/biqmaclib.html`.

These instances are taken from various sources. Here we provide some of the characteristics of the data sets.

6.1 Max-Cut instances

6.1.1 Rudy-generated instances.

The first group of instances follows [27] and consists of random graphs (of specified edge density) with various types of random edge weights. All graphs were produced by the graph generator *Rudy* [43]. For a detailed description and a list of the Rudy-calls the reader is referred to [44]. We generated ten instances of size $n = 100$ of the following types of graphs:

$G_{0.5}$: unweighted graphs with edge probability 1/2.

$G_{-1/0/1}$: weighted (complete) graphs with edge weights chosen uniformly from the set $\{-1, 0, 1\}$.

$G_{[-10,10]}$: graphs with integer edge weights chosen from the interval $[-10, 10]$ and density $d \in \{0.5, 0.9\}$.

$G_{[0,10]}$: graphs with integer edge weights chosen from the interval $[-10, 10]$ and density $d \in \{0.5, 0.9\}$.

6.1.2 Applications in Statistical Physics: Ising instances.

We also consider test-problems of Frauke Liers [personal communication, 2005] from applications in Statistical Physics. Two classes of instances are considered:

1. two- and three-dimensional grid graphs, with Gaussian-distributed weights (zero mean and variance one).

2. dense Ising instances (one dimensional Ising chain), i.e., complete graphs with a certain structure. These instances are obtained in the following way: all nodes lie evenly distributed on a cycle. The weights w of the edges depend on the Euclidean distance between two nodes and a parameter σ, such that the following proportion holds:

$$w_{ij} \sim \frac{\epsilon_{ij}}{(r_{ij})^{\sigma}}$$

   where ϵ_{ij} is chosen according to a Gaussian distribution with zero mean and variance one and r_{ij} is the Euclidean distance between nodes i and j. In our experiments the exponent $\sigma \in \{2.5,\ 3\}$. For a more detailed study of the one dimensional Ising chain we refer to the dissertation of Frauke Liers [34] and the references therein.

6.2 Instances of (QP)

Pardalos and Rodgers ([37]) have proposed a test problem generator for Unconstrained Quadratic Binary Programming. Their procedure generates a symmetric integer matrix Q to define the objective function for (QP), with the linear term c represented by the main diagonal of Q, and has several parameters to control the characteristics of the problem, namely:

n: the number of variables

d: the density, i.e., the probability that a nonzero will occur for any off-diagonal coefficient (q_{ij})

c^-: the lower bound of the diagonal coefficients (q_{ii})

c^+: the upper bound of the diagonal coefficients (q_{ii})

q^-: the lower bound of the off-diagonal coefficients (q_{ij})

q^+: the upper bound of the off-diagonal coefficients (q_{ij})

s: a seed to initialize the random number generator

q_{ii} drawn from a discrete uniform distribution in the interval $[c^-, c^+]$, $i = 1, \ldots, n$
$q_{ij} = q_{ji}$ drawn from a discrete uniform distribution in the interval $[q^-, q^+]$, $1 \leq i < j \leq n$.

Several test problems generated this way are provided in the OR-library [7] or [8]. We have chosen all the problems of sizes of our interest, which are the data sets `bqpgka`, due to [21] and `bqp100` and `bqp250`, see [9].

Furthermore, in [11] the sets `c` and `e` of `bqpgka` are extended. We call these instances `bqpbe`.

The characteristics of the test problems are as follows:

`bqpgka`

	n	d	c^-	c^+	q^-	q^+
`bqpgka, set a`	$30, \ldots, 100$	$0.0625, \ldots, 0.5$	-100	100	-100	100
`bqpgka, set b`	$20, \ldots, 120$	1.0	0	63	-100	0
`bqpgka, set c`	$40, \ldots, 100$	$0.1, \ldots, 0.8$	-100	100	-50	50
`bqpgka, set d`	100	$0.1, \ldots, 1.0$	-75	75	-50	50
`bqpgka, set e`	200	$0.1, \ldots, 0.5$	-100	100	-50	50

`bqpbe`

Size ranging from $n = 100$ to $n = 250$ nodes; density ranging from $d = 0.1$ to $d = 1.0$; $c^- = -100$; $c^+ = 100$; $q^- = -50$ and $q^+ = 50$.

`beasley`

Two sizes of $n = 100$ and $n = 250$ nodes; $d = 0.1$; $c^- = -100$; $c^+ = 100$; $q^- = -100$ and $q^+ = 100$.

6.3 Comparing (MC) and (QP) instances

Looking at the existing computational studies on Max-Cut, it is striking that all approaches seem to have a harder time with (MC)-type instances, while (QP)-type instances look more manageable. This looks like a paradox since, as it was shown in Section 1, (MC) and (QP) are equivalent.

We will now take a closer look at this issue. We believe that one possible reason of the difference comes from the way people tend to construct reasonably difficult instances to test their algorithms.

For those who work with MC-type instances it is quite natural to consider as a typical difficult problem the one given by a random graph with edge weight $= 1$ and an edge probability $= \frac{1}{2}$.

In the (QP) case, the situation is less clear. If all the data are nonnegative, then the optimum is $x = 0$. If all the data are nonpositive then the optimum is $x = e$. Therefore, it makes sense (as is done by the Pardalos and Rodgers generator ([37])) to generate instances where the data are randomly chosen with mean $= 0$.

In Figure 2 we compare two 'random' instances from both classes. To be specific, we generated a random graph (edge weight $= 1$, edge probability $= \frac{1}{2}$) on $n = 25$ vertices, and a random instance of QP of equivalent size (all entries in Q and c are randomly drawn from $[-100, 100]$). We enumerated all solutions (roughly 16×10^6 of them), sorted them by objective value, and normalized these values to lie in the interval $[0, 1]$. The figure clearly shows that the random (MC) instance should be much harder for maximization, as many more solutions are within only a tiny fraction of the optimal solution. In the figure on the right we zoom the picture to the 10 000 best solutions in both cases, and here the difference becomes even more evident.

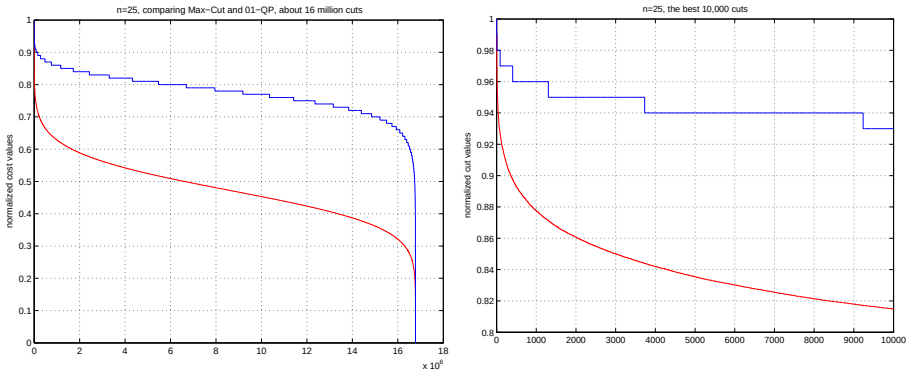

Figure 2: Random data ($n = 25$), normalized cost values for an unweighted random graph and a random QP instance (left plot), and zooming in to the 10 000 best solutions (right plot).

Extrapolating from this small example to larger ones, it should be clear that the (QP) instances are more or less symmetric with respect to maximization and minimization, while the (MC) instances have 'high mass' of good solutions concentrated around the maximal solution. Identifying (and proving) global optimality in the latter case should therefore be expected to be much more difficult than in the (QP) case. This is confirmed in all our computational experiments reported in the following section. It is also consistent with the computational results published in the literature.

Finally, this figure also suggests that having a small initial gap does not necessarily imply that the problem can be solved 'easily'. This will be illustrated also in the computational results given in Table 2, where the initial gap and the computation times are provided.

7 Numerical Results

We implemented the algorithm in C. It is publicly usable as "Biq Mac" – a solver for *bi*nary *q*uadratic and *Ma*x-*C*ut problems at

`http://biqmac.uni-klu.ac.at/`.

If not stated otherwise, test runs were performed on a Pentium IV with 3.6 GHz and 2 GB RAM, operating system Linux. For a more detailed study of the numerical results the reader is referred to [44].

Before we present our computational results, we give in Table 1 a rough overview of the capability of the approaches presented in Section 3. We use the following abbreviations.

LP: Linear programming based Branch-and-Bound, see Section 3.1.

V: Linear programming combined with the volume algorithm, see Section 3.1.

EO: An exact approach using eigenvalue optimization based on (12), see Section 3.2.

QP: The recent work of Billionnet and Elloumi [11] based on convex quadratic optimization, see Section 3.3.

SDPMET: SDP combined with triangle inequalities and solving (13) by an interior point algorithm, see Section 3.4.

PP: Method of Pardalos and Rodgers [37], see Section 3.5.

	LP	V	EO	QP	SDPMET	PP	Biq Mac
quadr 0-1, $n = 100, d = .1$	✔	✔	☹	✔	☕	✔	✔
quadr 0-1, $n = 250, d = .1$	?	?	☹	☹	☹	☹	☕
2-dim. torus, $n = 20 \times 20$	✔	✔	☹	☹	☹	?	🚲
3-dim. torus, $n = 7 \times 7 \times 7$	✔	✔	☹	☹	☹	?	☕
$G_{0.5}, n = 100$	☹	?	☹	🚲	🚲	?	✔
$G_{-1/0/1}, n = 100$	☹	?	🚲	🚲	🚲	?	✔

Table 1: Who can do what?

We consider different types of instances and use the following symbols. A ✔ means, that the approach can solve instances of this type in a routine way. A ☕ indicates that one can have (at least) one cup of coffee while waiting for the solution and maybe there are instances that cannot be solved at all. The 🚲 suggests to have some holidays and come back in a couple of days to see whether the job is finished, and the ☹ indicates that the chances for solving the problem with this method are very low. If we do not know, whether an algorithm can solve certain classes of instances or not, we indicate this with a question mark. Most likely, though, we could place ☹ instead of a question mark.

7.1 Numerical results of Max-Cut instances

7.1.1 Rudy-generated instances.

Table 2 lists the computation times (minimum, average and maximum), the number of nodes (minimum, average, maximum) of the resulting Branch-and-Bound tree and the relative gap in the root node (minimum, average, maximum). The branching rule used for these kind of instances is R2.

The average computation time for all kinds of instances is approximately one hour. Nevertheless, some instances may be solved within minutes whereas for others it could take more than three hours. We also want to point out that, for example, the $G_{0.5}$ instances have a much smaller gap in the root node than the $G_{-1/0/1}$ instances, while the number of nodes in the B&B tree is roughly the same. This means that even if the initial gap is already very small, it can

	n	d	time (h:min) min	avg	max	nodes in B&B tree min	avg	max	initial gap (%) min	avg	max
$G_{0.5}$											
	100	0.5	5	50	3:44	65	610	2925	0.48	0.79	1.20
$G_{-1/0/1}$											
	100	0.99	7	56	2:31	79	651	1811	3.53	6.03	9.26
$G_{[-10,10]}$											
	100	0.5	9	38	1:13	97	435	815	2.94	5.26	8.02
	100	0.9	5	57	3:12	51	679	2427	1.63	5.58	8.52
$G_{[1,10]}$											
	100	0.5	7	48	2:02	111	576	1465	0.49	0.91	1.21
	100	0.9	12	40	1:26	155	464	1007	0.40	0.53	0.62

Table 2: Biq Mac results for Max-Cut problems. For each problem type, 10 instances were solved. Run times on a Pentium IV, 3.6 GHz, 2GB RAM. "initial gap" indicates the relative gap in the root node of the B&B tree in %.

be hard to identify the optimal solution if there are many solutions close to the optimal one (as explained in Section 6.3).

The results show that on these classes of instances we outperform all other solution approaches known so far. The currently strongest results on these graphs are due to Billionnet and Elloumi [11] (details about their algorithm are given in Section 3). They are not able to solve instances $G_{-1/0/1}$ of size $n = 100$ at all. Also, they could solve only two out of ten instances of $G_{0.5}$, $n = 100$.

7.1.2 Applications in Statistical Physics: Ising instances.

As explained in Section 6.1.2, we consider two kind of Ising instances: toroidal grid graphs and complete graphs.

Instances of the first kind can be solved efficiently by an LP-based Branch-and-Cut algorithm (see [35]). The computation times of this LP-based method and of our algorithm are reported in Table 3. As it can be seen, on these sparse instances the LP-based method outperforms our algorithm. However, we find a solution within a gap of 1% in reasonable time for all these samples.

The run time of the Branch-Cut-and-Price algorithm ([34]) developed for the second kind of problems depends strongly on the parameter σ. For σ close to zero, we have a complete graph with Gaussian-distributed weights. But for σ chosen suitably large, some of the edges become 'unimportant' and the technique of fixing variables, described in Section 3.1, works very well for these graphs. In Table 4 the computation times of [34] and our algorithm are given. For $\sigma = 3.0$, roughly speaking we have the same computation times. But for $\sigma = 2.5$, the Branch-Cut-and-Price algorithm already takes more than 20 hours for instances of size $n = 150$, whereas our algorithm needs almost similar computation times as in the $\sigma = 3.0$ case.

For both kind of instances we used branching rule R3.

Problem number	n	[35] time	Biq Mac time	nodes	gap (%)
2 dimensional					
g10_5555	100	0.15	10.12	1	
g10_6666	100	0.14	15.94	1	
g10_7777	100	0.18	14.89	1	
g15_5555	225	0.44	304.03	3	0.0028
g15_6666	225	0.78	359.87	3	0.0043
g15_7777	225	0.67	346.89	3	0.0165
g20_5555	400	1.70	6690.99	9	0.0611
g20_6666	400	3.50	35205.95	45	0.2489
g20_7777	400	2.61	8092.80	11	0.1245
3 dimensional					
g5_5555	125	2.68	18.01	1	
g5_6666	125	3.29	24.52	1	
g5_7777	125	3.07	26.00	1	
g6_5555	216	20.56	280.85	3	0.0032
g6_6666	216	37.74	2025.74	19	0.3648
g6_7777	216	27.30	277.95	3	0.0146
g7_5555	343	95.25	432.71	1	
g7_6666	343	131.34	550.12	1	
g7_7777	343	460.01	117782.75	243	0.6879

Table 3: Test runs on torus graphs with Gaussian distribution. The Branch-and-Cut algorithm [35] runs on a 1.8 GHz PC, Biq Mac runs on a Pentium IV, 3.6 GHz. Times are given in seconds. Column "nodes" lists the number of nodes in the resulting B&B tree, and column "gap (%)" indicates the relative gap in the root node of the B&B tree in %.

Problem number	n	[34] time	Biq Mac time	Problem number	n	[34] time	Biq Mac time
$\sigma = 3.0$				$\sigma = 2.5$			
100_5555	100	4:52	1:36	100_5555	100	18:22	1:32
100_6666	100	0:24	0:34	100_6666	100	6:27	1:06
100_7777	100	7:31	0:48	100_7777	100	10:08	0:47
150_5555	150	2:36:46	4:38	150_5555	150	21:28:39	4:25
150_6666	150	4:49:05	3:55	150_6666	150	23:35:11	5:39
150_7777	150	3:48:41	6:06	150_7777	150	31:40:07	9:19
200_5555	200	9:22:03	10:07	200_5555	200	–	10:05
200_6666	200	32:48:03	18:53	200_6666	200	–	17:55
200_7777	200	8:53:26	22:42	200_7777	200	–	21:38
250_5555	250	21:17:07	1:46:29	250_5555	250	–	3:00:28
250_6666	250	7:42:25	15:49	250_6666	250	–	1:17:04
250_7777	250	17:30:13	57:24	250_7777	250	–	1:10:50
300_5555	300	17:20:54	2:20:14	300_5555	300	–	6:43:47
300_6666	300	10:21:40	1:32:22	300_6666	300	–	9:04:38
300_7777	300	18:33:49	3:12:13	300_7777	300	–	13:00:10

Table 4: Test runs on Ising instances (complete graphs). Branch-Cut-and-Price [34] runs on a 1.8 GHz PC, Biq Mac runs on a 3.6 GHz PC. Times in hours:minutes:seconds. The relative gap in the root node of the B&B tree is at most 0.1%, the number of nodes in the tree ranges from 3 to 139.

7.2 Numerical results of (QP) instances

In this section we report the results for the instances derived from (QP). Best known lower and upper bounds for `bqpgka` and `beasley` data are reported at the pseudo-Boolean web-site [12]. Our results are as follows:

`bqpgka`

Set a. All problems are solved in the root node of the Branch-and-Bound tree.

Set b. These instances could all be solved, but were extremely challenging for our algorithm. The reason is, that the objective value in the Max-Cut formulation is of magnitude 10^6, and therefore even a relative gap of 0.1% is not sufficient to fathom the node. However, if we allow a relative error of at most 0.1%, we can solve all problems in the root node of the Branch-and-Bound tree.

Set c. Also these instances were solved in the root node of the Branch-and-Bound tree.

Set d. These instances could be solved within at most 7 minutes.

Set e. The instances with 10, 20, 30 and 40% density could all be solved within 2 hours of computation time. The instance with density $d = 50\%$ has been solved after 35 hours. According to [12], these problems have not been solved before. The branching rule we used for solving these instances is R2.

`bqpbe`

We report the results of [11] and our results in Table 5. As it is shown in this table, [11] could not solve all out of the ten problems from the $n = 120$ variables and 80% density instances on, whereas our method still succeeded in solving them all. From the instances $n = 150$, $d = 80\%$, the convex-quadratic approach failed to solve any instance within their time limit of 3 hours. We still managed to obtain solutions to all of these instances (although for one graph it took about 54 hours). We applied branching rule R2 to solve these problems.

`beasley`

Solving the 10 problems of size $n = 100$ can be done in the root node within one minute. For the set of problems of size $n = 250$, only two out of the ten problems have been solved before, as reported by Boros et al. [12]. For the other eight problems we could prove optimality for the first time. Eight out of the ten instances were solved within 5 hours, the other two needed 15 and 80 hours, respectively. Since most of these instances have been solved for the first time, we report in Table 6 the optimal values together with the run times, the number of nodes in the B&B tree, and

		[11]				Biq Mac			
			CPU time (sec)				CPU time (sec)		
n	d	solved	min	avg.	max	solved	min	avg.	max
100	1.0	10	27	372	1671	10	86	178	436
120	0.3	10	168	1263	4667	10	29	162	424
120	0.8	6	322	3909	9898	10	239	1320	3642
150	0.3	1		6789		10	1425	2263	2761
150	0.8	0		–		10	1654	1848	2133
200	0.3	0		–		10	7627	37265	193530
200	0.8	0		–		10	5541	47740	148515
250	0.1	0		–		10	12211	13295	16663

Table 5: Comparison between [11] and Biq Mac. Computation times of the convex-quadratic algorithm were obtained on a laptop Pentium IV, 1.6 GHz (time limit 3 hours), our results were computed on a Pentium IV of 3.6 GHz.

the relative gap in the root node (in %). The branching strategy used for these instances is R3.

Bounds of the `beasley` and some of the `bqpgka` data sets have also been calculated by Boros, Hammer, Sun, and Tavares [13]. They use the so-called iterated roof dual as bounding routine and report that, applying this bound computation in a Branch-and-Bound framework, they could solve the four sparsest instances of the `bqpgka-d` data sets and two of the instances `beasley-250`. However, on most instances their bound behaves worse than the basic SDP bound and therefore, using it within a B&B scheme, might not lead to satisfying results.

8 Extension to the equipartition problem

Simple modifications can make our algorithm work for solving related problems. In this section we show such a modification that allows solving the equipartition problem using Biq Mac.

Consider the problem of bisecting a graph such that the sum of the weights on the edges that are cut is minimum and the two sets of the partition have the same cardinality. Given a graph $G = (V, E)$ with $|V|$ even, and edge weights w_{ij}, the minimum weight equipartition problem reads

$$z_{equi} = \min\{\frac{1}{2}\sum_{\{i,j\}\in E} w_{ij}(1 - x_i x_j) : \sum_{i=1}^{n} x_i = 0, x \in \{-1, 1\}^n\} \tag{18}$$

By setting $X := xx^T$, the following natural semidefinite relaxation arises:

$$z_{equi-rel} = \{\min \frac{1}{4}\mathrm{tr}LX : \mathrm{tr}JX = 0, \mathrm{diag}(X) = e, X \succeq 0\}, \tag{19}$$

Problem number	solution	Biq Mac time (h:min:sec)	B&B nodes	gap (%)
250-1	45607	2:34:31	37	0.4357
250-2	44810	1:20:23	19	0.5647
250-3	49037	1:21:13	19	0.1395
250-4	41274	1:15:23	17	0.3927
250-5	47961	1:29:24	21	0.3462
250-6	41014	14:35:02	223	1.0252
250-7	46757	2:31:12	37	0.4380
250-8	35726	88:55:05	4553	2.1931
250-9	48916	3:12:57	47	0.7808
250-10	40442	4:34:40	63	0.6178

Table 6: Test runs on the $n = 250$ QP instances of the OR-library [7]. The density is $d = 0.1$. Column "time" gives the run times on a Pentium IV, 3.6 GHz. Column "B&B nodes" lists the number of nodes in the resulting B&B tree and column "gap (%)" indicates the relative gap in the root node of the B&B tree in %.

where J is the matrix of all ones.

Let A be the adjacency matrix of an unweighted graph. If solving the Max-Cut problem of a graph with cost matrix

$$B = -A + J$$

gives a bipartition (S^*, T^*) with $|S^*| = |T^*| = \frac{n}{2}$ and weight k, then

$$\frac{n^2}{4} - k$$

is the optimal value of the equipartition problem. (The "$-$" in $B = -A + J$ arises, because we do a maximization instead of minimizing, and the J comes from the constraint $\mathrm{tr} JX = 0$, that is lifted into the objective function. The Lagrange multiplier for this constraint is guessed to be one.)

We consider the instances introduced in [29] of size $n = 124$ and $n = 250$ and summarize in Table 7 the best results for these instances known so far (see [31]). With our algorithm we could prove optimality of the known lower bounds of all instances of size $n = 124$, and one of the instances of size $n = 250$. To the best of our knowledge, these exact solutions were obtained for the first time. The improved gap for the instances of size $n = 250$ and densities 0.02, 0.04 and 0.08 were obtained after a time limit of 32 hours cpu-time. We observe, however, that the recent dissertation [3] contains similar results.

		best known			
n	d	bound	$\lvert E_{cut}\rvert$	gap	new gap
124	0.02	12.01	13	0	0
124	0.04	61.22	63	1	0
124	0.08	170.93	178	7	0
124	0.16	440.08	449	8	0
250	0.01	26.06	29	2	0
250	0.02	103.61	114	10	8
250	0.04	327.88	357	29	22
250	0.08	779.55	828	48	35

Table 7: Best known results of the equipartition problem for the Johnson graphs and the new gap obtained by Biq Mac.

9 Summary

In this paper we have presented an algorithm, that uses a Branch-and-Bound framework to solve the Max-Cut and related problems. At each node of the tree we calculate the bound by using a dynamic version of the bundle method that solves the basic semidefinite relaxation for Max-Cut strengthened by triangle inequalities.

We conclude, that

- our approach solves any instance of all the test-bed considered up to $n = 100$ nodes. To the best of our knowledge, no other algorithm can manage these instances in a routine way.
- we solve problems of special structure and sparse problems up to $n = 300$ nodes.
- for the first time optimality could be proved for several problems of the OR-library. All problems that are reported at the Pseudo-Boolean website [12] with dimensions up to $n = 250$ are now solved.
- for the first time optimality of the equipartition problem for some of the Johnson graphs has been proved, for those where we could not close the gap we reduced the best known gap significantly.
- for sparse problems it is not advisable to use our approach. Since linear programming based methods are capable of exploiting sparsity, solutions might be obtained much faster when applying these methods to sparse data.

Using our algorithm to solve this problem has been made publicly available at

`http://biqmac.uni-klu.ac.at/`.

References

[1] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007. `http://opus.kobv.de/tuberlin/volltexte/2007/1611/`.

[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Oper. Res. Lett.*, 33(1):42–54, 2005.

[3] M. Armbruster. *Branch-and-Cut for a Semidefinite Relaxation of the Minimum Bisection Problem.* PhD thesis, University of Technology Chemnitz, 2007.

[4] F. Barahona and L. Ladányi. Branch and cut based on the volume algorithm: Steiner trees in graphs and max-cut. *RAIRO Oper. Res.*, 40(1): 53–73, 2006.

[5] F. Barahona and A. R. Mahjoub. On the cut polytope. *Math. Programming*, 36(2):157–173, 1986.

[6] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0-1 programming. *Math. Programming*, 44(2, (Ser. A)):127–137, 1989.

[7] J. E. Beasley. Or-library: distributing test problems by electronic mail. *J. Oper. Res. Soc.*, 41(11):1069–1072, 1990.

[8] J. E. Beasley. Or-library. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 1990.

[9] J. E. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem. Technical report, The Management School, Imperial College, London SW7 2AZ, England, 1998.

[10] S. J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM J. Optim.*, 10(2):443–461 (electronic), 2000.

[11] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Math. Programming*, 109(1, Ser. A):55–68, 2007.

[12] E. Boros, P. L. Hammer, and G. Tavares. The pseudo-boolean optimization website. `http://rutcor.rutgers.edu/~pbo/`, 2005.

[13] E. Boros, P. L. Hammer, R. Sun, and G. Tavares. A max-flow approach to improved lower bounds for quadratic $0-1$ minimization. Technical report, RUTCOR Research Report RRR 7-2006, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA, 2006.

[14] S. Burer, R. D. Monteiro, and Y. Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM J. Optim.*, 12(2): 503–521 (electronic), 2001/02.

[15] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch-and-cut algorithm. *J. Statist. Phys.*, 80(1-2):487–496, 1995.

[16] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993.

[17] M. M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1997.

[18] M. Elf, C. Gutwenger, M. Jünger, and G. Rinaldi. *Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS*, volume 2241 of *Lecture Notes in Computer Science*, pages 157–222. Springer, 2001.

[19] I. Fischer, G. Gruber, F. Rendl, and R. Sotirov. Computational experience with a bundle approach for semidefinite cutting plane relaxations of Max-Cut and equipartition. *Math. Programming*, 105(2-3, Ser. B):451–469, 2006.

[20] A. Frangioni, A. Lodi, and G. Rinaldi. New approaches for optimizing over the semimetric polytope. *Math. Program.*, 104(2-3, Ser. B):375–388, 2005.

[21] F. Glover, G. Kochenberger, and B. Alidaee. Adaptative memory tabu search for binary quadratic programs. *Management Sci.*, 44(3):336–345, 1998.

[22] M. X. Goemans and D. P. Williamson. .878-approximation algorithms for max cut and max 2sat. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 422–431, Montreal, Quebec, Canada, 1994.

[23] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995. Preliminary version see [22].

[24] P. Hansen. Methods of nonlinear $0-1$ programming. *Ann. Discrete Math.*, 5:53–70, 1979.

[25] C. Helmberg. Fixing variables in semidefinite relaxations. *SIAM J. Matrix Anal. Appl.*, 21(3):952–969 (electronic), 2000.

[26] C. Helmberg. A cutting plane algorithm for large scale semidefinite relaxations. In *The sharpest cut*, MPS/SIAM Ser. Optim., pages 233–256. SIAM, Philadelphia, PA, 2004.

[27] C. Helmberg and F. Rendl. Solving quadratic $(0,1)$-problems by semidefinite programs and cutting planes. *Math. Programming*, 82(3, Ser. A): 291–315, 1998.

[28] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–361, 1996.

[29] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Oper. Res.*, 37(6):865–892, 1989.

[30] M. Jünger, G. Reinelt, and G. Rinaldi. Lifting and separation procedures for the cut polytope. Technical report, Universität zu Köln, 2006. In preparation.

[31] S. E. Karisch and F. Rendl. Semidefinite programming and graph equipartition. In *Topics in semidefinite and interior-point methods (Toronto, ON, 1996)*, volume 18 of *Fields Inst. Commun.*, pages 77–95. Amer. Math. Soc., Providence, RI, 1998.

[32] S. Kim and M. Kojima. Second order cone programming relaxation of nonconvex quadratic optimization problems. *Optim. Methods Softw.*, 15 (3-4):201–224, 2001.

[33] M. Laurent. The max-cut problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 241–259. John Wiley & Sons, Chichester, 1997.

[34] F. Liers. *Contributions to Determining Exact Ground-States of Ising Spin-Glasses and to their Physics*. PhD thesis, Universität zu Köln, 2004.

[35] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In A. Hartmann and H. Rieger, editors, *New Optimization Algorithms in Physics*, pages 47–68. Wiley, 2004.

[36] M. Muramatsu and T. Suzuki. A new second-order cone programming relaxation for MAX-CUT problems. *J. Oper. Res. Soc. Japan*, 46(2):164–177, 2003.

[37] P. M. Pardalos and G. P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing*, 45(2): 131–144, 1990.

[38] P. M. Pardalos and G. P. Rodgers. Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture. *Ann. Oper. Res.*, 22(1-4):271–292, 1990.

[39] S. Poljak and F. Rendl. Solving the max-cut problem using eigenvalues. *Discrete Appl. Math.*, 62(1-3):249–278, 1995.

[40] S. Poljak and F. Rendl. Nonpolyhedral relaxations of graph-bisection problems. *SIAM J. Optim.*, 5(3):467–487, 1995.

[41] F. Rendl. Semidefinite programming and combinatorial optimization. *Appl. Numer. Math.*, 29(3):255–281, 1999.

[42] F. Rendl, G. Rinaldi, and A. Wiegele. A branch and bound algorithm for Max-Cut based on combining semidefinite and polyhedral relaxations. In *Integer programming and combinatorial optimization*, volume 4513 of *Lecture Notes in Comput. Sci.*, pages 295–309. Springer, Berlin, 2007.

[43] G. Rinaldi. Rudy. `http://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz`, 1998.

[44] A. Wiegele. *Nonlinear optimization techniques applied to combinatorial optimization problems.* PhD thesis, Alpen-Adria-Universität Klagenfurt, 2006.