Solving k-way Graph Partitioning Problems to Optimality: The Impact of Semidefinite Relaxations and the Bundle Method

Miguel F. Anjos, Bissan Ghaddar, Lena Hupp, Frauke Liers, and Angelika Wiegele

Abstract This paper is concerned with computing global optimal solutions for maximum k-cut problems. We improve on the SBC algorithm of Ghaddar, Anjos and Liers in order to compute such solutions in less time. We extend the design principles of the successful BiqMac solver for maximum 2-cut to the general maximum k-cut problem. As part of this extension, we investigate different ways of choosing variables for branching. We also study the impact of the separation of clique inequalities within this new framework and observe that it frequently reduces the number of subproblems considerably. Our computational results suggest that the proposed approach achieves a drastic speedup in comparison to SBC, especially when $k = 3$. We also made a comparison with the orbitopal fixing approach of Kaibel, Peinhardt and Pfetsch. The results suggest that while their performance is better for sparse instances and larger values of k, our proposed approach is superior for smaller k and for dense instances of medium size. Furthermore, we used CPLEX for solv-

Miguel F. Anjos
Canada Research Chair in Discrete Nonlinear Optimization in Engineering, GERAD & École Polytechnique de Montréal, Montréal, QC, Canada H3C 3A7, e-mail: anjos@stanfordalumni.org

Bissan Ghaddar
Centre for Operational Research and Analysis, Defence Research and Development Canada, Department of National Defence, 101 Colonel By Drive, Ottawa, Ontario, Canada, K1A 0K2, e-mail: bghaddar@uwaterloo.ca

Lena Hupp
Department Mathematik, Friedrich-Alexander Universität Erlangen-Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany, e-mail: lena.hupp@math.uni-erlangen.de

Frauke Liers
Department Mathematik, Friedrich-Alexander Universität Erlangen-Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany, e-mail: frauke.liers@math.uni-erlangen.de

Angelika Wiegele
Alpen-Adria-Universität Klagenfurt, Institut für Mathematik, Universitätsstr. 65-67, 9020 Klagenfurt, Austria, e-mail: angelika.wiegele@aau.at

ing the ILP formulation underlying the orbitopal fixing algorithm and conclude that especially on dense instances the new algorithm outperforms CPLEX by far.

1 Introduction

The maximum k-cut (max-k-cut) problem is a graph partitioning problem concerned with finding an optimal k-way partitioning of the set of nodes of an undirected simple graph with weights on the edges. An edge is cut if its endpoints are in different sets of the partition, and a partition is also called a cut of the graph. Thus the weight of a cut is equal to the sum of the weights on the edges cut by its corresponding partition. There are a number of different versions of graph partitioning problems in the literature, depending on the number of sets allowed in a partition, on the types of edge weights allowed, and on the possible presence of additional side constraints such as restrictions on the number of nodes allowed in each partition. Most versions are known to be NP-hard. Graph partitioning problems have myriad applications in areas as varied as telecommunications network planning [20], VLSI circuit design [9], sports scheduling [45, 21], and statistical physics [39].

The special case of max-k-cut with k=2 is known as the max-cut problem. The max-cut problem has been extensively studied; in particular it is known to be equivalent to quadratic unconstrained binary optimization. Among the numerous references for max-cut, we point out Barahona and Mahjoub [10], Deza and Laurent [18], and Boros and Hammer [12]. A prominent application of max-cut is in determining energy-minimum states, i.e., ground states, of Ising spin glasses. The first exact branch-and-cut approach for its solution was presented in [9] and developed further in [39]. Extending the number of shores to $k > 2$, maximum k-cuts need to be computed when determining ground states of Potts glasses. In the physics literature, ground states are usually computed heuristically, but more reliable conclusions can be drawn by analyzing exact solutions.

The max-k-cut problem is sometimes also called the minimum k-partition problem by noting that maximizing the k-cut is equivalent to minimizing the sum of the weights of the edges connecting nodes in the same partition. It was studied by Chopra and Rao in [14] who identified several valid and facet-defining inequalities for the k-partition polytope. Further results can be found in Chopra and Rao [15] and Deza, Grötschel, and Laurent [17].

Armbruster et al. [8] consider the minimum bisection problem, where $k = 2$ and the number of nodes in both partitions has to be less than a given value $F \leq \frac{n}{2}$. The case $F = \lceil \frac{n}{2} \rceil$ corresponds to a minimum equipartition problem since the sizes of both partitions then have to be (as close as possible to) equal. Alternatively, this latter constraint added to the max-cut problem gives the equicut problem which can be motivated by an application to Coulomb glasses in theoretical physics. Motivated by this application, Anjos et al. [5] recently proposed an enhanced branch-and-cut algorithm for equicut based on an approach proposed by Brunetta et al. [13].

More generally, the *k*-way equipartition problem is a minimum *k*-partition problem with the additional constraint that the *k* partitions have to be of the same size. Mitchell [44] applied a branch-and-cut algorithm based on linear programming (LP) to the *k*-way equipartition problem with application to a sports league realignment problem. Lisser and Rendl [41] considered an application of *k*-way equipartition in telecommunications and investigated both semidefinite and linear relaxations of the problem with iterative cutting plane algorithms.

Strong approximation guarantees have been obtained for several of these NP-hard problems. A famous example is the randomized approximation algorithm for max-cut proposed by Goemans and Williamson [25] that uses a semidefinite programming (SDP) relaxation. Frieze and Jerrum [23] extended the approach of Goemans and Williamson to max-*k*-cut and obtained a polynomial-time approximation algorithm and a corresponding rounding technique. In particular, they proved the existence of constants α_k, $k \geq 2$, such that

$$\mathbf{E}(w(\mathscr{V}_k)) \geq \alpha_k w(\mathscr{V}_k^*)$$

where $w(\mathscr{V}_k) = \sum_{1 \leq r < s \leq k} \sum_{i \in V_r, j \in V_s} w_{ij}$, $\mathscr{V}_k^*$ determines an optimal cut, and $\mathbf{E}$ denotes the expected value. For small values of *k*, the best-known lower bounds for these constants are given by de Klerk et al. [16]. The improved SDP relaxation for max-cut of Anjos and Wolkowicz [7] provides very tight bounds for max-cut and perfectly captures the faces of dimension 1 of the cut polytope [6]. This ability to capture portions of the structure of the underlying polytope was proved for the whole Lasserre hierarchy by Laurent [34]. Eisenblätter [20] used an SDP relaxation for the minimum *k*-partition problem and proved that all the feasible solutions for the SDP problem cannot violate the (facet-defining) triangle and clique inequalities for the *k*-partition polytope by more than a small amount, thus showing that an SDP relaxation can closely approximate the structure of the *k*-partition polytope.

Our interest is in computing global optimal solutions for max-*k*-cut problems. Computationally speaking, SDP relaxations often yield stronger bounds than LP relaxations. However, this strength usually comes at the expense of long running times. Thus, it is not clear beforehand whether linear or semidefinite relaxations lead to best performance.

For max-cut, sparse instances can usually be solved efficiently with LP-based methods for large graphs. The web-based Spin Glass Server [4] is especially designed for fast solutions of instances defined on grids that arise in statistical physics [39]. For instance, for a two-dimensional lattice with $L \leq 80$ and periodic boundary conditions, one ground-state computation takes less than two minutes on average on a SUN Opteron (2.2 GHz) machine; for 120^2 lattices the computation takes 28 minutes [40]. On the other hand, SDP-based methods perform better for dense instances of max-cut [46]. The SDP-based web server BiqMac [1] can solve max-cut instances with arbitrary structure with up to 100 vertices [48].

For the *k*-way equipartition problem, the LP-based branch-and-cut algorithm of Mitchell [44] found the optimal solution for the NFL realignment problem where $k = 8$ and $n = 32$, whereas a percentage gap of less than 2.5% was given for graphs of sizes 100 to 500. Lisser and Rendl [41] found that for graph sizes ranging from 100 to 900 vertices and for $k = 5$ and $k = 10$, the SDP approach produced a gap between 4%-6% from the optimal solution and had overall better performance than the LP approach.

For sparse instances of minimum bisection, the computational results of Armbruster et al. [8] suggest that SDP relaxations are superior to the corresponding LP relaxations. On the other hand, Anjos et al. [5] compared basic LP and SDP relaxations for the equicut problem, and found that linear bounds can be competitive with the semidefinite ones and can be computed much faster. While their results appear to contradict the above observations, it is important to note that they focus on dense instances coming from the physics application, and that their specialized relaxation includes constraints that are not valid for the minimum bisection polytope in general.

In this paper, we focus on max-k-cut for $k \geq 3$. Our motivation is that while effective computational procedures that yield globally optimal solution for arbitrary instances with up to 100 vertices and sparse graphs of considerably larger sizes have been implemented for the k=2 case, to the best of our knowledge, most of the procedures proposed in the literature either cannot be applied for general k, provide no guarantee of global optimality, or enforce additional constraints.

Among the exceptions, Kaibel, Peinhardt and Pfetsch [31, 32] applied a symmetry-breaking method called orbitopal fixing (OF) to graph partitioning problems within an LP-based branch-and-cut. Symmetry arises in graph partition problems because different feasible solutions may represent the same partition. The feasible set of the problem can thus be partitioned into orbits so that all the solutions in an orbit represent the same partition. This structure is exploited by OF through choosing one representative solution from each orbit, namely the lexicographically maximal one, and the branching and pruning steps are adjusted to restrict the enumeration to only such solutions. This is a specialization to partition problems of the isomorphism pruning technique of Margot [42, 43]. The authors present results for the minimum k-partition problem in sparse graphs with up to 50 nodes and a few hundred edges [31].

Another exception is the SDP-based branch-and-cut algorithm for the minimum k-partition problem proposed by Ghaddar, Anjos and Liers [24]. Their SBC algorithm combines the SDP relaxation proposed by Eisenblätter [20] with valid inequalities for the k-partition polytope and with a novel iterative clustering heuristic (ICH) that finds feasible solutions using the SDP optimal solution. The computational results reported in [24] show that ICH consistently provides feasible solutions that are better than those obtained using the hyperplane rounding techniques of Goemans and Williamson (for $k = 2$) and of Frieze and Jerrum (for $k \geq 3$). Ghaddar et al. presented results showing that SBC computes globally optimal solutions for dense graphs with up to 60 nodes, for (sparse) grid graphs with up to 100 nodes, and for different values of $k \geq 3$.

In this paper, we combine the approach of Ghaddar et al. with the design principles of BiqMac [48] to compute globally optimal solutions for max-k-cut more efficiently. We refer to this new approach as bundleBC. Although straightforward in principle, this combination raises several challenges, including the following:

- branching decisions may lead to subproblems that are infeasible or that have no interior (this can be avoided for max-cut by appropriate switching and shrinking steps);
- a wider variety of cutting planes needs to be generated and managed dynamically; and
- the constraints of the SDP relaxation itself need to be handled differently, i.e., we relax the lower bound constraints of the initial SDP relaxation and treat them as cuts.

Our computational results suggest that bundleBC achieves a drastic speedup in comparison to SBC, especially when $k = 3$. Furthermore, a comparison with the results reported by Kaibel et al. [31], suggests that for $k = 3$ and medium-sized dense instances (30 nodes), our approach performs better than their OF approach, whereas their performance is better for sparse instances and for larger values of k. Additionally, we used CPLEX to evaluate the ILP model underlying the OF approach.

This paper is organized as follows. In Section 2 we state the formal definition of the max-k-cut problem and briefly summarize the relevant formulations and relaxations in the literature. The proposed exact algorithm is described in Section 3. Section 3.1 is concerned with the upper bound computation using a bundle to solve the SDP relaxations, and Section 3.2 is described the heuristic we use for computing lower bounds. Section 3.3 describes the 6 branching rules that we tested, and how we handle the possibility that branching sometimes yields SDP subproblems that are infeasible or that have no interior. Section 4 presents the basic implementation details of bundleBC. Computational results are reported in Section 5. Section 5.1 describes the benchmark sets of instances that we used, Section 5.2 reports the performance of the 6 branching rules that we considered, and Section 5.3 studies the impact of clique inequalities on the performance of bundleBC. Section 5.4 presents comparisons of bundleBC with SBC, and Section 5.5 compares the performance of bundleBC with the orbitopal fixing approach and presents the CPLEX results. Section 6 concludes the paper.

2 Problem Description, Formulations and Relaxations

An instance of the max-k-cut problem is specified by fixing an undirected graph $G = (V,E)$ with edge weights w_{ij} of the edges, and a positive integer $k \geq 2$. The objective is to find a partition of V into at most k disjoint partitions $V_1,\ldots,V_k$ such that the sum of the weights of edges joining different partitions is maximized. We assume without loss of generality that G is a complete graph (missing edges can be added with a corresponding weight of zero).

From a semidefinite perspective, the max-k-cut problem can be formulated as:

$$\max \sum_{i,j\in V,\, i<j} w_{ij} \frac{(k-1)(1-X_{ij})}{k} \tag{1}$$
$$\text{s.t. } X_{ii} = 1 \qquad \forall i \in V \tag{2}$$
$$X_{ij} \in \{\frac{-1}{k-1}\,, 1\} \qquad \forall i,j \in V,\, i<j \tag{3}$$
$$X \succeq 0,$$

where $X_{ij} = \frac{-1}{k-1}$ if vertices i and j are in different partitions, and $X_{ij} = 1$ if they are in the same partition. Replacing the binary constraint (3) by $\frac{-1}{k-1} \leq X_{ij} \leq 1$ results in a semidefinite relaxation. However, the constraint $X_{ij} \leq 1$ can be dropped since it is enforced implicitly by the constraints $X_{ii} = 1$ and $X \succeq 0$. We end up with the following SDP relaxation:

$$(\text{SM}k\text{C}) \quad \max \sum_{i,j\in V,\, i<j} w_{ij} \frac{(k-1)(1-X_{ij})}{k} \tag{4}$$
$$\text{s.t. } X_{ii} = 1 \qquad \forall i \in V \tag{5}$$
$$X_{ij} \geq \frac{-1}{k-1} \qquad \forall i,j \in V,\, i<j \tag{6}$$
$$X \succeq 0$$

or, alternatively, using the Laplace matrix L of the graph and rewriting the constraints $X_{ii} = 1$, we end up with

$$(\text{SM}k\text{C}) \quad \max \frac{k-1}{2k} \langle L, X \rangle \tag{7}$$
$$\text{s.t. } \operatorname{diag}(X) = e \tag{8}$$
$$X_{ij} \geq \frac{-1}{k-1} \qquad \forall i,j \in V,\, i<j \tag{9}$$
$$X \succeq 0$$

with e being the vector of all ones of length $|V|$. Note that if we fix $k = 2$ in (SMkC), we obtain the SDP relaxation used for max-cut by Goemans and Williamson [25].

The relaxation (SMkC) was first used by Frieze and Jerrum [23]; it is the basis of the SBC algorithm of Ghaddar et al. [24]. In that algorithm, the SDP relaxation was further tightened by adding valid inequalities. The two types of valid inequalities used in SBC are the triangle and the clique inequalities. The triangle inequalities are based on the observation that if any two nodes i and j are in the same partition, and j and another node k are in the same partition, then also nodes i and k necessarily have to be in the same partition. For the SDP formulation, the $3\binom{|V|}{3}$ triangle inequalities have the form:

$$X_{ij} + X_{jh} - X_{ih} \leq 1, \tag{10}$$

where i, j, and $h \in V$. The $\binom{|V|}{k+1}$ clique inequalities ensure that for every subset of $k+1$ nodes, at least two of the nodes must belong to the same partition:

$$\sum_{i,j \in Q, i<j} X_{ij} \geq -\frac{k}{2} \quad \forall Q \subseteq V \text{ where } |Q| = k+1.$$

Together with the constraints (10), this implies that there are at most k partitions.

3 Proposed Exact Algorithm

We use a branch-and-bound framework to solve the max-*k*-cut problem to global optimality. To set up the framework, the following three issues must be addressed:

- how to obtain upper bounds;
- how to obtain lower bounds, i.e., high-quality cuts; and
- how to branch.

The computation of the upper bounds is the subject of Section 3.1. Computing lower bounds is discussed in Section 3.2, and the question how to branch is addressed in Section 3.3. Algorithm 1 gives the steps as they are executed at each node of the branch-and-bound tree.

3.1 Computing Upper Bounds

It is well known that the bounds obtained by the LP relaxation of the ILP formulation are often weaker than the bounds obtained using the SDP relaxation (see e.g. [20, 24]). However, their computation is usually time consuming. In this work, we focus on an approach that maintains the strength of the relaxations using a fast approximation procedure to speed up computing times.

To this end, we make use of the SDP relaxation (SM*k*C) tightened by facets of the partition polytope. Specifically we use triangle and clique inequalities. Solving the resulting relaxation is not trivial because the number of inequalities (for large graphs) is too large and the SDP problem becomes intractable for interior point methods. Thus we need an alternative machinery to obtain this bound, namely a dynamic version of the bundle method.

3.1.1 Bundle Methods

The bundle method was first proposed by Lemarechal [37], and later on further investigated and refined by several authors, e.g., [33, 50, 38]. It has been developed for finding the approximate minimizer of a non-smooth convex function $f(\gamma)$ over

$\gamma \in R^n$. In order to apply the bundle method it is necessary to be able to obtain for any given γ the function value $f(\gamma)$ and a subgradient $g \in \partial f(\gamma)$. We assume that an oracle is available to return these values (see Section 3.1.3 for the specifics of the oracle that we use). This information is collected for different γs in a so-called “bundle” and used to construct a minorizing cutting plane model $\hat{f}$ of f.

To find a new value of γ, the displacement from the current point $\hat{\gamma}$ is penalized by adding a term proportional to $\|\gamma - \hat{\gamma}\|$ to the cutting plane model $\hat{f}$. Thus, the bundle algorithm requires minimizing

$$\hat{f}(\gamma) + \frac{1}{2\sigma}\|\gamma - \hat{\gamma}\| \tag{11}$$

with σ being some suitably chosen weight. Solving this problem is done by solving a sequence of convex quadratic problems of “small” dimension, i.e. dimension equal to the size of the bundle. The minimizer gives a new trial point $\tilde{\gamma}$ for which the oracle supplies the function value and a subgradient. This new information is added to the bundle and used to improve the cutting plane model. Then the whole process is repeated until the subgradient at the current point is sufficiently close to zero.

The bundle method as a tool for solving SDP problems has already been used by Poljak and Rendl [47] for solving the basic SDP relaxation for max-cut. Later on, the spectral bundle method has been introduced [29, 27]. In [26] a variant of the spectral bundle method is developed that allows adding and deleting of cutting planes on the fly, and convergence of this method is proved.

Here we follow the concept of Fischer et al. [22]. Their idea is to apply the bundle method to the partial Lagrangian dual function, partial in the sense that only some of the constraints are dualized and lifted into the objective function by using Lagrangian multipliers, whereas constraints that are considered to be “easy” are handled directly inside the oracle. In other words, an oracle call amounts in solving a semidefinite program having only “easy” constraints.

In contrast to interior point methods, bundle methods are capable of solving semidefinite programs with a few thousand constraints. The price one pays for this is in the accuracy of the solution. However, the results in [22] demonstrate that the bundle algorithm often returns a reasonably accurate approximation.

3.1.2 Conic Bundle

Our aim is to solve the semidefinite program

$$\begin{aligned}
(\mathrm{SM}k\mathrm{C}_{\mathrm{strengthened}}) \qquad \max\ & \frac{(k-1)}{2k}\langle L, X\rangle && (12)\\
\text{s.t. } & \operatorname{diag}(X) = e && (13)\\
& \mathscr{A}(X) \le b && (14)\\
& X \succeq 0
\end{aligned}$$

where we collect in $\mathscr{A}(X) \le b$ all the bound constraints, i.e., $X_{ij} \ge \frac{-1}{k-1}$, $\forall i, j \in V$, $i < j$, the set of triangle-inequalities and the set of clique-inequalities. Dualizing all the inequality constraints, we obtain the partial Lagrangian:

$$\begin{aligned}\mathscr{L}(X;\gamma) &= \frac{(k-1)}{2k}\langle L, X\rangle + \gamma^\top(b - \mathscr{A}(X)) \\ &= b^\top\gamma + \left\langle \frac{(k-1)}{2k}L - \mathscr{A}^\top(\gamma), X \right\rangle,\end{aligned}$$

and the dual functional reads

$$\begin{aligned}f(\gamma) &= \max_{X \succeq 0, \operatorname{diag}(X) = e} \mathscr{L}(X;\gamma) \\ &= b^\top\gamma + \max_{X \succeq 0, \operatorname{diag}(X) = e} \left\langle \frac{(k-1)}{2k}L - \mathscr{A}^\top(\gamma), X \right\rangle\end{aligned} \tag{15}$$

The number of inequality constraints is too large to handle even after they are dualized. Our approach is to include all the bound constraints, and to add only those inequalities that are active at the optimum. Since this information is not known at the beginning, the choice of triangle- and clique-inequalities is updated in the course of the algorithm, as described below in Section 3.1.4.

The function (15) is then minimized over $R^n_{\ge 0}$ using the bundle method. We use the Conic Bundle software of Helmberg [2]. This implementation of the bundle method supports the minimization of the function arising from a Lagrangian dual, as in our case, i.e., it allows to generate primal solutions. Furthermore it offers the possibility of adding and removing constraints in the course of the algorithm, as needed for our purposes.

3.1.3 Oracle

As already mentioned, in each bundle iteration an oracle is called to compute the function value and a subgradient at the current γ. The function evaluation amounts to solving the SDP problem

$$\begin{aligned}(\text{SMC}_{\text{basic}}) \quad \max\ & \left\langle \frac{(k-1)}{2k}L - \mathscr{A}^T(\gamma), X \right\rangle \\ \text{s.t. } & X_{ii} = 1 \quad \forall i \in V \\ & X \succeq 0.\end{aligned}$$

The optimal solution $\tilde{X}$ of this SDP is then used to compute the function value

$$f(\gamma) = b^\top\gamma + \left\langle \frac{(k-1)}{2k}L - \mathscr{A}^\top(\gamma), \tilde{X} \right\rangle$$

and a subgradient

$$g(\gamma) = b - \mathscr{A}(\tilde{X}).$$

Note that the cost matrix depends on γ and therefore changes at each iteration. The feasible set of ($\text{SMC}_{\text{basic}}$) is the so-called elliptope, which has been well studied, see e.g. [36, 35]. The problem ($\text{SMC}_{\text{basic}}$) can thus be solved efficiently by interior point methods, even for large dimension. We implemented the primal-dual interior point method proposed in [30].

After branching, we have to add equality constraints of the form $X_{ij} = \frac{-1}{k-1}$ to ($\text{SMC}_{\text{basic}}$), as explained in Section 3.3 below. Since the number of these constraints is small, we can still use an interior point method to solve the SDP problem. However, we may end up with a problem having no interior, for example if we have a k-clique for which all the edge variables $X_{ij} = \frac{-1}{k-1}$. If this happens, we solve the SDP problem using CSDP [11] since its infeasible interior point algorithm runs well in this situation.

3.1.4 Adding Valid Inequalities

Once the SDP relaxation (SMkC) is solved, one can look for violated inequalities and add them to the relaxation, hence improving the upper bound. Triangle and clique inequalities are added at each iteration of the bundle algorithm and non-binding inequalities are detected and removed. Looking for violated triangle inequalities by complete enumeration is not computationally expensive. We describe in Section 4.1 how we manage the search and addition of violated triangle inequalities.

On the other hand, exact separation of clique inequalities is an $\mathscr{N}\mathscr{P}$-hard problem, and complete enumeration becomes intractable already for small values of k. Therefore, we use a separation heuristic that generates inequalities that are promising. It does not necessarily determine a violated inequality whenever one exists, however the algorithm is fast and yields good bounds.

The clique inequalities that are binding at optimality usually cover the whole graph, and each vertex in the graph is contained in several different clique inequalities. The separation heuristic is designed to have a similar behaviour. For each vertex v in the graph, the algorithm grows a clique of size $k+1$ containing v. Vertices are added to the cliques in a greedy fashion. In each iteration, a vertex is added to a clique of size smaller than $k+1$ that contributes the smallest amount to the left-hand side of the corresponding clique inequality. The heuristic is described in detail in [24]. For a graph with n vertices, this procedure generates n clique inequalities. Violated ones are added to the problem formulation.

3.2 Lower Bound Heuristic

Using the conic bundle we can generate approximate primal solutions in the course of the minimization algorithm. We use a heuristic method to compute a feasible k-cut from these approximate primal matrices X^*. This way we produce lower bounds which are useful for fathoming in the branch-and-cut tree.

There are two heuristics for extracting k-cuts from a primal solution X^*. The first one is the heuristic proposed by Frieze and Jerrum [23] and called FJ in the following. It works as follows:

1. Compute unit vectors $v_1, \dots, v_n \in \mathbb{R}^n$ satisfying $v_i^T v_j = X^*_{ij}$ where $i, j \in V$.
2. Randomly generate k vectors $r_1, \dots, r_k \in \mathbb{R}^n$ with their kn components drawn from independent and identically distributed random variables with a standard normal distribution with mean 0 and variance 1.
3. Partition V into $\mathscr{V}_k = \{V_1, \dots, V_k\}$ according to

$$V_j = \{i : v_i \cdot r_j \geq v_i \cdot r_{j'}, \text{ for } j \neq j'\} \text{ for } 1 \leq j \leq k.$$

The second heuristic is called ICH. It was proposed by Ghaddar et al. [24] and used in their SBC algorithm. ICH works by aggregating information from X^* corresponding to subgraphs of G. Specifically, ICH sums the X^*_{ij} values on the edges between each of the $\binom{n}{k}$ subsets of k vertices, then sorts the resulting list of values, and places a subset of k vertices all in the same partition (or all in different partitions) when the sum is one of the largest values (or one of the smallest). The intuition behind this approach is that aggregated information is more reliable than single elements of data.

The implementation of the lower bound computation is described in Section 4.2 below.

3.3 Branching

The final ingredient of a branch-and-bound algorithm is how to subdivide the set of feasible solutions. It is well known that part of the success of a branch-and-bound algorithm depends on the choice of the branching variable X_{ij}.

3.3.1 Branching Rules

We use the information in the solution X^* of the SDP relaxation of the current node to choose a branching variable X^*_{ij}. We adapt the rules R1-R4 of [28] for max-cut in order to derive different choices for branching variables.

There are two important differences with respect to the max-cut case in [28] that we must address. First, while for max-cut the entries in X^* are all in the interval $[-1,1]$, the SDP relaxation (SMkC) restricts the entries in X^* to the interval $[-\frac{1}{k-1},1]$. Second, since we dualize the bound constraints $X^*_{ij} \geq -\frac{1}{k-1}$, some values of X^*_{ij} may lie outside this interval. We considered different ways to deal with these differences.

Rules R1 and R3 are adapted most easily. Rule R1 chooses the "most decided" variable, i.e., we simply branch on the edge ij that is closest to $-\frac{1}{k-1}$ or to 1. By choosing an edge that seems to be already decided, the hope is that for the opposite decision the node will be fathomed quickly. This results in a deep but narrow branch-and-bound tree.

Rule R3 branches on the variable that is "least decided", i.e., we branch on the edge ij for which X^*_{ij} is closest to the middle of the interval $[-\frac{1}{k-1},1]$. If all the variables are either nearly 1 or less than or equal to $-\frac{1}{k-1}$, we choose ij corresponding to the minimum value of X^*_{ij}. By branching on the most undecided edge, we hope that the upper bounds will improve quickly.

Rules R1 and R3 do not distinguish between the variables with values outside the interval $[-\frac{1}{k-1},1]$ and the others.

Rules R2 and R4 are more elaborate. Instead of working with individual entries, these rules are based on the closeness of the rows of the matrix X^* to $\{-\frac{1}{k-1},1\}$ vectors.

Rule R2 looks for the two rows i' and j' that are closest to a $\{-\frac{1}{k-1},1\}$ vector. Let m denote the middle of the interval $[-\frac{1}{k-1},1]$. The branching edge i', j' is chosen as

$$i' = \text{argmin}_{1\leq i\leq n} \sum_{r\neq i, r=1}^{n} ((1-m) - |X^*_{ir} - m|)^2$$

$$j' = \text{argmin}_{1\leq j\leq n, j\neq i'} \sum_{r\neq j, r=1}^{n} ((1-m) - |X^*_{jr} - m|)^2.$$

Rule R4 looks for rows i' and j' such that i' is closest to a $\{-\frac{1}{k-1},1\}$ vector wheras j' is farthest from being feasible. Here i' and j' are chosen such that

$$i' = \text{argmin}_{1\leq i\leq n} \sum_{r\neq i, r=1}^{n} ((1-m) - |X^*_{ir} - m|)^2$$

$$j' = \text{argmin}_{1\leq j\leq n} \sum_{r\neq j, r=1}^{n} (X^*_{jr} - m)^2.$$

Concerning the variables that are outside the interval $[-\frac{1}{k-1},1]$, we investigated two options for each of R2 and R4. The first option is to treat them just like the others (this corresponds to our rules R2 and R4). The second option for R2 is our rule R2a according to which we do not consider these variables as branching candidates unless all the variables inside the feasible interval are equal to 1 or $-\frac{1}{k-1}$. When this is the case, rule R2a selects the variable outside the interval with the smallest value.

Similarly, rule R4a works just as R4 but first considers only variables with values in the interval $[-\frac{1}{k-1}, 1]$ as candidates. If all those variables are already equal to 1 or $-\frac{1}{k-1}$, R4a selects the variable X^*_{ij} with the smallest value.

3.3.2 Shrinking and SDP Relaxations Without Interior

In the case where we fix $X_{i'j'} = 1$ at a particular node of the branch and bound tree, the resulting problem is equivalent to maximum k-cut of dimension $n-1$. Hence we can shrink the graph, i.e., we reduce the graph size by eliminating the vertex j'. The Laplacian matrix $\tilde{L}$ for the shrunken graph has entries $\tilde{l}_{ij}$, $i, j \in \{1,\dots,n\}\backslash\{j'\}$, as follows:

$$\tilde{l}_{ij} = \begin{cases} l_{ij} & \text{if } i, j \neq i' \\ l_{ii'} + l_{ij'} & \text{if } i \neq i',\ j = i' \\ l_{i'j} + l_{j'j} & \text{if } i = i',\ j \neq i' \\ l_{i'i'} + 2l_{i'j'} + l_{j'j'} & \text{if } i, j = i' \end{cases}$$

When we fix $X_{ij} = \frac{-1}{k-1}$, we cannot shrink the graph immediately, but we could shrink the graph as soon as there is a k-clique with all the values on its edges fixed to $\frac{-1}{k-1}$. However, performing this shrinking would require either expensive clique searches or more than two branches at each node of the branch-and-bound tree. Neither possibility is attractive, and moreover good cuts found on the shrunken graph cannot be extended to the original graph in a straightforward way. Therefore we omit these shrinkings, but as a consequence the SDP relaxation to be solved by the oracle may have no interior. When this happens, we solve the relaxations using CSDP [11] as mentioned earlier in Section 3.1.3.

Algorithm 1 One node of the branch-and-bound algorithm

```
1. Initialize $\gamma$ and solve (SMC$_{\text{basic}}$) using the oracle. Obtain a primal matrix $X^*$ and an upper
   bound $ub$.
2. Apply a heuristic to the current $X^*$ to obtain a $k$-cut and a lower bound $lb$.
3. Separate triangle inequalities.
4. While progress is made
     a. Do a descent step, i.e., obtain improved $ub$.
     b. If number of descent steps $mod$ $10 = 0$, apply a heuristic to the current $X^*$ to obtain a
        $k$-cut and a lower bound $lb$..
     c. If $lb \geq ub$ then stop: return and fathom node.
     d. Remove triangles and cliques if non-binding.
     e. Separate triangle and clique inequalities.
5. Apply a heuristic to the current $X^*$ to obtain a $k$-cut and a lower bound $lb$.
6. If $lb \geq ub$ then stop: return and fathom node.
7. Choose an edge for branching and return.
```

4 Implementation Details

In this section we explain how we set various parameters for the overall algorithm.

We used the Conic Bundle with its default settings. In particular, the relative precision requirement for successful termination was set to the default value of 10^{-5}.

The following subsections describe the preliminary experiments we performed to decide on a strategy for adding triangle inequalities and possibly clique inequalities, and a heuristic for computing lower bounds. In principle, there are several different parameter values and their combinations to test. We focused on the instances on complete graphs with Gaussian or bimodal distribution, and always averaged over five instances of the same size. In the course of our experiments we found that the resulting settings also worked well for the other types of graphs.

4.1 Adding Triangle Inequalities

In the course of the bundle iterations we have to find a good set of triangle inequalities to add. Since enumeration of all triangle inequalities is cheap, we do this after every descent step of the bundle algorithm. The tolerance for considering an inequality as violated is 10^{-3}, and we build a heap of (at most) 5000 most-violated triangle inequalities.

Then we want to add m violated inequalities. We experimented with doing this in three different ways:

- selecting m inequalities randomly among the 5000;
- selecting the $\frac{m}{2}$ most violated ones and $\frac{m}{2}$ randomly from the remaining;
- selecting the m most violated.

It turned out that none of these options clearly stood out from the others, though the second option seemed to be slightly better. Thus we chose the second strategy for our algorithm.

As for the choice of m, we ran experiments with $m = 500$ and $m = 1000$. Again there was no clear winner but $m = 500$ was slightly better so we chose this value.

4.2 Computing Lower Bounds

As mentioned in Section 3.2 we have two candidates for computing lower bounds, namely the heuristics ICH and FJ. While the computational results for minimum k-partition in [24] suggest that ICH consistently provides better k-cuts than FJ, its running times are much longer. For this reason, and because we want to solve large instances of max-k-cut, we choose to use FJ.

We experimented with how often to run the heuristic at each node of the branch-and-bound tree. While calling the heuristic often is time-consuming, not having a good lower bound at hand can cause the tree to be much larger. We tried three different settings for the frequency of the heuristic calls at each node:

- after every descent step;
- after every 10^{th} descent step;
- only at the beginning and at the end.

The computational results did not give strong evidence that one of above mentioned options is better than the others. The second setting improved slightly over the others, therefore this was our choice for our algorithm.

5 Computational Results

Our 32-Bit executables were run on 2.3 GHz Intel Xeon processors with 32 GB memory. For each instance, we allowed a maximum CPU time of 10 hours. We refer to our new algorithm as bundleBC.

5.1 The Benchmark Sets of Instances

We used the following sets of instances for our computational results:

Set A To have instances with varying number of vertices, we generated graphs with $|V|$ ranging from $10,20,\dots,50$. Edges are chosen randomly such that we yield graphs with edge densities $25\%, 50\%$, and 100%. The weights on the edges are randomly chosen, either following a Gaussian or a bimodal ± 1 distribution. For each combination of $|V|$ and edge density, we generated 5 different instances and we always report averages over the 5 instances with the same values of $|V|$, edge density, and weight distribution.

Set B We also considered the instances from [3] for our numerical experiments. The first two classes of instances consist of complete graphs. Edge weights are either chosen as $|i-j|$ for edge (i,j), or are drawn randomly in $\{0,1,\dots,9\}$.
A different class of instances stems from an application in physics in which energy-minimum states of so-called Potts glasses need to be determined. In this application, instances are regular two- or three-dimensional grids with edge weights that are either Gaussian distributed around zero having variance one, or they are taken from $\{\pm 1\}$, where 50% of the weights are negative. These instances were generated using rudy graph generator [49].
The name of an instance encodes its size followed by the distribution of the weights and the random seed that initializes rudy. For example, the instance 2g_3_93 denotes a two-dimensional grid with Gaussian distributed weights of size 3^2 and random seed 93, whereas instance 3pm_234_234 denotes a three-dimensional grid with ± 1 weights of size $2 \times 3 \times 4$ and random seed 234.

Set C Finally, we take the set of instances from [31]. They generate instances with $|V| = 30$ and different number of edges m, namely $m = 200$ (sparse), $m = 300$ (medium), and $m = 400$ (dense). Furthermore they consider graphs with $|V| = 50$ and $m = 560$. Edges are chosen uniformly at random until the specified number of edges is reached. The weights on these edges are drawn independently and uniformly at random from $\{1,\dots,1000\}$. For each $|V|$ and each edge density three instances are generated.

The values of k were chosen as $k = 3$ and $k = 5$. For instances from [3] we tested additionally $k = 7$, and for the instances from [31] we consider $k \in \{3,6,9,12\}$ to allow a comparison with the results in [31].

5.2 Choosing a Branching Rule

We implemented the six branching rules R1, R2, R2a, R3, R4, and R4a as they were explained in Section 3.3 and ran experiments using the instances described in Section 5.1. The different types of instances all display a similar behavior; thus we restrict our presentation to the results on the instances of benchmark set A.

We use performance profiles as proposed by Dolan and Moré [19] to facilitate the comparison of the branching rules. We set up our profiles as follows. Let P be the set of parameters we want to compare (for example the set of all different branching rules) and let I be the set of instances for which we ran our experiments with the different parameter settings $p \in P$. For each instance i and parameter $p \in P$ the performance ratio for the running time is calculated as

$$PR^{\text{time}}_{i,p} = \frac{\text{RunningTime}_{\,i,p}}{\min\{\,\text{RunningTime}_{\,i,p} : p \in P\}},$$

and the performance ratio for the number of subproblems is obtained as

$$PR^{\text{sub}}_{i,p} = \frac{\#\,\text{subproblems}_{\,i,p}}{\min\{\#\,\text{subproblems}_{\,i,p} : p \in P\}}.$$

If an instance i could not be solved for parameter setting p within the given time limit of $10\,h = 36,000\,s$ then we set $PR^{\text{time}}_{i,p} = PR^{\text{time}}_{\max}$ and $PR^{\text{sub}}_{i,p} = PR^{\text{sub}}_{\max}$ for suitably large values. (The specific choice of these does not affect the resulting profiles.) Our performance profiles are defined by the empirical distribution function

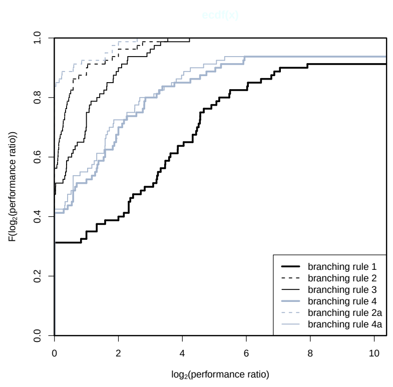
$$F(pr) = P(i \in I : \log_2(PR_{i,p}) \le pr)$$

where we use a $\log_2$ scale for ease of visualization.

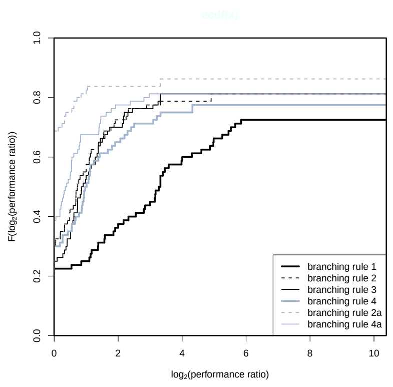
Unlike the observations in Sections 4.1 and 4.2, we found that the choice of branching rule has a strong influence on the computing times. Indeed the CPU times sometimes differ by more than two orders of magnitude between different rules. The main observation is that branching rule R2a is best, and that R2 is usually the second best. This dominance of R2a and R2 is independent of the size of the graph, its density, and the value of *k*. On the other hand, R1 usually leads to the worst performance.

The performance profiles of the CPU time for the different options of choosing a branching variable are shown in Figure 1. The top figures represent the time comparison for complete graphs, the bottom figures for graphs with 25% and 50% edge density. Figures on the left refer to $k = 3$, figures on the right to $k = 5$.

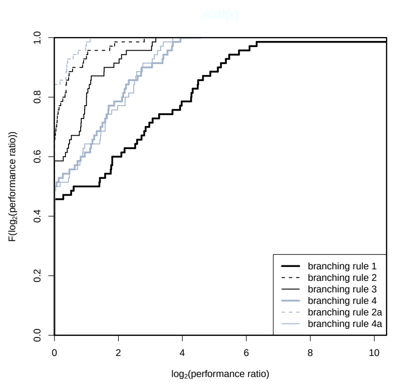
These results demonstrate that the impact of the different branching strategies is greater for $k = 3$ than for $k = 5$. In other words, for $k = 5$ the lines in the profile are closer to each other. Considering complete graphs versus graphs with edge densities 25% and 50%, the performance profiles indicate that the branching strategy has a greater impact on the run time for complete graphs.



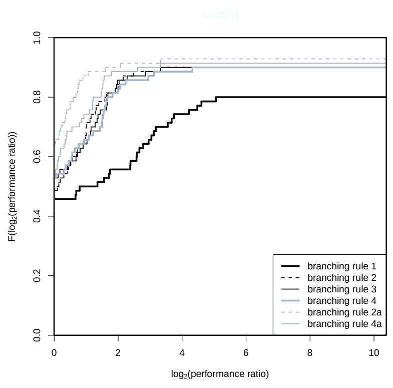
(a) $k = 3$, edge density 100%



(b) $k = 5$, edge density 100%

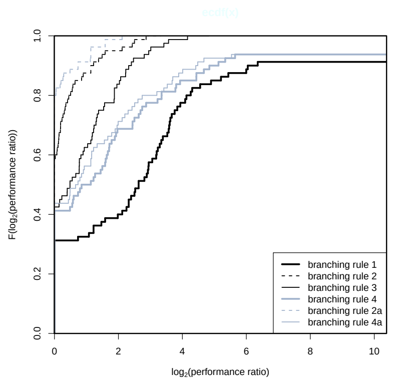


(c) $k = 3$, edge densities 25% and 50%

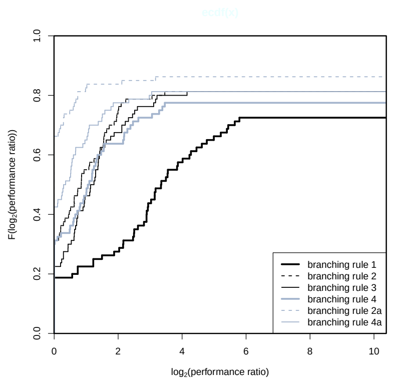


(d) $k = 5$, edge densities 25% and 50%

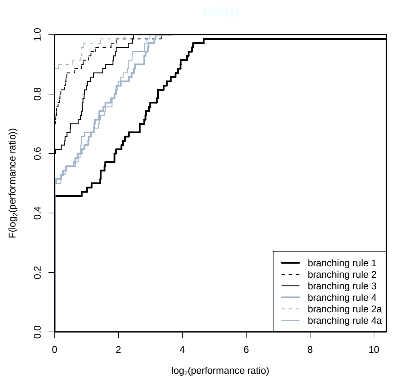
Fig. 1 Performance profiles for the 6 branching rules with respect to the running times for complete graphs (top profiles) and graphs with edge density 25% and 50% (bottom profiles). Edge weights follow a Gaussian or a bimodal distribution.



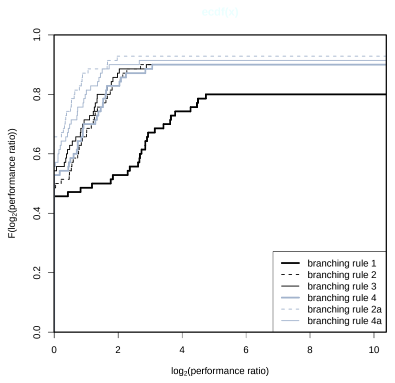
(a) $k = 3$, edge density 100%



(b) $k = 5$, edge density 100%



(c) $k = 3$, edge densities 25%, 50%



(d) $k = 5$, edge densities 25%, 50%

Fig. 2 Performance profiles for the 6 branching rules with respect to the number of nodes in the branch-and-bound tree for complete graphs (top profiles) and graphs with edge density 25% and 50% (bottom profiles). Edge weights follow a Gaussian or a bimodal distribution.

The number of nodes in the branch-and-bound tree is clearly related to the time needed for solving the problem. However, since in each iteration of the bundle algorithm we obtain a valid upper bound, we may be able to stop the bound computation after very few bundle iterations. This usually happens if an edge seems to be already decided whether it is cut or not: the opposite decision should lead to fathoming the node quickly. Therefore a larger number of nodes in the branch-and-bound tree may still lead to shorter overall run times if the bound computation can be stopped early in many of the nodes.

We looked at the influence of the different branching strategies on the number of nodes in the branch-and-bound tree. The performance profiles are given in Figure 2. We do not observe any significant differences in the performance profiles showing the CPU time (Figure 1) and those referring to the number of nodes in the branch-

and-bound tree (Figure 2). Indeed, R2a is again the best performer and R2 is usually second best. Concerning the different characteristics of the problem, once more the impact of the branching rule is greater for $k = 3$ than for $k = 5$, and is less evident for instances having edge density 25% and 50%.

For all subsequent results, we fixed the branching rule to R2a because it usually gives the best results.

5.3 Separating Cliques

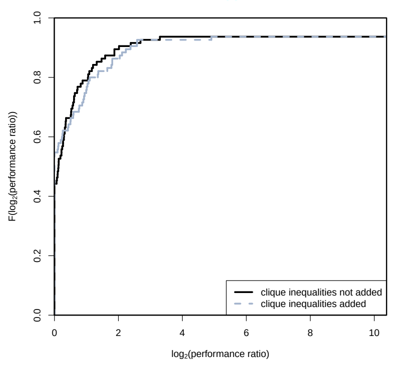
In this section we study the impact of clique separation on the performance of bundleBC. We first compared bundleBC with and without clique separation for $k = 3,5$ for the benchmark set A of randomly generated graphs with varying edge density. The performance profiles comparing the CPU time and the number of nodes in the branch-and-bound tree for the entire benchmark set A are presented in Figure 3.

For the bimodal instances we report detailed results in Tables 1 and 2. (Detailed results for the instances with Gaussian distributed weights with clique separation turned on are reported in Table 4 of Section 5.5, where they are used for the comparison with the results from an integer LP model.) Tables 1 and 2 report the average CPU time (in seconds) and the average number of subproblems (# subs) for solving the instances to optimality. The results in each line correspond to the average over five different instances or over those instances that could be solved within the time limit. The columns entitled nrinst report the number of instances that could be solved within the time limit.

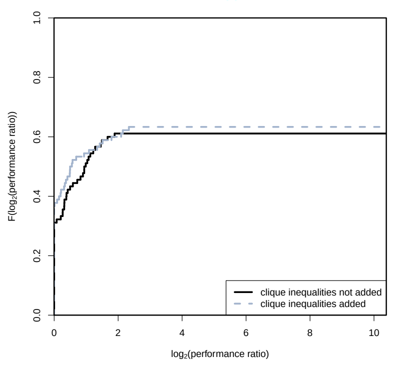
The profiles in Figure 3 show that while run times do not differ significantly, the number of subproblems is smaller when clique separation is turned on. These results are explained by the fact that a subproblem takes longer to solve when clique separation is turned on than when it is turned off. However, investing this additional CPU time may pay off if the bounds are sufficiently tighter to reduce the number of subproblems, and hence the running time of the algorithm. This effect is particularly observable for the bimodal instances in Tables 1 and 2. We see there that in most cases the number of subproblems is considerably reduced by using clique separation, and the computational time may also improve, especially for instances with edge density 25% and 50%. Moreover, there is one instance (of type $|V| = 40$ with 50% edge density) that can only be solved within the time limit when clique separation is used.

We also compared bundleBC with and without clique separation for $k = 3,5,7$ for the benchmark set B from [3]. The detailed results are reported in Table 3. These results support the same conclusions, namely that the number of subproblems is often reduced when clique separation is turned on, and the computational time frequently improved. Furthermore, three of the instances can only be solved within the time limit if clique separation is turned on (data_2g_8_37, data_2g_8_648, and data_random_40_k=3).

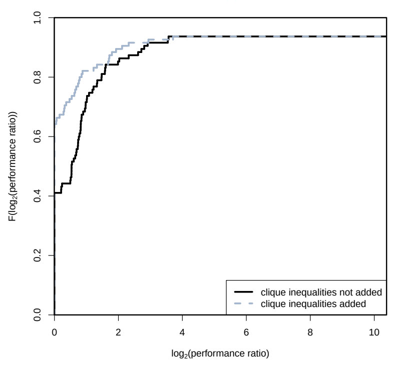
In summary, there are several instances for which the number of subproblems is not improved by the use of cliques in bundleBC. Although the heuristic separation of cliques is fast, for these cases the overall running time is obviously longer than without using cliques. On the other hand, for several instances the separation of cliques reduces the number of subproblems considerably. Furthermore, certain instances can only be solved within the time limit if clique separation is used. We conclude that it is beneficial to use clique separation in bundleBC, and we use it for all subsequent computations.



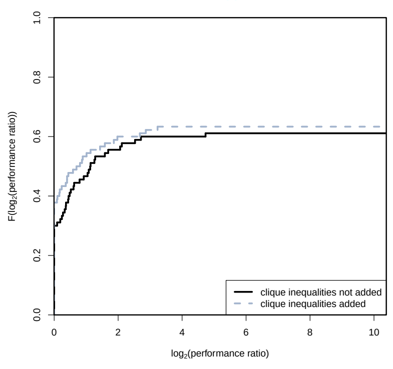
(a) CPU time, $k = 3$



(b) CPU time, $k = 5$



(c) Number of subproblems, $k = 3$



(d) Number of subproblems, $k = 5$

Fig. 3 Performance profiles concerning clique separation with respect to running time (top profiles) and the number of subproblems (bottom profiles) for the entire benchmark set A.

		without cliques			with cliques		
$\|V\|$	k	time (sec.)	# subs	nrinst	time (sec)	# subs	nrinst
10	3	0	1	5	0	1	5
20	3	0.6	3.4	5	2.2	4.2	5
30	3	35.6	45	5	35.6	29.4	5
40	3	377.2	225.4	5	728.4	279.8	5
50	3	12,925.0	4,340.2	5	10,758.8	2,467.4	5
10	5	0.0	1.0	5	0.0	1.0	5
20	5	7.0	14.6	5	9.0	14.6	5
30	5	818.6	666.2	5	640.8	473.8	5
40	5	28,198.0	12,076.0	2	23,435.0	9,457.0	2

Table 1 Results for instances of benchmark set A with 100% edge density and edge weights following a bimodal distribution.

	edge density		without cliques			with cliques		
$\|V\|$	in %	k	time (sec.)	# subs	nrinst	time (sec)	# subs	nrinst
10	50	3	0	1	5	0	1	5
20	25	3	0	1.4	5	0	1.4	5
20	50	3	0.8	3.0	5	0.2	2.2	5
30	25	3	9.4	5.8	5	8.8	3.8	5
30	50	3	26.2	28.2	5	22.4	15	5
40	25	3	169.4	60.6	5	151.6	37.8	5
40	50	3	479.0	221.0	5	846.8	185.4	5
50	25	3	3,326.6	722.2	5	2,201.6	352.6	5
50	50	3	8,851.6	2,630.6	5	8,277.8	1,519.8	5
10	50	5	0	1	5	0	1	5
20	25	5	0.2	1.4	5	0	1.4	5
20	50	5	8.8	13.4	5	14.4	19.4	5
30	25	5	18.4	9.4	5	13.2	5.8	5
30	50	5	513.6	368.2	5	321.6	209.8	5
40	25	5	3,242.8	1,033.8	5	2,865.8	804.6	5
40	50	5	6,844.5	2,092.0	2	18,451.3	5,151.7	3
50	25	5	427.0	47.0	1	1,480.0	185.0	1
50	50	5	–	–	0	–	–	0

Table 2 Results for instances of benchmark set A with varying edge density (25% and 50%) and edge weights following a bimodal distribution.

5.4 Comparison with SBC

In this section we present some comparisons of the performance of bundleBC with that of SBC [24]. For this purpose we use the results in Table 3 where we solved the instances of benchmark set B to optimality for $k = 3$, $k = 5$, and $k = 7$ using bundleBC. In contrast to SBC, we now solve the SDP-relaxations approximately in a shorter time using a bundle method. It is therefore interesting to compare SBC and bundleBC in terms of the quality of the bounds as well as the CPU times necessary to compute them. We use the percentage gap calculated with respect to the value of

	without cliques						with cliques					
	$k = 3$		$k = 5$		$k = 7$		$k = 3$		$k = 5$		$k = 7$	
instance	cpu	#subs	cpu	#subs	cpu	#subs	cpu	#subs	cpu	#subs	cpu	#subs
data_2g_3_93	0	11	0	25	1	45	0	11	0	25	0	43
data_2g_4_164	0	3	5	601	1	7	1	21	16	359	18	1,059
data_2g_5_25	2	7	17	29	3	3	6	47	38	57	8	7
data_2g_6_366	19	27	6	5	191	81	6	45	24	13	37	11
data_2g_6_66	25	15	245	77	87	35	30	21	77	23	72	21
data_2g_6_701	7	17	6	3	5	3	9	47	21	13	6	3
data_2g_7_1034	13	3	388	85	833	243	33	7	186	39	177	23
data_2g_7_491	6	3	22,661	17,339	–	–	5	3	5,724	3,623	–	–
data_2g_7_77	27	7	127	29	567	91	44	17	193	29	52	11
data_2g_8_37	792	47	1,979	261	–	–	111	9	2,355	231	30,370	6,703
data_2g_8_648	381	29	141	7	–	–	104	5	3,035	389	169	9
data_2g_8_88	28	7	2,595	253	393	33	39	7	89	7	1,439	113
data_2g_9_819	214	9	1,128	27	12,874	627	380	17	1,106	39	150	5
data_2g_9_9211	1,768	75	523	13	2,418	65	169	5	2246	51	11,607	691
data_2g_10_1001	106	5	–	–	–	–	182	7	–	–	–	–
data_2g_10_824	104	5	–	–	–	–	201	11	–	–	–	
data_2pm_4_44	0	1	1	0	0	1	0	1	0	1	0	1
data_2pm_5_55	0	1	0	1	0	1	0	1	0	1	0	1
data_2pm_6_66	16	5	10	3	7	1	11	3	11	3	11	3
data_2pm_7_777	1	1	115	17	93	13	0	1	266	29	56	7
data_2pm_8_888	254	21	493	21	309	11	116	7	245	11	303	13
data_2pm_9_999	576	21	7,215	337	3,369	127	789	29	2,507	51	963	23
data_3g_234_234	0	3	28	47	4	5	0	3	6	7	1	3
data_3g_244_244	16	15	11	7	9	7	20	11	19	17	45	43
data_3g_333_333	1	5	10	7	21	25	0	3	26	21	13	11
data_3g_334_334	23	27	84	27	51	9	16	19	40	13	36	11
data_3g_344_344	4	13	456	119	66	5	4	13	48	5	89	7
data_3g_444_444	198	7	979	37	1,678	81	222	11	1,548	73	868	29
data_3pm_234_234	0	1	0	1	0	1	0	1	0	1	0	1
data_3pm_244_244	3	1	6	1	29	7	3	1	6	1	12	7
data_3pm_333_333	3	7	0	1	0	1	0	3	0	1	0	1
data_3pm_334_334	39	13	156	19	81	9	8	3	141	35	91	27
data_3pm_444_444	11,094	887	11,955	341	11,315	303	23,667	875	12,956	429	9,843	423
data_3pm_344_344	160	21	1,618	205	1,240	191	102	13	2,269	349	2,493	413
data_3pm_345_345	162	11	3,787	303	576	41	625	53	909	39	724	57
data_clique_20	0	1	0	1	3	21	0	1	0	1	1	9
data_clique_30	0	1	4	7	15	41	0	1	3	7	5	15
data_clique_40	1	3	20	13	3	5	1	3	6	5	13	17
data_clique_50	9	9	4	3	44	27	2	3	17	9	76	45
data_clique_60	3	3	7	3	91	33	4	3	9	3	79	29
data_clique_70	36	11	17	3	152	31	20	7	78	13	276	53
data_random_20_k=3	0	11	1	1	9	51	0	3	0	1	13	67
data_random_30_k=2	39	275	614	3,231	340	1,111	32	217	588	2,673	269	825
data_random_30_k=3	32	317	1,009	5,671	944	3,339	10	73	932	3,991	1,104	3,327
data_random_40_k=2	255	1,493	–	–	2,889	5,747	130	521	–	–	4,994	9,653
data_random_40_k=3	29	145	9,057	23,877	–	–	32	137	7,925	18,243	32,168	57,489
data_random_50_k=2	2,606	8,267	–	–	–	–	5,248	14,337	–	–	–	–
data_random_50_k=3	5,248	14,337	–	–	–	–	3,963	10,741	–	–	–	–

Table 3 Number of subproblems and running times for solving the instances of benchmark set B to optimality using bundleBC for $k = 3$, $k = 5$, and $k = 7$. Instances that could not be solved to optimality within the time limit are indicated by -.

the best known primal solution prim as the ratio

$$\text{gap} = \left| \frac{\text{bound at root} - \text{prim}}{\text{prim}} \right| .$$

We first comment on the quality of the bounds and the running times of bundleBC at the root node only, i.e., before any branching is done. The root node bounds for SBC are typically zero or close to zero for two- and three-dimensional grids so that branching rarely takes place [24]. However, their computation can take very long. For example, it took SBC more than ten hours to reach a gap of 1% at the root node for an instance with $k = 3$ on a 10^2 grid with Gaussian distributed edge weights [24, Table 5], while bundleBC solved the instances from [3] defined on 10^2 grids to optimality within at most 4 minutes and needing only up to 11 subproblems (see Table 3). Hence, while the bounds computed by bundleBC are weaker than those determined by SBC, their calculation is much faster and this makes them more useful within a branch-and-bound procedure.

Turning to the solution of instances to optimality, we recall from [24] that SBC almost never has to branch. However, several instances could not be solved within 24 hours of computation time. While it is not surprising that bundleBC branches more often than SBC, bundleBC gains from the fact that the number of subproblems is often in the order of several hundreds only for this set of instances. Because computing one subproblem is much faster for bundleBC than for SBC for small k, bundleBC achieves a drastic speedup over SBC. On the other hand, the instances get more difficult for bundleBC as k increases, unlike what we observed for SBC in [24].

5.5 Comparison with [31]

Kaibel et al. [31] report experimental results on max-k-cut instances for an LP-based branch-and-cut algorithm using orbitopal fixing (OF). Furthermore, some of the instances in [3] were addressed in the more recent paper [32]. In particular, several of the Gaussian distributed instances defined on regular grids were solved using OF in very short computing times.

We evaluated bundleBC (with clique separation active) on the benchmark set C of instances from [31]. Our results are reported in Table 4 where the first three columns indicate for each line the number of nodes ($|V|$), the number of edges ($|E|$) and the value of k of the instances averaged. The subsequent two columns report the number of subproblems and the CPU time of bundleBC for solving the instances to optimality. Similarly to what was done in [31], averages were taken over three instances for each row unless some of the instances could not be solved to optimality within the time limit, in which case the number of instances over which the average is taken is denoted in the last column.

			bundleBC		
$\|V\|$	$\|E\|$	k	time (sec.)	# subs	nrinst
30	200	3	60.6	99.0	3
30	300	3	41.0	102.3	3
30	400	3	13.6	50.3	3
50	560	3	17,497.5	10,167.0	2
30	200	6	472.7	681.0	3
30	300	6	324.3	558.3	3
30	400	6	627.7	1,691.0	3
50	560	6	–	–	0
30	200	9	–	–	0
30	300	9	1,514.7	2,610.3	3
30	400	9	1,181.3	2,554.3	3
50	560	9	–	–	0
30	200	12	–	–	0
30	300	12	–	–	0
30	400	12	462.0	898.3	3
50	560	12	–	–	0

Table 4 Results for the instances of benchmark set C.

We first compare the performance of bundleBC with that reported in [31]. The instances classified as 'easy' in [31] with $|V| = 30, |E| = 200$ need almost always zero time with the OF approach for all considered values of k, whereas for bundleBC not all of these instances are easy. In fact, for $k = 9$ and $k = 12$, bundleBC timed out on all three instances. On the other hand, for the instances with $|V| = 30, |E| = 400$ that are denser and more difficult for [31], it is clear that bundleBC needs fewer subproblems than OF. Moreover, the average number of subproblems for OF are $9,864 (k = 3)$, $159,298 (k = 6)$, $70,844 (k = 9)$, and $2,098 (k = 12)$. The fact that for bundleBC these numbers are almost always at least one order of magnitude lower shows that our bounds are considerably stronger than those generated by OF.

Computation times are trickier to compare because we used different modern machines from [31]. Nevertheless, it seems that for $k = 3$ and the medium-sized instances with 30 nodes our approach performs better than OF, especially for denser graphs. On the other hand, their performance is better for larger values of k as OF can exploit symmetries well, while these instances are more difficult for bundleBC. Finally, we cannot solve the most difficult instances from [31] that can be solved within several hours of computing time with OF.

Even though we do not have the implementation of the algorithm from [31] at hand, we implemented their integer LP model with variables x_{ip} specifying whether node i is contained in partition p or not. This formulation is polynomially-sized in the x_{ip} variables. As a comparison with our approach, we solve this integer LP problem with CPLEX 12.1 using the standard parameter settings and 6 cores per job. For each job a real time limit of 10 hours was imposed. For larger values of k, the integer LP model is not effective in practice because of the symmetries that are present. However, for small values of k such as $k = 3$ and $k = 5$, the speedup achieved by using OF is not so significant and hence the results are likely more representative of the behaviour of OF [31].

We used our benchmark set A of instances to test this model. The results for both CPLEX and bundleBC on the subset of instances that have Gaussian distributed edge weights are reported in Tables 5 and 6 where we average for each row over 5 instances or the number of instances that finished within the time limit. It turned out that CPLEX ran out of memory even for relatively small instances; for example, for an instance with $k = 3$ and $|V| = 40$, CPLEX ran out of memory after more than 8000 seconds and still had a gap of 43%. The results of CPLEX for the instances with bimodal edge weights have a similar outcome, as shown in Tables 7 and 8. This is in marked contrast with bundleBC that solved many of these instances within a few minutes to optimality.

		CPLEX		bundleBC		
$\|V\|$	k	real time(sec.)	nrinst	real time(sec)	# subs	nrinst
10	3	0.03	5	0	1	5
20	3	17.96	5	4.2	15.8	5
30	3	8,113.31	5	72.4	38.2	5
40	3	–	0	375.6	119.8	5
50	3	–	0	8228.0	1582.3	3
10	5	0.14	5	0.2	7.0	5
20	5	6,674.24	3	16.8	37.4	5
30	5	–	0	1,126.8	807.0	5
40	5	–	0	11,489.0	2,919.0	3

Table 5 Results for instances of benchmark set A with 100% edge density and edge weights following a Gaussian distribution.

6 Conclusion

We extended the SBC algorithm of Ghaddar, Anjos and Liers for minimum k-partition using the design principles of the successful BiqMac solver for maximum 2-cut to obtain bundleBC, a new algorithm for computing global optimal solutions for maximum k-cut problems. As part of this extension, we investigated different ways of choosing variables for branching. We also studied the impact of the separation of clique inequalities within this new framework and observed that it frequently reduces the number of subproblems considerably. The computational results suggest that bundleBC achieves a significant speedup in comparison to SBC, especially when $k = 3$. A comparison with the results reported from the application of the OF technique by Kaibel, Peinhardt and Pfetsch suggests that while their performance is better for sparse instances and larger values of k, bundleBC is superior for $k = 3$ and for dense instances of medium size. Solving the ILP formulation for max-k-cut used by Kaibel, Peinhardt and Pfetsch with CPLEX clearly demonstrates the advan-

	edge density		CPLEX		bundleBC		
$\lvert V\rvert$	in %	k	time (sec.)	nrinst	time (sec)	# subs	nrinst
10	50	3	0.02	5	0.0	20.6	5
20	25	3	0.03	5	1.0	17.0	5
20	50	3	1.69	5	4.6	30.6	5
30	25	3	2.61	5	11.4	8.2	5
30	50	3	184.48	5	24.6	18.2	5
40	25	3	95.90	5	108.8	147.8	5
40	50	3	–	0	265.0	455.0	5
50	25	3	9,763.68	3	2,833.4	5,774.2	5
50	50	3	–	0	10,237.6	17,323.8	5
10	50	5	0.03	5	0.4	80.2	5
20	25	5	0.08	5	16.0	174.6	5
20	50	5	7.07	5	4.4	9.4	5
30	25	5	85.94	5	50.6	28.6	5
30	50	5	–	0	124.2	67.8	5
40	25	5	–	0	1,609.2	1,672.2	5
40	50	5	–	0	2,081.6	606.6	5
50	25	5	–	0	3,780.7	24,272.3	3
50	50	5	–	0	–	–	0

Table 6 Results for instances of benchmark set A with varying edge density (25% and 50%) and edge weights following a Gaussian distribution.

		CPLEX	
$\lvert V\rvert$	k	time (sec.)	nrinst
10	3	0.03	5
20	3	88.76	5
30	3	6,053.06	5
40	3	–	0
50	3	–	0
10	5	0.26	5
20	5	–	0
30	5	–	0
40	5	–	0

Table 7 Results for instances of benchmark set A with 100% edge density and edge weights following a bimodal distribution. The results for solving these instances with bundleBC are given in Table 1.

tage of our semidefinite approach. The strength of bundleBC is especially evident on dense instances and small values of k.

Acknowledgements We are grateful to Vera Schmitz for providing us with her implementation of a generic branch-and-bound procedure and to Andreas Schmutzer for help with various aspects of the implementation. We thank Brian Borchers and Christoph Helmberg for support with csdp and conic bundle respectively. We thank an anonymous referee for detailed criticism that helped improve the paper. We also thank Matthias Peinhardt for providing us with data for the instances from [31]. Finally we acknowledge the financial support of the German Science Foundation under contract Li 1675/1 and of the Natural Sciences and Engineering Research Council of Canada.

$\lvert V\rvert$	edge density in %	k	CPLEX time (sec.)	nrinst
10	50	3	0.02	5
20	25	3	0.05	5
20	50	3	2.40	5
30	25	3	4.48	5
30	50	3	1,215.32	5
40	25	3	1,504.37	5
40	50	3	–	0
50	25	3	29,817.4	1
50	50	3	–	0
10	50	5	0.04	5
20	25	5	0.27	5
20	50	5	405.55	5
30	25	5	1624.80	4
30	50	5	4459.24	1
40	25	5	5299.18	3
40	50	5	–	0
50	25	5	–	0
50	50	5	–	0

Table 8 Results for instances of benchmark set A with varying edge density (25% and 50%) and edge weights following a bimodal distribution. The results for solving these instances with bundleBC are given in Table 2.

References

1. *BiqMac solver. http://biqmac.uni-klu.ac.at*, retrieved 07/06/2012.
2. *Conic Bundle Library. http://www-user.tu-chemnitz.de/∼helmberg/ConicBundle/*, retrieved 28/10/2011.
3. max-*k*-cut instances, from http://www.eng.uwaterloo.ca/∼bghaddar/Publications.htm , retrieved 10/03/2011.
4. *Spin-glass server. http://www.informatik.uni-koeln.de/ls_juenger/research/sgs/index.html*, retrieved 07/06/2012.
5. M.F. Anjos, F. Liers, G. Pardella, and A. Schmutzer. Engineering branch-and-cut algorithms for the equicut problem. Cahier du GERAD G-2012-15, GERAD, Montreal, QC, Canada, 2012. To appear in the Fields Institute Communications on Discrete Geometry and Optimization.
6. M.F. Anjos and H. Wolkowicz. Geometry of semidefinite max-cut relaxations via matrix ranks. *J. Comb. Optim.*, 6(3):237–270, 2002.
7. M.F. Anjos and H. Wolkowicz. Strengthened semidefinite relaxations via a second lifting for the max-cut problem. *Discrete Appl. Math.*, 119(1–2):79–106, 2002.
8. M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. LP and SDP branch-and-cut algorithms for the minimum graph bisection problem: a computational comparison. *Math. Program. Comput.*, 4(3):275–306, 2012.
9. F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988.
10. F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.
11. B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11/12(1-4):613–623, 1999.

12. E. Boros and P. Hammer. The max-cut problem and quadratic 0-1 optimization: Polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, 33:151–180, 1991.
13. L. Brunetta, M. Conforti, and G. Rinaldi. A Branch-And-Cut Algorithm for the Equicut Problem. *Math. Program. B*, 78(2):243–263, 1997.
14. S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59:87–115, 1993.
15. S. Chopra and M. R. Rao. Facets of the *k*-partition problem. *Discrete Applied Mathematics*, 61:27–48, 1995.
16. E. de Klerk, D.V. Pasechnik, and J.P. Warners. On approximate graph colouring and Max-*k*-Cut algorithms based on the ϑ-function. *Journal of Combinatorial Optimization*, 8(3):267–294, 2004.
17. M. Deza, M. Grötschel, and M. Laurent. Complete descriptions of small multicut polytopes. *Applied Geometry and Discrete Mathematics - The Victor Klee Festschrift*, 4:205–220, 1991.
18. M. Deza and M. Laurent. *Geometry of Cuts and Metrics*. Algorithms and Combinatorics, Springer Verlag, 1997.
19. E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002.
20. A. Eisenblätter. The semidefinite relaxation of the *k*-partition polytope is strong. *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, 2337:273–290, 2002.
21. M. Elf, M. Jünger, and G. Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31(5):343–349, 2003.
22. 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.
23. A. Frieze and M. Jerrum. Improved approximation algorithms for max *k*-cut and max bisection. *Algorithmica*, 18:67–81, 1997.
24. B. Ghaddar, M.F. Anjos, and F. Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum *k*-partition problem. *Annals of Operations Research*, 188(1):155–174, 2011.
25. M. Goemans and D. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7(4):656–666, 1994.
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 K. C. Kiwiel. A spectral bundle method with bounds. *Math. Program.*, 93(2, Ser. A):173–194, 2002.
28. C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3, Series A):291–315, 1998.
29. C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM J. Optim.*, 10(3):673–696 (electronic), 2000.
30. 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.
31. V. Kaibel, M. Peinhardt, and M.E. Pfetsch. Orbitopal fixing. In Matteo Fischetti and David Williamson, editors, *Integer Programming and Combinatorial Optimization*, volume 4513 of *Lecture Notes in Computer Science*, pages 74–88. Springer Berlin / Heidelberg, 2007.
32. V. Kaibel, M. Peinhardt, and M.E. Pfetsch. Orbitopal fixing. *Discrete Optim.*, 8(4):595–610, 2011.
33. K. C. Kiwiel. *Methods of descent for nondifferentiable optimization*, volume 1133 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1985.
34. M. Laurent. Semidefinite relaxations for max-cut. In *The sharpest cut*, MPS/SIAM Ser. Optim., pages 257–290. SIAM, Philadelphia, PA, 2004.
35. M. Laurent and S. Poljak. On a positive semidefinite relaxation of the cut polytope. *Linear Algebra Appl.*, 223/224:439–461, 1995.
36. M. Laurent and S. Poljak. On the facial structure of the set of correlation matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):530–547, 1996.

37. C. Lemaréchal. Bundle methods in nonsmooth optimization. In *Nonsmooth optimization (Proc. IIASA Workshop, Laxenburg, 1977)*, volume 3 of *IIASA Proc. Ser.*, pages 79–102. Pergamon, Oxford, 1978.
38. C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Math. Programming*, 69(1, Ser. B):111–147, 1995.
39. F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. *Computing Exact Ground States of Hard Ising Spin Glass Problems by Branch-and-Cut*, pages 47–68. New Optimization Algorithms in Physics, Wiley, 2004.
40. F. Liers, J. Lukic, E. Marinari, A. Pelissetto, and E. Vicari. Zero-temperature behavior of the random-anisotropy model in the strong-anisotropy limit. *Physical Review B*, 76(17):174423, 2007.
41. A. Lisser and F. Rendl. Telecommunication clustering using linear and semidefinite programming. *Mathematical Programming*, 95:91–101, 2003.
42. F. Margot. Pruning by isomorphism in branch-and-cut. *Math. Program.*, 94(1, Ser. A):71–90, 2002.
43. F. Margot. Exploiting orbits in symmetric ILP. *Math. Program.*, 98(1-3, Ser. B):3–21, 2003. Integer programming (Pittsburgh, PA, 2002).
44. J. Mitchell. Branch-and-cut for the *k*-way equipartition problem. Technical report, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 2001.
45. J. E. Mitchell. Realignment in the National Football League: Did they do it right? *Naval Research Logistics*, 50(7):683–701, 2003.
46. L. Palagi, V. Piccialli, F. Rendl, G. Rinaldi, and A. Wiegele. Computational approaches to max-cut. In *Handbook on semidefinite, conic and polynomial optimization*, volume 166 of *Internat. Ser. Oper. Res. Management Sci.*, pages 821–847. Springer, New York, 2012.
47. S. Poljak and F. Rendl. Solving the max-cut problem using eigenvalues. *Discrete Appl. Math.*, 62(1-3):249–278, 1995. Partitioning and decomposition in combinatorial optimization.
48. F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121:307–335, 2010.
49. G. Rinaldi. *Rudy. http://www-user.tu-chemnitz.de/∼helmberg/rudy.tar.gz* , retrieved 07/04/2010.
50. H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results. *SIAM J. Optim.*, 2(1):121–152, 1992.