A Branch-and-Cut Algorithm based on Semidefinite Programming for the Minimum k-Partition Problem[1]

Bissan Ghaddar[2], Miguel Anjos[3], and Frauke Liers[4]

Abstract

The minimum k-partition (MkP) problem is the problem of partitioning the set of vertices of a graph into k disjoint subsets so as to minimize the total weight of the edges joining vertices in the same partition. The main contribution of this paper is the design and implementation of a branch-and-cut algorithm based on semidefinite programming (SBC) for the MkP problem. The two key ingredients for this algorithm are: the combination of semidefinite programming (SDP) with polyhedral results; and the iterative clustering heuristic (ICH) that finds feasible solutions for the MkP problem. We compare ICH to the hyperplane rounding techniques of Goemans and Williamson and of Frieze and Jerrum, and the computational results support the conclusion that ICH consistently provides better feasible solutions for the MkP problem. ICH is used in our SBC algorithm to provide feasible solutions at each node of the branch-and-bound tree. The SBC algorithm computes globally optimal solutions for dense graphs with up to 60 vertices, for grid graphs with up to 100 vertices, and for different values of k, providing the best exact approach to date for $k \geq 3$.

***Keywords*:** semidefinite programming, branch-and-cut, polyhedral cuts.

1 Introduction

The minimum k-partition problem (MkP) is a well-known optimization problem encountered in various applications such as telecommunication and physics. It is known to be $\mathcal{NP}$-hard in general and difficult to solve in practice. The MkP is equivalent to finding a maximum k-cut, where the weighted sum of all edges with their endpoints in distinct sets is maximized. It has applications in network planning [12], VLSI layout design [3], micro-aggregation of statistical data [11], sports team scheduling [23, 13], physics [19], and other areas. Several authors, including Barahona and Mahjoub [4], Deza and Laurent [10], and Boros and Hammer [6], studied the problem of partitioning a graph into two subsets. The special case with k=2 is known as the max-cut problem and is equivalent to unconstrained binary quadratic optimization.

The maximum k-cut problem has received more attention in the literature than the minimum k-partition problem, such as in Deza, Grötschel, and Laurent [9], Chopra and Rao [8] and the book by Deza and Laurent [10]. The minimum k-partition problem is formulated by Chopra and Rao in [7] where several valid and facet-defining inequalities are identified.

Mitchell [22] applied a linear programming (LP) based branch-and-cut algorithm to the k-way equipartition problem with application to the National Football League (NFL). The k-way equipartition problem is an MkP problem with an additional constraint that partitions have to be of the same size. Computational results 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

[1]Partially supported by the Marie Curie RTN 504438 (ADONET) funded by the European Commission. BG and MA were supported by NSERC Discovery Grant 312125 and MITACS Network of Centres of Excellence. FL was supported by the German Science Foundation under contract Li 1675/1

[2]Department of Management Sciences, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada, N2L 3G1, bghaddar@uwaterloo.ca

[3]Department of Management Sciences, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, Canada, N2L 3G1, manjos@uwaterloo.ca

[4]Institut für Informatik, Universität zu Köln Pohligstr. 1 D-50969 Köln, Germany, liers@informatik.uni-koeln.de

graphs of sizes 100 to 500. Moreover, Lisser and Rendl [20] described a telecommunication application for the k-way equipartition problem. They investigated both semidefinite and linear relaxations of the problem with iterative cutting plane algorithms. For graph sizes ranging from 100 to 900 vertices and k=5, 10, the SDP approach produces a gap between 4%-6% from the optimal solution and is better than the LP approach.

For $k = 2$, linear bounds are strong. As they also can exploit sparsity, sparse instances can usually be solved faster with linear than with SDP-based methods. On the other hand, SDP-based methods perform better for dense instances. Both a linear [2, 19] and an SDP-based server [1] are available in public domain. The former is especially designed for fast solutions of instances defined on grids that have application in physics [19]. The latter can solve max-cut instances of graphs of any structure up to 100 vertices.

While effective computational procedures that yield globally optimal solution for arbitrary instances of 100 vertices and sparse graphs of larger sizes have been implemented for the k=2 case, to our knowledge all the procedures proposed in the literature either can't be applied to general k, provide no guarantee for global optimality, or enforce additional constraints.

In this work, we present an exact algorithm for the minimum k-partition problem that uses positive semidefinite relaxations. We found experimentally that for $k > 2$ they yield much stronger bounds than linear relaxations, for both sparse and dense instances.

This paper is organized as follows. In Section 2, technical definitions and an overview over the literature on the MkP problem are given. The SDP-based branch-and-cut algorithm and the primal heuristic are presented in Section 3. In Section 4, the heuristic is compared to the hyperplane rounding of Goemans and Williamson [16] and Frieze and Jerrum [14] in terms of bounds. In addition, computational results for the branch-and-cut algorithm on several important classes of instances, and for different values of k, are presented. The computational results show the potential of SBC for tackling the MkP problem. Finally, conclusions and future research directions are discussed in Section 5.

2 Problem Description and Some Related Previous Results

An instance of the minimum k-partition problem consists of 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 $V_1, ..., V_k$ such that $\sum_{l=1}^{k} \sum_{i,j \in V_l} w_{ij}$ is minimized. An example is shown in Figure 1.

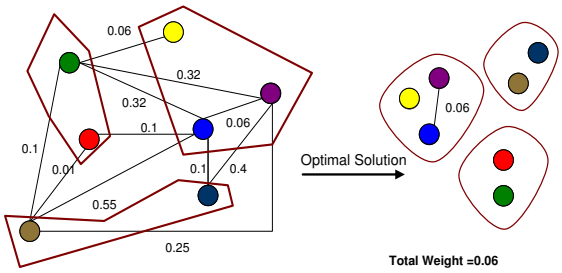

Figure 1: A k-partition of a graph with $|V| = 7$ and $k = 3$. The solution value is 0.06.

Without loss of generality the graph G can be completed to $K_{|V|}$ by adding zero-weight edges. The edge set is then $E = \{ij \mid 1 \leq i < j \leq n\}$. Define the variable z_{ij} as

$$z_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same partition,} \\ 0 & \text{otherwise.} \end{cases}$$

Chopra and Rao considered in [8] the following integer linear programming (ILP) formulation for MkP:

$$\begin{aligned}
\textbf{(ILPMKP)}\quad \min\ & \sum_{i,j\in V} w_{ij} z_{ij} && && (1)\\
\text{s.t. } & z_{ih} + z_{hj} - z_{ij} \le 1 && \forall\, h,i,j \in V && (2)\\
& \sum_{i,j\in Q} z_{ij} \ge 1 && \forall\, Q \subseteq V \text{ where } |Q| = k+1 && (3)\\
& z_{ij} \in \{0,1\} && \forall\, i,j \in V, &
\end{aligned}$$

where inequalities (2) and (3) are the triangle and clique inequalities, respectively. Constraint (2) requires the values of the variables to be consistent. For example, if z_{ih} and z_{hj} indicate that i, h, and j are in the same partition, then by transitivity the value of z_{ij} has to reflect that as well. Constraint (3) imposes that at least two from every subset of $k+1$ vertices have to be in the same partition. Together with the constraints (2), this implies that there are at most k partitions. There are $3\binom{|V|}{3}$ triangle inequalities and $\binom{|V|}{k+1}$ clique inequalities.

Here we are interested in exact solutions that we generate with a branch-and-cut algorithm. The latter is an often successful framework in combinatorial optimization. Usually, (linear programming) relaxations are used and strengthened during the run of the algorithm. However, we found for MkP that linear bounds obtained by relaxing the integrality constraints in (ILPMKP) are weak in practice which could result in complete enumeration of all solutions. Furthermore, we found experimentally that the semidefinite relaxation bound that we introduce next is much stronger than the LP bound [15]. This motivates us to use SDP relaxations within branch-and-cut for the MkP problem.

2.1 SDP Relaxation for the MkP Problem

Semidefinite programming relaxations of combinatorial optimization problems were pioneered by Lovasz [21] in 1979 in order to compute the Shannon capacity of a graph. Moreover, Goemans and Williamson [16] used SDP to provide a performance guarantee of an approximation algorithm for the satisfiability and the max-cut problem. The latter lead to a rapid growth of the field. The MkP problem was formulated in [12] using SDP as follows:

$$\begin{aligned}
\min\ & \sum_{i,j\in V,\, i<j} w_{ij} \frac{(k-1)X_{ij}+1}{k} && && (4)\\
\text{s.t. } & X_{ii} = 1 && \forall i \in V && (5)\\
& X_{ij} \in \{\frac{-1}{k-1},\, 1\} && \forall i,j \in V,\, i<j && (6)\\
& X \succeq 0, &&
\end{aligned}$$

where $X_{ij} = \frac{-1}{k-1}$ can be interpreted as vertices i and j being in different partitions and $X_{ij} = 1$ means that they are in the same partition. Replacing constraint (6) by $\frac{-1}{k-1} \le X_{ij} \le 1$ results in a semidefinite relaxation. However, constraint $X_{ij} \le 1$ 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:

$$\begin{aligned}
\textbf{(SMKP)}\ \min\ & \sum_{i,j\in V,\, i<j} w_{ij} \frac{(k-1)X_{ij}+1}{k} && && (7)\\
\text{s.t. } & X_{ii} = 1 && \forall i \in V && (8)\\
& X_{ij} \ge \frac{-1}{k-1} && \forall i,j \in V,\, i<j && (9)\\
& X \succeq 0. &&
\end{aligned}$$

The SDP relaxation can be further tightened by adding valid inequalities, i.e., inequalities that are satisfied for all positive semidefinite matrices that are feasible for the original SDP. The two types of valid inequalities added are the triangle and the clique inequalities formulated for SDP. Observing that in any cycle of length three exactly zero or two edges are cut, the triangle inequalities have the form:

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

where i, j, and $h \in V$. It is not hard to see that the clique inequalities take the form:

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

To verify validity, recall that the clique inequalities ensure that for every set $Q \subseteq V$ with $|Q| = k+1$ at least 2 vertices have to be in the same partition. This means that at least one X_{ij} equals 1. Therefore,

$$\begin{aligned}
&\sum_{i,j \in Q,\, i<j} X_{ij} \geq 1 + \sum_{i=1}^{\binom{k+1}{2}-1} \frac{-1}{k-1} \quad \forall\, Q \subseteq V \text{ where } |Q| = k+1. \\
\Leftrightarrow &\sum_{i,j \in Q,\, i<j} X_{ij} \geq 1 + \left[\frac{(k+1)k}{2} - 1\right] \frac{-1}{k-1} \\
\Leftrightarrow &\sum_{i,j \in Q,\, i<j} X_{ij} \geq \frac{-k}{2}.
\end{aligned}$$

The validity of the triangle inequality can be verified similarly. Once the (SMKP) relaxation is solved, one can separate violated triangle and clique inequalities. Adding them to the SDP problem will strengthen the relaxation.

2.2 Approximation Algorithm for Max k-Cut

In the previous section we discussed how to obtain a lower bound for the MkP problem. In this section and in Section 3.2 we give an overview over an approximation algorithm and our ICH heuristic that can be used to obtain an upper bound for this problem.

Goemans and Williamson [16] used semidefinite programming in the design of a randomized approximation algorithm for the max-cut problem which always produces solutions of expected value of at least 0.87856 times the optimal value. This was the first time that a performance guarantee could be given by semidefinite programming for an $\mathcal{NP}$-hard optimization problem. The results in [16] showed that the cut generated using the randomized algorithm was in the range of 4% to 9% away from the semidefinite bound in practice. Hence, it is an effective heuristic technique for generating cuts.

Frieze and Jerrum presented in [14] an extension of [16] to obtain a polynomial-time approximation algorithm for the max k-cut problem. They consider the following SDP relaxation:

$$\textbf{(MkC-SDP)} \qquad \max \frac{k-1}{k} \sum_{i,j \in V,\, i<j} w_{ij}(1 - X_{ij}) \tag{10}$$

$$\text{s.t. } X_{ij} \geq \frac{-1}{k-1} \qquad \forall i, j \in V,\, i < j \tag{11}$$

$$X_{ij} \succeq 0. \tag{12}$$

It can be easily shown that (MkC-SDP) is equivalent to the MkP formulation described earlier. Frieze and Jerrum described a rounding heuristic based on the SDP relaxation that can be used to obtain a feasible solution of the max k-cut problem. This method works as follows:

1. Solve (MkC-SDP) to get an optimal solution, $X = (X_{ij})$. Find unit vectors $v_1, \ldots, v_n \in \mathbb{R}^n$ satisfying $v_i^T v_j = X_{ij}$ where $i, j \in V$. This can be done by computing the Cholesky factorization $V^T V$ of X.

2. Choose k independent random vectors $r_1, \ldots, r_k \in \mathbb{R}^n$.
3. Partition V into $\mathcal{V}_k = \{V_1, \ldots, V_k\}$ according to $V_j = \{i : v_i \cdot r_j \geq v_i \cdot r_{j'}, \text{ for } j \neq j'\}$ for $1 \leq j \leq k$. For this we would additionally need $\parallel r_i \parallel = 1 \, \forall i = 1, \ldots, k$, however this complicates the analysis. So the kn components of $r_1, \ldots, r_k$ are chosen as independent random variables from a standard normal distribution with mean 0 and variance 1.

The authors proved in [14] the existence of a sequence of constants $\alpha_{(k \geq 2)}$ such that:

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

where $w(\mathcal{V}_k) = \sum_{1 \leq r < s \leq k} \sum_{i \in V_r, j \in V_s} w_{ij}$, $\mathcal{V}_k^*$ determines an optimal cut, and $\mathbf{E}$ denotes the expected value. In [14], it was shown that the sequence of α_k satisfies the following theorem:

Theorem 1 *[14] α_k satisfies*

1. $\alpha_k > \frac{k-1}{k}$
2. $\alpha_k - \frac{k-1}{k} \sim \frac{2 \ln k}{k^2}$
3. $\alpha_2 \geq 0.878567 \qquad \alpha_3 \geq 0.800217 \qquad \alpha_4 \geq 0.850304 \qquad \alpha_5 \geq 0.874243$

The process of Frieze and Jerrum can be iterated by varying the random vectors $r_1, \ldots, r_k$ and taking the best solution (i.e., minimum upper bound). The cut obtained by this hyperplane rounding technique may be further improved in practice by local improvement steps.

α_k has the lowest value when $k = 3$, which is $\alpha_3 \geq 0.800217$. This means that hyperplane rounding yields the weakest guarantee for the 3-partition problem.

3 An SDP-based Branch-and-Cut Framework for the MkP Problem

During the run of the branch-and-cut algorithm, a sequence of relaxations of the original problem is solved at each node of the branch-and-bound tree. Cutting-planes are used to improve the relaxations, tightening the bounds. The branch-and-bound part of the algorithm guarantees that a globally optimum solution is obtained.

In this work, we use SDP relaxations within a branch-and-cut framework since we found experimentally that they are stronger than the corresponding linear bounds. The root node of the branch-and-bound tree is the original SDP relaxation (SMKP). In each iteration, we separate valid inequalities, add them to the relaxation and resolve the SDP. If a feasible partition can be computed in the root node, we terminate. Otherwise, when no more violated inequalities can be generated, the algorithm branches. In the branching step, two subproblems are created by fixing an infeasible variable (i.e., a variable that is neither 1 nor $\frac{-1}{k-1}$ in the optimal solution of the SDP relaxation) to 1 in one subproblem and to $\frac{-1}{k-1}$ in the other. This means that in one subproblem we force vertices i and j to be in the same partition and in the other to different partitions. The sub problems are solved recursively. The branch-and-cut algorithm stops when all subproblems have been fathomed. A subproblem is fathomed if it is either infeasible, determines a feasible partition, or if we can conclude that it does not contain an optimum solution. The incumbent solution is the best solution (giving an upper bound, since we are minimizing) found so far in the tree. After termination, the incument is a globally optimum solution.

In the following sections, we describe in detail our branch-and-cut technique using SDP as the bounding procedure. The addition of triangle and clique inequalities at each node markedly improves the SDP lower bound. Moreover, at each node a feasible solution is computed to get an upper bound.

3.1 Separation of Valid Inequalities

As discussed earlier, the SDP relaxation can be further tightened by adding valid inequalities. Once (SMKP) is solved, one can check for violated triangle and clique inequalities and add them to the SDP problem, hence getting a better lower bound.

The number of triangle and clique inequalities added at each iteration depends on the size of the problem. We use complete enumeration for adding triangle inequalities. The triangle inequalities are sorted by the magnitude of the violation and added starting with the most violated ones. If not enough triangle inequalities are violated, we add clique inequalities.

Exact separation of clique inequalities is an $\mathcal{NP}$-hard problem, and exact enumeration becomes intractable already for small values of k. Therefore, we design a heuristic separation that generates inequalities that are 'important' in practice. It does not necessarily determine a violated inequality whenever one exists, however we find that it is fast and yields good bounds.

In order to find which clique inequalities are important in practice, we conducted several experiments in which we enumerated and added all violated clique inequalities. We assume that an inequality is important if it is binding at the optimum of the resolved problem, i.e. if it is satisfied with equality. We found that the binding clique inequalities usually cover the whole graph, and that each vertex in the graph is contained in several different clique inequalities. So the heuristic separation is designed to imitate this behavior as follows. For each vertex v in the graph, we grow a clique of size $k+1$ containing v. Vertices are added to the cliques in a greedy fashion. In each iteration, we add the vertex 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 Algorithm 1.

Algorithm 1 Heuristic for separating clique inequalities for (SMKP-C)

1. Given the graph $G = (V, E)$, let v_j be a vertex of G.
2. Initialize $j = 1$.
3. Let Q be the set of vertices that form a clique, $Q = \phi$.
4. Add vertex v_j to Q.
5. Choose vertex $v_{j'}$ with the smallest $\sum_{v_j \in Q, v_{j'} \in V \setminus Q} X_{jj'}$ value.
6. Add vertex $v_{j'}$ to the set Q.
7. If $|Q| < k+1$ go to step 5.
8. If violated, add the clique inequality formed to the set of inequalities.
9. If $j < |V|$ increment j, empty the clique Q, and go to step 4.

The separation routine consists of two parts: first the algorithm searches for violated triangle inequalities as described above. If no more than ρ triangle inequalities are added, the heuristic is used to find violated clique inequalities. If less than ρ inequalities are found, we branch. In the computational experiments of Section 4, ρ is set to 200. The triangle inequalities are added first since we experimentally found that they are stronger than the clique inequalities.

3.2 ICH: An Iterative Clustering SDP-based Heuristic

The ICH heuristic is designed to find a feasible solution from the optimum solution of the SDP relaxation at each node of the tree. It is a recursive procedure that groups vertices together to form a graph of smaller size and then it is recursively applied on the smaller graph until the desired partition size is reached. Given a graph $G(V, E)$ with n vertices, weights w_{ij} between edges, and number of partitions k, the heuristic is described in Algorithm 2. The intuition behind this approach is the use of aggregate information which is more reliable than single elements of data. When we sum the X^*_{ij} values on the edges between three vertices, we have a better idea of whether or not these three vertices should be in the same partition than by looking at each edge separately. The sorting of the data is done to take advantage of the best information first and use the less certain information only if necessary. An illustration of the algorithm is shown in Figure 2.

Algorithm 2 ICH Heuristic

1. Initialize a parameter r, the current number of partitions, to zero.
2. Initialize a parameter m, the current number of nodes, to n.
3. Solve the SDP relaxation with m nodes and get the optimal solution X^*.
4. Take each triplet of vertices i, j, and h and sum the values on their edges: $T_{ijh} = X^*_{ij} + X^*_{ih} + X^*_{jh}$.
5. Sort the values of T_{ijh}.
6. (a) Choose vertices i, j, and h with $T_{ijh} \geq tol$ to be in the same partition.
   (b) If any vertices remain unassigned to a partition, choose vertices with $T_{ijh} \leq tol$ to be in separate partitions.
   (c) Update r to be the number of current partitions.
7. If $r > k$,
   (a) Aggregate the vertices that are in the same partition to form one new vertex i'.
   (b) Update the value of m and the aggregate weight matrix $\bar{W}$.
   (c) Return to step 3.
8. End.

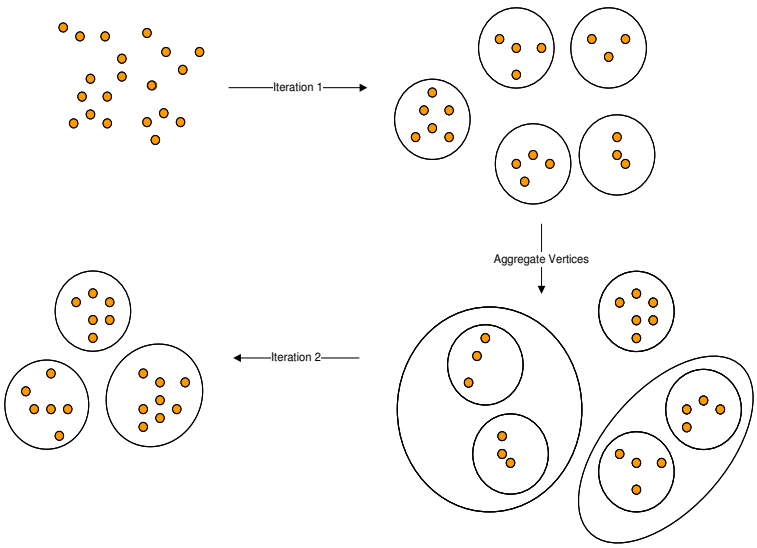

Figure 2: The ICH heuristic example with $n = 20$ and $k = 3$.

3.2.1 The ICH Heuristic with Convex Combination

The convex combination technique to improve on the Goemans-Williamson hyperplane rounding was proposed and implemented for $k = 2$ in [25]. Using this convex combination technique results in a better solution than using only hyperplane rounding. This motivated us to apply the convex combination idea to the Frieze and Jerrum [14] algorithm presented in Section 2.2.

Given the SDP solution matrix X_1^* and the hyperplane rounding feasible solution matrix $X_1^{feasible}$, we take their convex combination to obtain the following matrix:

$$X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}$$

Next we take matrix X_2 and perform the hyperplane rounding technique on this matrix to get a new feasible solution.

Similarly, we applied the convex combination technique to the ICH heuristic. Taking the feasible solution matrix $X_1^{feasible}$ obtained from the ICH heuristic and the SDP solution matrix, X_1^*, we consider a convex combination of the following form:

$$X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}.$$

Then we can apply the ICH heuristic to the X_2 matrix to get a new feasible solution, $X_2^{feasible}$. However, we experimentally found that the new feasible solution $X_2^{feasible}$ was always identical to $X_1^{feasible}$. This result is not too surprising since multiplying X_1^* by α only scales the values of X_{ij} and will not change their sorted order. In addition, since we got $X_1^{feasible}$ from X_1^*, they most likely have vertices i, j, and h with the same sorted order. Once we multiply $X_1^{feasible}$ by $(1-\alpha)$ then this will only scale the values but will not change their sorted order. We have $X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}$ so adding the edges values, X_{ij}, of the three vertices using the matrix X_2 will give the same result as when we add the edges of the three vertices using the matrix X_1 since the order of T_{ijh} values in the sorting will likely remain the same (with a difference in the value since it is scaled and shifted). This was the case in all our computational experiments.

Hence, the convex combination technique does not seem to improve the solution for the ICH heuristic. This gives evidence that the heuristic is strong enough that it does not benefit from performing the convex combination improvement technique.

A computational comparison of the ICH heuristic and the hyperplane rounding technique is presented in Section 4.1.

3.3 Branching Rules

Part of the success of a branch-and-bound algorithm depends on the choice of the variable to branch on. Based on the results of the analysis done by Helmberg and Rendl in [17], we decided to use in our branch-and-cut implementation a version of their branching rule R3 which branches on the variable that is 'least decided' in the optimal solution of the SDP relaxation of the current node. Our branching rule works as follows:

Select the edge ij with X_{ij}^* farthest from 1 and $\frac{-1}{k-1}$, i.e., branch on the edge ij that minimizes $|\frac{2X_{ij}^*(k-1)-k+2}{k}|$.

By branching on the most difficult decision X_{ij}, we hope that the bound will improve fast.

3.4 The SBC Algorithm

We implemented the algorithms and the methods that we described in the preceding sections into a branch-and-cut algorithm. A description of it is provided in Algorithm 3.

4 Computational Results

4.1 Comparison of Hyperplane Rounding and the ICH Heuristic

We implemented ICH and the hyperplane rounding presented in [14] using C and MATLAB respectively. In this section, we compare the two algorithms to find a feasible solution for the MkP problem.

Algorithm 3 SBC Algorithm

Step 1: Initialization Form the root node by using the (SMKP) problem without fixing any variables.

Step 2: Terminating If all nodes are fathomed then terminate with the incumbent solution, $X_{incumbent}$, as the optimal solution and the corresponding objective value ν^* as the optimal objective value.

Step 3: Solving Choose a node t not yet solved. Solve the SDP relaxation of the current subproblem to get a solution X_t^* and a lower bound ω_t.

Step 4: Adding Valid Inequalities Separate violated triangle and clique inequalities as discussed in Section 3.1. If none are violated go to Step 5. Otherwise, go to Step 3.

Step 5: Obtaining a Feasible Solution Get a feasible solution $X_{feasible_t}$ using ICH heuristic as discussed in Algorithm 2 and an upper bound ν_t as the objective value of $X_{feasible_t}$. Try to improve ν_t locally by local exchange routines. Update the incumbent if $\nu_t < \nu^*$.

Step 6: Fathoming

1. By Solving: If the solution X_t^* has all entries $\frac{-1}{k-1}$ or 1, i.e., ω_t and ν_t are identical. Go to Step 2.
2. By Bound: If the SDP relaxation gives $\omega_t \geq \nu^*$, then branching on this node will not improve the incumbent. Go to Step 2.
3. By Infeasibility: If the SDP relaxation doesn't have a feasible solution. Go to Step 2.

Step 7: Branching Choose a variable that is non-feasible (i.e., not 1 or $\frac{-1}{k-1}$) and create two new nodes by fixing the variable to 1 for one node and $\frac{-1}{k-1}$ for the other node. Go to Step 2.

Since the hyperplane rounding presented by [14] is randomized, each time we run the algorithm a different feasible solution might be obtained. As a result, this algorithm was run 30 times and the minimum and the average of the upper bound (UB) were computed. The average value can be interpreted as an estimate of the expected value of the UB that this algorithm would give. On the other hand, the minimum value is the best solution found over the 30 runs. This minimum value is the value reported in Tables 1-3. More detailed results are presented in [15].

In addition to randomly generated edge weights of complete graphs, we consider a set of test problems arising in a physics application e.g., [18] provides some recent physics analysis and introduces the physics literature. The two techniques were tested on the following three types of graphs for $k = 2$ and for $k = 3$:

- Random Instances: These instances consist of complete graphs where the edge weights are randomly generated between 0 and 9.
- Spinglass2g Instances: These instances consist of graphs that were generated using the rudy graph generator [24]. Spinglass2g generates a toroidal two dimensional grid with Gaussian distributed weights.
- Grid_2D Instances: These instances consist of graphs that were generated using the rudy graph generator [24]. Grid_2D generates a planar bidimensional grid with edge weights all equal to 1.

	$\|V\|$	SDP LB	ICH	Frieze & Jerrum	α for Frieze & Jerrum with convex combination								
					0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
k=2	30	912	**912**	**912**	**912**	**912**	**912**	921	927	941	925	**912**	**912**
	40	1690.463	**1691**	1728	1728	1742	1773	**1691**	**1691**	1716	1719	**1691**	**1691**
	50	2716.069	2729	2858	2848	2767	2823	2857	2815	2787	2810	2802	2855
	60	3985.364	4001	4151	4054	4151	4128	4106	4172	4196	4112	4095	4159
	70	5384.104	5401	5608	5667	5625	5452	5581	5654	5520	5501	5584	5557
	80	7032.764	7098	7389	7389	7382	7211	7321	7363	7381	7281	7341	7230
	90	9190.776	9292	9830.2	9551	9572	9595	9480	9599	9508	9450	9592	9568
	100	11382.92	11496	11747	11784	11747	11881	11854	11878	11871	11878	11936	11860
k=3	30	493.7	557	589	614	588	611	623	598	605	580	592	558
	40	925.5	992	1088	1117	1135	1108	1094	1130	1077	1101	1096	1089
	50	1497.1	1656	1694	1752	1735	1725	1704	1766	1761	1757	1706	1737
	60	2351.9	2548	2724	2749	2809	2739	2774	2716	2722	2649	2692	2705
	70	3223.4	3477	3679	3815	3708	3774	3706	3789	3676	3685	3615	3664
	80	4293.5	4508	4848	4892	4888	4790	4809	4909	4857	4877	4892	4815
	90	5420.1	5774	6132	6249	6134	6084	6054	6263	6117	6151	6139	6098
	100	6634.2	6973	7491	7566	7661	7529	7534	7602	7561	7549	7496	7433

Table 1: Computational Results for random instances with $k = 2$ and 3. Numbers in bold indicate that the heuristic solution is the optimal solution.

	$\|V\|$	SDP LB	ICH	Frieze & Jerrum	α for Frieze & Jerrum with convex combination								
					0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$k = 2$	3 × 3	-795504	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**
	4 × 4	-592434	-592202	-592202	**-592202**	**-592202**	**-592202**	**-592202**	**-592202**	**-592202**	**-592202**	**-592202**	-536551
	5 × 5	-1543707	**-1543707**	-1451470	**-1543707**	-1182988	**-1543707**	**-1543707**	-1461391	**-1543707**	-1278662	**-1543707**	**-1543707**
	6 × 6	-2846691	**-2846691**	**-2846691**	**-2846691**	-2554116	**-2846691**	**-2846691**	-2680268	**-2846691**	**-2846691**	-2718202	**-2846691**
	7 × 7	-3020297	**-3020297**	-2818053	-2787849	-2787849	-2818053	**-3020297**	**-3020297**	**-3020297**	-2807370	**-3020297**	**-3020297**
	8 × 8	-4489989	**-4489989**	**-4489989**	-4396002	**-4489989**	-4252115	**-4489989**	-4271661	-3702040	**-4489989**	**-4489989**	-4005560
	9 × 9	-6230102	**-6230102**	-5522456	**-6230102**	-5950570	-5858896	-5522456	-5644327	-5953242	-5213005	**-6230102**	-6153144
	10 × 10	-7872968	**-7872968**	**-7872968**	**-7872968**	-7760364	**-7872968**	-6869239	**-7872968**	-6480148	-7042388	**-7872968**	-7540716
$k = 3$	4 × 4	-954108	-954077	-954077	-819312	-831392	-741231	-798278	-761526	-580161	-741298	-751103	-852946
	5 × 5	-1484348	-1367840	-1185097	-1484348	-1166946	-1103329	-1124676	-1188754	-836275	-1319361	-1175218	-966957
	6 × 6	-2758520	**-2758520**	-2147425	-2115524	-1732624	-1802507	-1625819	-2758520	-1414849	-1872757	-1595561	-2690359
	7 × 7	-3282586	**-3282586**	-2115560	-2115560	-2115560	-2889403	-1587528	-1902404	-1841520	-2171880	-2016232	-2756529
	8 × 8	-4063059	**-4063059**	-2705506	-2469005	-2090016	-2128793	-2219785	-2419073	-2523733	-3154943	-2896465	-2502696
	9 × 9	-5236178	-4758332	-2247374	-2225260	-2324296	-1970217	-2127385	-2235664	-2026423	-2085498	-2307414	-3155256
	10 × 10	-7230203	-6570984	-3150645	-3251798	-3442696	-2750327	-3124199	-3122204	-3638336	-3395681	-2941674	-3579241

Table 2: Computational results for spinglass2g with $k = 2$ and 3. Numbers in bold indicate that the heuristic solution is the optimal solution.

				Frieze &	α for Frieze & Jerrum with convex combination								
	$\|V\|$	SDP LB	ICH	Jerrum	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$k=2$	3×3	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	4×4	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	3	**0**
	5×5	0	**0**	**0**	2	**0**	2	**0**	**0**	**0**	6	6	**0**
	6×6	0	**0**	**0**	**0**	**0**	**0**	**0**	3	**0**	2	**0**	2
	7×7	0	**0**	4	4	4	3	3	**0**	**0**	**0**	**0**	3
	8×8	0	**0**	**0**	**0**	4	20	**0**	4	**0**	**0**	13	**0**
	9×9	0	**0**	**0**	3	**0**	10	2	**0**	8	**0**	**0**	12
	10×10	0	**0**	**0**	**0**	**0**	10	**0**	3	7	**0**	4	**0**
$k=3$	3×3	0	**0**	1	2	**0**	1	**0**	**0**	1	**0**	1	**0**
	4×4	0	**0**	2	3	3	1	3	2	1	4	2	3
	5×5	0	**0**	4	7	7	4	4	1	5	6	4	4
	6×6	0	**0**	7	12	8	12	8	12	7	8	7	9
	7×7	0	**0**	14	15	16	18	14	13	17	13	13	13
	8×8	0	**0**	17	16	20	20	24	20	17	19	17	20
	9×9	0	**0**	22	32	29	23	24	25	25	24	26	21
	10×10	0	**0**	36	39	34	33	39	37	35	39	35	33

Table 3: Computational Results for grid_2D with $k = 2$ and 3. Numbers in bold indicate that the heuristic solution is the optimal solution.

From Tables 1-3, we notice that ICH is in most cases at least as good as the hyperplane rounding minimum. Moreover, even using different values of α for the hyperplane rounding with convex combination, the results are still not as good as those of the ICH heuristic. Therefore, the UBs provided by ICH are generally tighter and using it at each node of the branch-and-cut algorithm helps reducing the size of the tree.

From Table 2 we see that for spinglass2g instances where positive and negative edge weights are present, the ICH heuristic provides a better solution than the minimum value of hyperplane rounding for all values of α. This shows that ICH is still very effective in the presence of negative weights unlike the hyperplane rounding (we note that the performance guarantee from Theorem 1 does not apply for these instances). Moreover, by comparing the UB provided by ICH with the LB, we see that the ICH heuristic provides a tight bound at the root node and sometimes immediately finds the optimal solution.

We note that for the grid_2D instances we can find a solution by inspection. For the case $k = 2$ there is a unique solution, while for $k = 3$ we have multiple solutions. Moreover, for $k = 2$ the SDP matrix X^* satisfies $X_{ij} \in \{-1, 1\}$ while for $k = 3$ the SDP matrix X^* doesn't have its entries $X_{ij} \in \{-\frac{1}{2}, 1\}$ but the matrix is in practice often a convex combination of several multiple solutions. We included the results for grid_2D instances to show that even if we don't have the SDP matrix with $X_{ij} \in \{-\frac{1}{k-1}, 1\}$ entries, the ICH heuristic can still extract a feasible solution that was found to be optimal for all test cases tried, unlike the hyperplane rounding.

4.2 Computational Results for SBC Algorithm

We implemented in C the branch-and-cut SDP-based Algorithm (SBC) described in Section 3.4. To solve the SDP, which has to be done at each node of the tree, we used the CSDP solver [5]. The computations were done on a a 1200 MHz Sun Sparc machine.

$\|V\|$	Optimal Solution	Time	Number of Nodes
20	147	0:00:06	1
30	495	0:00:09	1
40	1183	0:02:10	1
50	2312	0:14:20	1
60	3990	0:06:41	1
70	6348	0:58:29	1

Table 4: SBC results for clique instances where $k = 3$. The time is given in hr:min:sec. The last column is the number of nodes required to reach the optimal solution

4.2.1 Test Instances

The test instances consist of graphs generated using the graph generator rudy of Rinaldi [24]. The instances consist of complete graphs and two and three-dimensional grid graphs with Gaussian distributed and ± 1 edge weights:

- Clique: generates complete graphs with edge weight of edge (i, j) chosen as $|i - j|$.
- Spinglass2g: described in Section 4.1.
- Spinglass3g: generates a toroidal three-dimensional grid with Gaussian distributed weights. The grid has size $n =$(rows×columns×layers).
- Spinglass2pm: generates a toroidal two-dimensional grid with ± 1 weights. The grid has size $n =$(rows×columns). The percentage of negative weights is 50%.
- Spinglass3pm: generates a toroidal three dimensional grid with ± 1 weights. The grid has size $n =$(rows×columns×layers). The percentage of negative weights is 50%.

Table 4 shows the computational result for clique instances. Tables 5-6 show the computational results for the SBC algorithm for two-dimensional and three-dimensional grid instances with $k = 3$. In addition to the optimum solution value, the lower bound and the upper bound at the root node as well as the time at the root node are presented. Moreover, the number of nodes of the branch-and-bound tree as well as the time to reach a certain percentage gap are given in the tables. The symbol $\curvearrowleft$ denotes that a gap smaller than the one written in the corresponding column was achieved at the root node. For Table 5, we give optimum solutions for sizes up to 100 vertices (10×10 grids) and provide a feasible solution for larger sizes (up to 169 vertices) with a percentage gap of less than 6%.

$\|V\|$	Best Solution Value	Root Node			Number of Nodes - Time to achieve % Gap				
		LB	UB	Time	0%	1%	2%	5%	10%
3×3	-449795	-449795	-449795	0:00:05	1 - 0:00:5	↶	↶	↶	↶
4×4	-954077	-954077	-954077	0:00:16	1 - 0:00:16	↶	↶	↶	↶
5×5	-1484348	-1484722	-1484348	0:00:18	2 - 0:00:23	1 - 0:00:18	↶	↶	↶
6×6	-2865560	-2865560	-2865560	0:05:12	1 - 0:05:12	↶	↶	↶	↶
7×7	-3282435	-3282435	-3282435	0:52:08	1 - 0:52:08	↶	↶	↶	↶
8×8	-5935341	-5935341	-5935341	2:21:43	1 - 2:21:43	↶	↶	↶	↶
9×9	-4758332	-4806178	-4758332	3:35:49	4 - 13:41:17	1 - 3:35:49	↶	↶	↶
10×10	-6570984	-6630202.5	-6570984	10:36:23	6 - 18:09:41	1 - 10:36:23	↶	↶	↶
11×11	-8586382	-9015701.1	-8586382	5:48:50	-	-	-	1 - 5:48:50	↶
12×12	-10646782	-11189768	-10646782	9:31:00	-	-	-	1 - 9:31:00	↶
13×13	-11618406	-12292274	-11618406	29:33:27	-	-	-	-	1 - 29:33:27
14×14	-13780370	-14607192	-13780370	47:16:57	-	-	-	-	1 - 47:16:57
$2 \times 3 \times 4$	-2197030	-2197030	-2197030	0:01:14	1 - 0:01:14	↶	↶	↶	↶
$2 \times 3 \times 5$	-2026448	-2026448	-2026448	0:08:02	1 - 0:08:02	↶	↶	↶	↶
$2 \times 4 \times 5$	-3392938	-3392938	-3392938	0:36:18	1 - 0:36:18	↶	↶	↶	↶
$3 \times 3 \times 3$	-1882389	-1882389	-1882389	0:00:21	1 - 0:00:21	↶	↶	↶	↶
$3 \times 3 \times 4$	-3192317	-3192317	-3192317	0:26:52	1 - 0:26:52	↶	↶	↶	↶
$3 \times 3 \times 5$	-4204246	-4209348	-4204246	2:52:31	5 - 3:38:37	1 - 2:52:31	↶	↶	↶
$3 \times 4 \times 4$	-5387838	-5421403	-5387838	0:58:15	3 - 1:38:51	1 - 0:58:15	↶	↶	↶
$3 \times 4 \times 5$	-5240435	-5323788	-5049424	6:02:52	13 - 19:12:31	10 - 16:43:10	7 - 11:21:53	1 - 6:02:52	↶
$4 \times 4 \times 4$	-7474525	-7529318	-7474525	3:22:37	3 - 10:12:11	1 - 3:22:37	↶	↶	↶

Table 5: SBC results for spinglass2g and spinglass3g instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

$\|V\|$	Best Solution Value	Root Node			Number of Nodes - Time to achieve % Gap				
		LB	UB	Time	0%	1%	2%	5%	10%
4×4	-13	-13	-13	0:00:00	1 - 0:00:00	↶	↶	↶	↶
5×5	-20	-20	-20	0:00:04	1 - 0:00:04	↶	↶	↶	↶
6×6	-29	-29	-29	0:00:22	1 - 0:00:22	↶	↶	↶	↶
7×7	-40	-40	-40	0:01:52	1 - 0:01:52	↶	↶	↶	↶
8×8	-55	-55	-55	0:26:38	1 - 0:26:38	↶	↶	↶	↶
9×9	-65	-65	-65	7:35:49	1 - 7:35:49	↶	↶	↶	↶
$2 \times 3 \times 4$	-20	-20	-20	0:00:03	1 - 0:00:03	↶	↶	↶	↶
$2 \times 4 \times 4$	-28	-28	-27	0:03:54	4 - 0:01:02	↶	↶	1 - 0:20:14	↶
$3 \times 3 \times 3$	-26	-26	-26	0:00:11	1 - 0:00:11	↶	↶	↶	↶
$3 \times 3 \times 4$	-36	-36	-36	0:00:50	1 - 0:00:50	↶	↶	↶	↶
$3 \times 4 \times 4$	-48	-48	-48	0:11:59	1 - 0:11:59	↶	↶	1 - 2:40:22	↶
$3 \times 4 \times 5$	-65	-66	-62	4:38:12	16 - 8:55:33	10 - 7:09:22	7 - 6:04:19	1 - 4:38:12	↶
$4 \times 4 \times 4$	-65	-65	-64	4:32:18	19 - 8:36:15	12 - 7:38:33	1 - 4:32:18	↶	↶

Table 6: SBC results for spinglass2pm and spinglass3pm instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

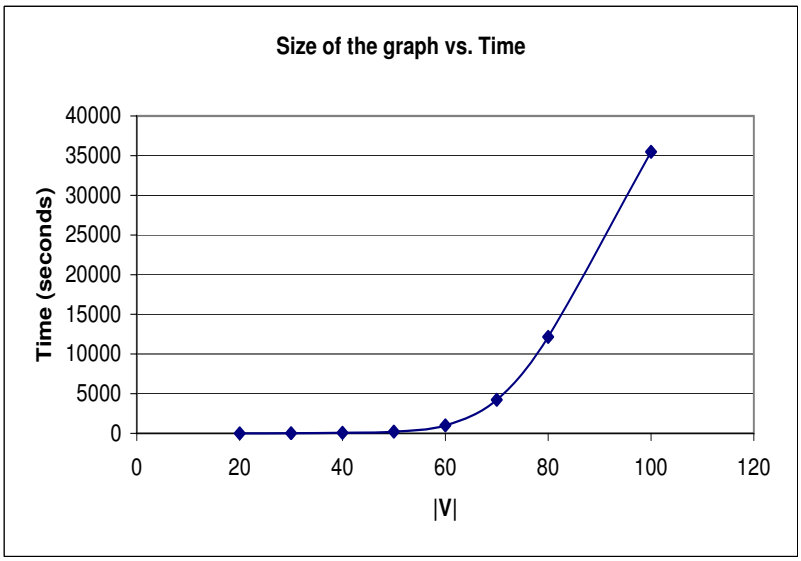

Figure 3: Size of Random instances versus computational time for $k = 3$. The average is always taken over five instances of the same size.

		$k = 5$		$k = 7$	
	$\|V\|$	Objective Value	Time	Objective Value	Time
spinglass2g	6×6	-2865560	0:23:41	-2865560	0:21:00
	7×7	-3843979	0:42:31	-3864156	0:39:23
	8×8	-5935341	2:09:07	-5935341	2:13:05
	9×9	-5745419	2:39:38	-6026024	2:18:56
	10×10	-6860706	19:14:02	-7644016	17:32:29
spinglass3g	$2 \times 3 \times 4$	-2212707	0:00:10	-2212707	0:00:08
	$2 \times 3 \times 5$	-2081357	0:08:07	-2081358	0:05:35
	$2 \times 4 \times 5$	-3578762	0:17:00	-3578762	0:13:01
	$3 \times 3 \times 3$	-2932403	0:00:47	-2932403	0:00:03
	$3 \times 3 \times 4$	-3552295	0:26:58	-3559337	0:21:15
	$3 \times 3 \times 5$	-4561622	2:04:49	-4648539	1:02:09
	$3 \times 4 \times 4$	-5371414	1:14:11	-5466518	1:18:02
	$4 \times 4 \times 4$	-7619675	9:30:19	-7646881	4:57:05

Table 7: SBC results for $k = 5$ and 7. The time is given in hr:min:sec.

4.2.2 Comparison and Analysis

The computational results which we have presented for spin-glass problems lead to the following observations:

1. SBC is able to determine optimum solutions for problems with Gaussian distributed and ± 1 edge weights for two- and three-dimensional grids with up to $n = 60$ vertices, and for k=3 within reasonable time. For $60 < n \leq 100$ we reach a gap of 1% within a reasonable amount of time, however reaching a 0% gap takes longer.
2. The remarkable tightness of the bounds obtained at the root node make it worthwhile to conduct a branch-and-cut algorithm since the bounds will likely help reduce the number of nodes in the tree.
3. Furthermore, the ICH heuristic often provides an optimal solution at the root node or after only a few branches. Most of the times computing the lower bound is the bottleneck; often ICH obtains the global optimal solution, but we cannot prove optimality right away.
4. For $k = 5$ and 7, our empirical analysis shows that for a given $|V|$ as k increases, the computational time decreases. Moreover, for some test cases the objective function values of the same test instance with different k values are the same, see Table 7. This is because the solution partitioned the vertices into less than k partitions due to the presence of positive and negative edge weights.

5 Conclusions and Future Work

In this paper we presented an exact algorithm for computing minimum k-partitions. Inside a branch-and-cut algorithm we used positive semidefinite relaxations that were further tightened using polyhedral results. The resulting algorithm is called SBC. The SBC algorithm was implemented and tested using several instances, and our computational results show the potential of SBC in tackling the MkP problem. We developed and implemented the novel ICH heuristic which appears to be a promising method for generating a good feasible solution. The proposed model often improves the upper bound and gives good feasible solutions. ICH can be applied to the MkP problem for different values of k. When compared with other approaches in the literature such as the hyperplane rounding technique by Frieze and Jerrum [14], it provides a better solution in practice. Moreover, the ICH heuristic was used in a SDP-based branch-and-cut approach to provide optimal solution for MkP.

Future research will investigate the solver used to solve the SDP at each node of the tree since it is the major bottleneck in the SBC algorithm. In particular, exploiting the structure of the graph and its sparsity may lead to an effective way for solving the SDP relaxations. Future work also includes adjusting the SBC algorithm and the ICH heuristic so that they can be applied to closely related partitioning problems such as the k-way equipartition problem.

References

[1] *Biq Mac solver. http://biqmac.uni-klu.ac.at/.*

[2] *Spin-glass server. http://www.informatik.uni-koeln.de/ls_juenger/research/sgs/index.html.*

[3] 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.

[4] F. Barahona and A. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[5] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11/12(1-4):613–623, 1999.

[6] 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.

[7] S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59:87–115, 1993.

[8] S. Chopra and M. R. Rao. Facets of the k-partition problem. *Discrete Applied Mathematics*, 61:27–48, 1995.

[9] 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.

[10] M. Deza and M. Laurent. *Geometry of Cuts and Metrics*. Algorithms and Combinatorics, Springer Verlag, 1997.

[11] J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):189–201, 2002.

[12] 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.

[13] M. Elf, M. Jünger, and G. Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31(5):343–349, 2003.

[14] A. Frieze and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18:67–81, 1997.

[15] B. Ghaddar. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. Master's thesis, University of Waterloo, 2007.

[16] 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.

[17] 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.

[18] L. W. Lee, H. G. Katzgraber, and A. P. Young. Critical behavior of the three- and ten-state short-range Potts glass: A Monte Carlo study. *Physical Review B*, 74:104–116, 2006.

[19] 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.

[20] A. Lisser and F. Rendl. Telecommunication clustering using linear and semidefinite programming. *Mathematical Programming*, 95:91–101, 2003.

[21] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions Information Theory*, IT-25:1–7, 1979.

[22] J. Mitchell. Branch-and-cut for the k-way equipartition problem. Technical report, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 2001.

[23] J. E. Mitchell. Realignment in the National Football League: Did they do it right? *Naval Research Logistics*, 50(7):683–701, 2003.

[24] G. Rinaldi. *Rudy. http://www-user.tu-chemnitz.de/∼helmberg/rudy.tar.gz*.

[25] A. Wiegele. *Nonlinear Optimization Techniques Applied to Combinatorial Optimization Problems*. PhD thesis, Alpen-Adria-Universität Klagenfurt, 2006.