Improving the linear relaxation of maximum k-cut with semidefinite-based constraints[*]

Vilmar Jefté Rodrigues de Sousa[†] Miguel F. Anjos[‡]
Sébastien Le Digabel[§]

April 4, 2018

Abstract. We consider the maximum k-cut problem that involves partitioning the vertex set of a graph into k subsets such that the sum of the weights of the edges joining vertices in different subsets is maximized. The associated semidefinite programming (SDP) relaxation is known to provide strong bounds, but it has a high computational cost. We use a cutting plane algorithm that relies on the early termination of an interior point method, and we study the performance of SDP and linear programming (LP) relaxations for various values of k and instance types. The LP relaxation is strengthened using combinatorial facet-defining inequalities and SDP-based constraints. Our computational results suggest that the LP approach, especially with the addition of SDP-based constraints, outperforms the SDP relaxations for graphs with positive-weight edges and $k \geq 7$.

Keywords. Maximum k-cut, graph partitioning, semidefinite programming, eigenvalue constraint, semi-infinite formulation.

AMS subject classifications. 65K05, 90C22, 90C35

1 Introduction

This work focuses on the graph partitioning problem known as the maximum k-cut (max-k-cut). We consider an undirected graph $G = (V, E)$ with edge weights w_{ij} for all $(i, j) \in E$. The task is to partition the vertex set V into at most k subsets (called clusters or colors) such that the sum of the edges with end points in different partitions is maximized.

---

[*] GERAD and Département de Mathématiques et Génie Industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montreal, QC, Canada H3C 3A7.

[†] Vilmar.de.sousa@gerad.ca

[‡] www.miguelanjos.com

[§] www.gerad.ca/Sebastien.Le.Digabel

The max-k-cut problem is equivalent to the minimum k-partition problem [14, 43], and the special case $k = 2$ that is known as the max-cut problem has attracted considerable attention; see, e.g., [4, 15, 26, 37, 40].

Many industrial applications can be formulated as the max-k-cut problem, including VLSI layout design [4], statistical physics [27], and wireless communication problems [12, 36].

The general max-k-cut is known to be $\mathcal{NP}$-complete [38]. Nonetheless, many relaxations [6, 39], heuristics [29], approximations [13, 23], and exact methods [2, 11, 31] have been proposed, some of which we study below.

We carry out a computational study to identify the relevance of an inequality based on semidefinite programming (SDP) and to determine the strongest formulation for each type of instance. To the best of our knowledge, no research to date has specifically studied SDP-based inequalities for the linear relaxation of the max-k-cut.

This paper is organized as follows. Section 1.1 reviews the SDP and linear programming (LP) formulations of the max-k-cut problem. Section 2 presents the SDP-based inequalities. Section 3 describes in detail the cutting plane algorithm (CPA) used to solve the relaxations, and Section 4 discusses the test results. Finally, some concluding remarks are made in Section 5.

1.1 Formulations

This section presents a literature review of the two formulations of the max-k-cut problem studied in this work.

1.1.1 Semidefinite programming formulation

The vertex formulation of the max-k-cut leads to an SDP relaxation. In the approximation method of [13] the authors define the SDP variable $X = (X_{ij})$, $i, j \in V$, where $X_{ij} = \frac{-1}{k-1}$ if vertices i and j are in different partitions of the k-cut of G and $X_{ij} = 1$ otherwise. The SDP formulation of the max-k-cut problem, *MkC-SDP*, can then be expressed as:

$$
\begin{array}{llllr}
(\textit{MkC-SDP}) & \max\limits_{X} & \dfrac{(k-1)}{k} \sum\limits_{i,j \in V}^{i<j} w_{ij}(1 - X_{ij}) & & (1) \\
& \text{s.t.} & X_{ii} = 1 & \forall i \in V & (2) \\
& & X_{ij} \geq \dfrac{-1}{k-1} & \forall i, j \in V, i < j & (3) \\
& & X \succeq 0 & & (4)
\end{array}
$$

Note that the constraints $X_{ij} \leq 1$ for $i, j \in V$ are removed from this relaxation since they are enforced implicitly by the constraints $X_{ii} = 1$ and $X \succeq 0$.

Because of the strength of the SDP, many researchers have used this formulation to design approximations [7, 13] and exact methods [2, 14]. In particular, [13] extends the max-cut approximation of [15] to the max-k-cut. In [2] the bundleBC algorithm is proposed to solve max-k-cut problems with 60 vertices by combining the SDP branch-and-cut method of [14] with the principles of the `Biq Mac` algorithm [40]. In [2] the authors show that their method achieves a dramatic speedup in comparison to [14], especially when $k = 3$.

1.1.2 Linear formulation

Chopra & Rao [6] presented an edge-only 0-1 formulation of max-k-cut. For each $e \in E$, the variable x takes the value 0 when edge e is cut, and 1 otherwise. Hence, the edge-only linear relaxation of max-k-cut can be formulated as:

$$\begin{aligned}
(\textit{MkC-LP}) \qquad & \max_{x} \quad \sum_{i,j \in V}^{i<j} w_{ij}(1 - x_{ij}) && && (5)\\
& \text{s.t.} \quad x_{ih} + x_{hj} - x_{ij} \leq 1 && \forall i, j, h \in V && (6)\\
& \qquad \sum_{i,j \in Q, i<j} x_{ij} \geq 1 && \forall Q \subseteq V \text{ with } |Q| = k + 1 && (7)\\
& \qquad 0 \leq x_{ij} \leq 1 && \forall i, j \in V && (8)
\end{aligned}$$

where Constraints (6) and (7) correspond to the triangle and clique inequalities, respectively. These families of inequalities imply that there are at most k partitions in the integer formulation.

The LP formulation of max-k-cut has been extensively studied; see, e.g., [5, 6, 31]. In [5, 6] the authors give several valid inequalities and facet-defining inequalities for *MkC-LP* and for "node-and-edge" formulations, i.e., linear formulations with both node and edge variables. In [11], via projection of the edge-only formulation, the authors obtain new families of valid inequalities, along with new separation algorithms for the node-and-edge formulation. Their results show that these new inequalities are practical for large sparse graphs.

Two drawbacks of the *MkC-LP* formulation are mentioned in [12]. First, it cannot exploit structure of G, such as sparsity. Second, it has $\mathcal{O}(|E|)$ variables and $\mathcal{O}(|V|^{k+1})$ constraints. These disadvantages can be reduced by simplifying the input graph G. In this work, we exploit sparsity via a k-core reduction, a block decomposition [12, 22, 42], and a chordal extension [19, 43]. The second disadvantage is mitigated by a CPA (Section 1) that overcomes the huge number of inequalities by activating only important constraints in the relaxation.

Sparsity can also be exploited by node-and-edge formulations [1, 6, 12]. In [1] the authors used representative variables to break symmetry. They show that the relevance of their formulation increases with the number of partitions, but our preliminary tests show that node-and-edge formulations are expensive and impractical for large graphs.

1.1.3 SDP versus LP

Several researchers have compared the semidefinite relaxation with the linear relaxation for partitioning problems. In the branch-and-cut method for the minimum k-partition problem [14], the authors claim that linear bounds are weak and that this could result in the enumeration of all the solutions in a branch-and-bound method.

The relation between the LP and SDP polytopes is studied in [10], where the authors show that the strength of the SDP bounds is related to the fact that "hypermetric inequalities" are implicit in the *MkC-SDP*. For example, they show that all triangle constraints are violated by at most $\sqrt{2} - 1$ and all clique constraints by less than $1/2$ in the SDP relaxation, in comparison with a violation of 1 for the LP relaxation.

Moreover, in [2] the authors claim that high computational times are the price to pay for the strength of SDP relaxations.

The linear and semidefinite relaxations of the graph partitioning problem where each cluster must have about the same cardinality (also known as the k-equipartition problem) are considered in [28]. The mathematical and experimental results indicate that the linear relaxation is stronger than the SDP relaxation for large values of k when a bound separation is used (see Section 3.1.2). However, for small values of k, the latter outperforms the former.

2 SDP-based inequality

Since SDP relaxations are expensive but often yield stronger bounds than linear relaxations, we use semi-infinite programming (SIP) to formulate an SDP-based inequality for the *MkC-LP*. In this section we briefly review SIP and then introduce the variable transformation that allows us to map the infinite SDP constraint to LP. Finally, we present the SDP-based inequality.

2.1 Semi-infinite formulation of SDP

The SIP can be defined as an optimization problem with finitely many variables and infinitely many constraints. The survey [21] discusses the theory, algorithms, and applications of semi-infinite programming. In [24] the authors study linear semi-infinite programming (LSIP) for generic SDPs.

We note that the convex constraint $X \succeq 0$ (4) is equivalent to

$$\mu^T X \mu = \mu\mu^T \bullet X \geq 0 \qquad \forall \mu \in \mathbb{R}^n \tag{9}$$

where $n = |V|$ and $\mathbb{R}^n$ can be considered as a compact set, where typically the Euclidean norm of μ is one. Theorem 1.1.8 of [20] proves this equivalent characterization of positive semidefinite matrices. Moreover, [20] provides more fundamental results from linear algebra and the properties of the cone of symmetric semidefinite matrices.

The matrix constraint (9) has an infinite number of rows. By replacing (4) by (9) in *MkC-SDP* we obtain the LSIP formulation of SDP. In the cut-and-price approach proposed in [25] the authors use the LSIP of the dual SDP formulation for the *max-cut* problem. Their results suggest that the linear approach is able to solve large-scale problems.

2.2 Variable transformations

To incorporate Constraint (9) in our linear formulation we need to transform the semidefinite variable $X \in \left[\frac{-1}{k-1}, 1\right]$ to the related $x \in [0,1]$ linear formulation. Using the identities $x_{ij} = \dfrac{k-1}{k} X_{ij} + \dfrac{1}{k}$ and $X_{ij} = \dfrac{k}{k-1} x_{ij} - \dfrac{1}{k-1}$ for all $i, j \in V$ we can map valid inequalities for the LP to the SDP and vice versa.

2.3 SDP-based inequality formulation

By applying the transformation proposed in Section 2.2 to Constraint (9) we derive the following class of inequalities for *MkC-LP*:

$$\sum_{i,j \in V}^{i<j} \mu_{ij} x_{ij} \geq \frac{1}{k} \sum_{i,j \in V}^{i<j} \mu_{ij} - \frac{k-1}{2k} \sum_{i \in V} \mu_{ii} \quad \forall \mu \in \mathbb{R}^n \tag{10}$$

In [24] the authors prove that these inequalities ensure that the set of linear solutions is feasible for the SDP. In Section 3.1.3 we propose an exact separation routine to deal with the infinite number of constraints.

3 Cutting-plane algorithm

A CPA is an iterative method used to obtain upper bounds on the optimal value of max-k-cut and to prove optimality. First, the CPA solves the relaxed problem (SDP or LP) to obtain an upper bound on the integer program, then it searches for violated inequalities and adds some of them to the relaxation. We first introduce the generic algorithm, then discuss methods for choosing the inequalities to add/remove, and finally present the method used to solve the relaxations.

We summarize the CPA in Figure 1. We say that an iteration is completed every time we enter Step 6, and we complete the CPA when we enter Step 4 for the last time. Note that other termination criteria can be used, e.g., number of iterations, computational time, and improvement at each iteration.

3.1 Separation routines

Separation routines are algorithms that search for violations of a given family of valid inequalities in a relaxed solution. In this section we present separation routines for some inequalities studied in [8], for Constraint (3) in the SDP formulation, and for Constraint (10) proposed in this work.

1. **Initialize.** Load the instance and set up the initial relaxation. Initialize the iterate i.
2. **Solve** the relaxation to optimality or with duality tolerance (ε_T) (Section 3.3).
3. **Search for violations.** Use the separation routine to find violated inequalities at the current solution (Section 3.1).
4. **Add inequalities.** If there are violated inequalities then add at most $NbIneq$ (see Section 3.1.4) of those that are most violated. Otherwise, if the relaxation was solved to optimality in Step 2 then **STOP** because the algorithm cannot improve the relaxation.
5. **Drop inequalities**. If any constraint is no longer important, remove it (Section 3.2).
6. **Modify current iterate.** Increment i. Reduce or increase ε_T, if necessary. Return to Step 2.

Figure 1: Cutting plane algorithm.

3.1.1 Separation of combinatorial inequalities

Some valid and facet-inducing inequalities have been proposed in [6] for the *MkC-LP*. Five of these families of constraints are explored computationally in [8], where heuristic and exact methods are proposed. In this work, we replicate the best separation routines:

- Triangle: complete enumeration.
- Clique: greedy heuristic.
- General clique: greedy heuristic.
- Wheel: greedy heuristic.
- Bicycle wheel: genetic algorithm.

In [8] the authors concluded that in practice, wheel and triangle are the best inequalities. Hence, we prioritize these two families of inequalities in our ranking algorithm (see Section 3.1.4).

3.1.2 Separation of bound inequalities

In [20] the author indicates that it is more efficient to start the CPA with only the diagonal Constraints (2) of the SDP formulation and to separate $X_{ij} \geq \frac{-1}{k-1}$ iteratively.

Figure 2 compares the performance of the SDP formulation with and without bound separation. It shows the percentage gap (see Section 4.1) versus the CPU time (s) for 10 random instances, with a density of 0.8 and dimension $|V| = 100$.

Figure 2 shows that the first (and only) SDP iteration without bound separation takes, on average, 350 s. In contrast, the CPA with bound separation achieves the same result

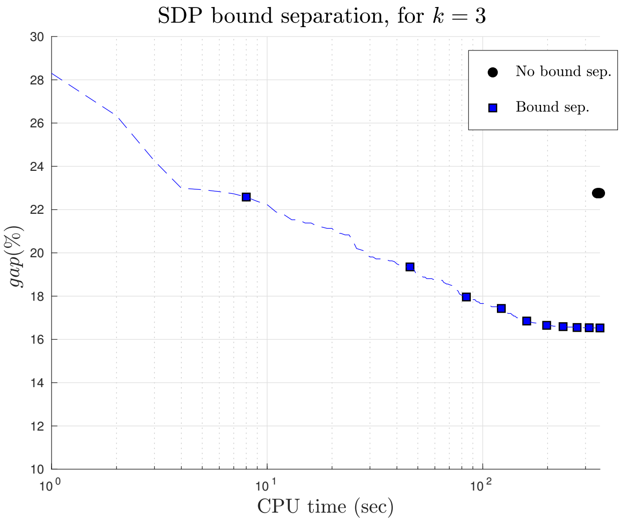

Figure 2: Separation of Constraint (3) in SDP formulation.

in less than 10 s.

For brevity, we show the profiles for $k = 3$ only. However, these results can be generalized to larger k and to sparse graphs. Moreover, our computational tests on instances with $|V| \geq 300$ show that when there is no bound separation the first iteration takes more than 1 h.

Therefore, we apply bound separation for Constraint (3). We perform a complete enumeration of all edges $e \in E$, and only the $NbIneq$ (see Section 3.1.4) most violated inequalities are added for the next CPA iteration.

3.1.3 Separation of SDP-based inequalities

The family of SDP-based inequalities (10) has infinite combinations, which makes it impractical to add them and to find the ones that are violated. Since the eigenvalues of a positive semidefinite matrix are non-negative [20], we can execute an exact separation routine in polynomial time to find the inequalities that are violated in a linear solution.

Let $\hat{x}$ be an optimal solution of *MkC-LP*. If the related symmetric matrix $\hat{X}$ is not semidefinite ($\hat{X} \nsucceq 0$) then it has at least one negative eigenvalue $\lambda < 0$, and the following inequalities are violated by $\hat{x}$:

$$\sum_{i,j \in V}^{i<j} v_{ij} x_{ij} \geq \frac{1}{k} \sum_{i,j \in V}^{i<j} v_{ij} - \frac{k-1}{2k} \sum_{i \in V} v_{ii} \qquad \forall \lambda < 0 \tag{11}$$

where the columns of $\mathbf{v}$ are the eigenvectors corresponding to the values λ. The addition of (11) to *MkC-LP* will cut off the LP solution and improve the iterate in a cutting plane scheme.

We use the term *LP-EIG* for the linear approach with eigenvalue separation. We use `Eigen` [18] to compute the eigenvalues and eigenvectors of $\hat{X}$. `Eigen` is a C++ template library for linear algebra, and it computes all the eigenvalues and eigenvectors for a self-adjoint matrix (real symmetric matrix) using a symmetric QR algorithm. The computational cost is approximately $\mathcal{O}(9n^3)$.

3.1.4 Maximum number of inequalities in CPA

As shown in [8], the inclusion of all the violated inequalities in a CPA iteration can be computationally impractical. It is better to rank the violated inequalities and append only those that are most violated. Empirical tests show that the maximum number of inequalities ($NbIneq$) should be set to $NbIneq = 2|V|$ for linear methods and $NbIneq = 400$ for the SDP formulations, similarly to [8].

3.2 Dropping inequalities

An inequality is said to be important when at optimality its slack variable (sk) is close to zero, i.e., the inequality is active. Removing unimportant constraints reduces the size of the relaxation and thus the computational time.

In [32] the authors observed that tests based on ellipsoids can determine when to drop a constraint, but the cost of these tests may exceed the computational savings. Therefore, we simply test whether a slack variable is larger than a fixed value ($\gamma = 0.001$), i.e. , we remove inequalities with $sk > 10^{-3}$.

Searching for unimportant inequalities at each CPA iteration takes time, and some constraints can be repeatedly added and removed. Our computational tests show that the best performance is obtained when the dropping is done at every third CPA iteration ($Ite_{drop} = 3$).

3.3 Solving the relaxations

One of the most important decisions in the CPA is the choice of the solution method for the relaxation. We solve the SDP and LP relaxations of the max-k-cut using the interior point method (IPM) of `MOSEK` [3]. Our computational tests indicated that the default IPM is not efficient so, inspired by the `PDCGM` solver [17], we considerably modified the IPM to improve the CPA performance. This section discusses the main changes; some of them are also applicable to other solvers.

In [16, 30] the authors claim that IPMs are an alternative to the simplex method for LP problems; they show that IPMs enable the solution of many large real-world problems. As observed in [32] IPMs can exploit parallelism.

We use the **early termination** of the IPM. We solve the relaxations approximately with a relative dual termination tolerance (ε_T). As shown in [35], non-extremal solutions may separate valid inequalities effectively, because the cuts may be deeper and usually fewer are needed. Inequalities generated by the early termination may provide deeper

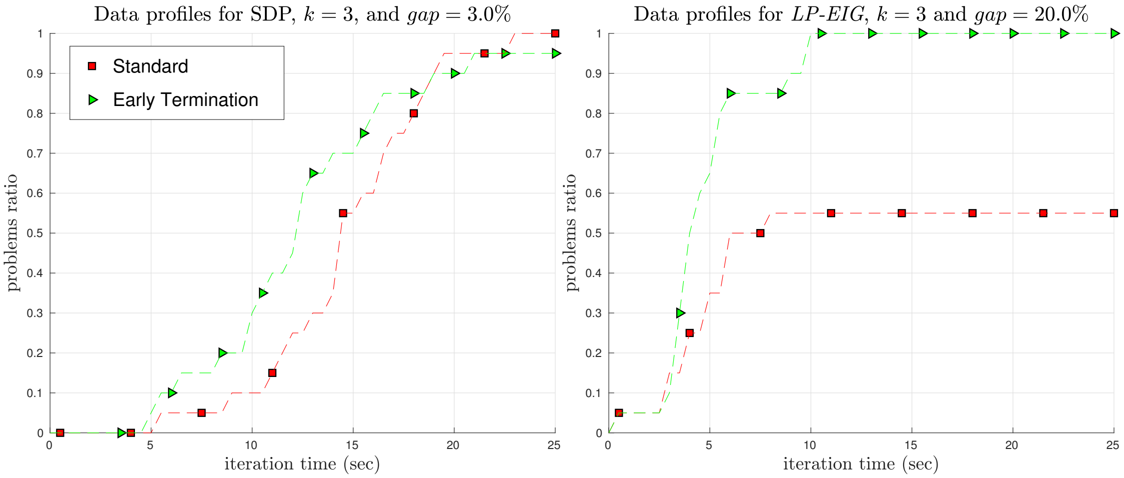

Figure 3: Study of early termination in IPM.

cuts because the iterate is further from the boundary of the polyhedron. Moreover, the early termination can save computational time by not executing all the IPM iterations.

In [30] the author gives the two principal drawbacks of separating valid inequalities before the current relaxation is solved to optimality. First, it may not be possible to find a constraint, so the time spent is wasted. Second, the separation routine may return inequalities that are violated by the current iterate but not by the optimal solution, so we may end up solving a relaxation with unimportant constraints.

To reduce the impact of the first disadvantage, we use a dynamic tolerance to decide when to stop the IPM, so we search for violated inequalities only when the duality gap is below a tolerance (ε_T). We increase ε_T by 25% if the number of violated constraints is greater than $2 \cdot NbIneq$ (see Section 3.1.4) and decrease ε_T if we have fewer than $NbIneq$ violated constraints. The best results were obtained when ε_T was initially set to 0.75.

The second disadvantage is mitigated by occasionally solving the relaxation to optimality. The best results were obtained when we set $Ite_{optLP} = 5$ for LP (i.e., we solved every fifth relaxation) and $Ite_{optSDP} = 2$ for SDP. When plotting the results we show only those obtained from relaxations solved to optimality.

Figure 3 plots the data profiles (see Section 4.3) of the early-termination and standard IPM for the SDP and *LP-EIG* relaxations; the CPU time is limited to 300 s. This figure gives the average results for 40 random dense (density=0.9) instances with $|V| = 100$ and $k = 3$, and the results can be generalized to other graphs. The gap is smaller for SDP than for *LP-EIG* because the latter formulations are unable to solve these problems with a gap below 10%. We conclude that SDP is stronger than *LP-EIG* for $k = 3$. However, in the next sections we show that this is not always the case: *LP-EIG* can be much stronger than SDP.

Figure 3 shows that early-termination outperforms the standard IPM, especially for the linear formulation of max-k-cut. For example, with a gap of 20% the early-termination solves all the *LP-EIG* problems in 10 s, whereas standard IPM solves just 55% of these problems. Therefore, we use the early-termination method in our computational tests in the next section.

4 Computational tests

We solve the SDP and LP relaxations of max-k-cut using the IPM of `MOSEK` [3] on a Linux PC with two Intel(R) Xeon(R) 3.07 GHz processors. We performed tests for $k \in \{3, 5, 10, 20\}$ on 228 test problems.

4.1 Terminology

In this section we present the terminology used for our analysis.

- **Best feasible solution** (LB_p): The value of the best known integer solution for problem p. If the optimal solution is unknown we calculate a feasible solution using the variable neighborhood search metaheuristic [33].
- **Final solution** ($UB_{p,m}$): The value of method m at the end of the CPA for problem p. It is also known as the upper bound for method m.
- **Performance ratio** ($gap_{p,m}$): The gap of method m is the difference between its upper bound and the best feasible solution. It is calculated as follows:

$$gap_{p,m} = \frac{UB_{p,m} - LB_p}{LB_p}. \tag{12}$$

- **Iteration time** ($itime_{p,m}$): The CPU time for one CPA iteration for method m and problem p. The time to solve the final iteration of a problem is t_{Last}.
- **Set of methods** ($\mathcal{M}$): The three methods listed below are relaxations of the max-k-cut problem, and all of them use CPA to improve their formulation with the separation of combinatorial inequalities (Section 3.1.1):
  - LP: Solves the LP formulation.
  - *LP-EIG*: Solves the LP formulation with the separation of SDP-based inequalities (Constraint (10)).
  - SDP: Solves the SDP formulation with the separation of bound inequalities (Section 3.1.2).

4.2 Instances

We consider 228 instances; 68 are from the `Biq Mac` library [44] and 160 were randomly generated using `rudy` [41].

- `Biq Mac` problems:
  - be: These are the Billionnet and Elloumi instances. For each density $d \in \{0.3, 0.8\}$ we use ten problems with edge weights chosen from $\{-50, 50\}$.

  - bqp: Ten weighted graphs with dimension 100, density 0.1, and edge weights chosen from $\{-100, 0, 100\}$.
  - g05: Ten unweighted graphs with edge probability 0.5 and dimension 100.
  - ising2: Six one-dimensional Ising chain instances for dimension $|V| \in \{200, 250, 300\}$.
  - ising3: Three one-dimensional Ising chain instances for dimension $|V| \in \{200, 250, 300\}$.
  - pm1d: Ten weighted graphs with edge weights chosen from $\{-1, 0, 1\}$, density 0.99, and dimension 100.
  - pm1s: Ten weighted instances with edge weights chosen from $\{-1, 0, 1\}$, density 0.1, and dimension 100.
- Random problems:
  - *nRnd_d*: Ten weighted problems for density $d \in \{0.2, 0.8\}$ and dimension $|V| \in \{100, 200, 300, 500\}$ with edge weights chosen from $\{-100, 100\}$.
  - *pRnd_d*: Ten weighted problems for density $d \in \{0.2, 0.8\}$ and dimension $|V| \in \{100, 200, 300, 500\}$ with edge weights chosen from $\{1, 100\}$. These problems are also known as the positive-weight instances.

4.3 Comparison methodology

We generate a substantial amount of data for each instance; because of space limitations we provide only the most important information. This section explains the tools used to analyze our results: the performance table, the performance profiles [9], and the data profiles [34]. We define our comparisons in terms of a set $\mathcal{P}$ of problems, a set $\mathcal{M}$ of optimization algorithms, and a set of fixed partitions or clusters $\mathcal{K}$.

4.3.1 Performance tables

The performance tables show the improvement of each method after 1 h of CPU time in our CPA. The results are divided into clusters of equal size, $k \in \{3, 10\}$. For each value of k we provide a table with the following information:

- For the `Biq Mac` instances the first column (*name*) is the problem name. For the random instances, the first column (*weight*) indicates the range of the weights.
- The density (dens.) and dimension ($|V|$) are presented in Columns 2 and 3.
- The next columns (4–15) present the UB gap at the start of CPA, the UB gap at the end, the CPU time (s) of the final iteration (t_{Last}), and the number of iterations ($\#_{ite}$) performed for each method $m \in \mathcal{M}$ over 1 h. Moreover, t_{Last} is defined for the final iteration for which the IPM is solved to optimality.

The results in the performance tables are averages for each family.

4.3.2 Performance profiles

The performance profiles are defined in terms of the gap for problem $p \in \mathcal{P}$. For method $m \in \mathcal{M}$ the performance profile is the proportion of problems for which the gap is at most α, i.e.,

$$\rho_m(\alpha) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : UP_{p,m} \leq \alpha\}. \tag{13}$$

Thus, for a given α we know the proportion of problems $p \in \mathcal{P}$ that are solved for method $m \in \mathcal{M}$.

4.3.3 Data profiles

As observed by [8], data profiles are useful for selecting the best method when a computational time limit is imposed. They show the temporal evolution of methods to a specific gap (gap_{max}). The data profiles are defined in terms of the iteration time, $itime_{p,m}$. For a given time β we define the data profile of method m by

$$d_m(\beta) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : itime_{p,m} \leq \beta \text{ and } gap_{p,m} \leq gap_{max}\}. \tag{14}$$

Thus, for a given gap_{max} and time β, we know the proportion of problems that can be solved for method $m \in \mathcal{S}$.

4.4 Computational results

This section presents and analyzes our computational results. Section 4.4.1 shows the performance tables for the `Biq Mac` instances. Section 4.4.2 presents these tables for the random instances. To compare the performance of SDP and ***LP-EIG*** we present the data profiles in Section 4.4.3 and the performance profiles in Section 4.4.4.

4.4.1 Performance tables: `Biq Mac` instances

Table 1 shows the performance of SDP, LP, and ***LP-EIG*** for the `Biq Mac` problems when $k = 3$. The SDP outperforms the linear methods in all the tests. For example, for be and bqp the first iteration of SDP is stronger than the final iterations of the linear methods. For ising2 and ising3 the SDP bounds are close to a feasible solution, but their computation is expensive: it takes approximately 1200 s to solve the IPM.

Table 2 shows the performance of SDP, LP, and ***LP-EIG*** for $k = 10$. For $k = 10$ the SDP method is more expensive and has worse performance than for $k = 3$. Moreover, ***LP-EIG*** outperforms SDP in 75% of the problems, with a smaller iteration time in most cases. The final gap of SDP is larger than the initial bound of the linear methods for ising2 and ising3.

name	dens.	$\lvert V\rvert$	SDP				*LP*				*LP-EIG*			
			$gap(\%)$				$gap(\%)$				$gap(\%)$			
			start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$
be	0.3	150	34.30	**21.49**	36	53	51.94	51.70	27	66	51.94	51.62	550	28
	0.8	150	32.95	**20.97**	50	53	46.94	46.94	0	51	46.94	37.07	143	142
bqp	0.1	100	32.23	11.35	7	49	65.01	13.09	1	806	65.01	**11.32**	29	388
g05	0.5	100	3.73	**2.04**	13	33	5.35	5.35	0	97	5.35	3.35	189	258
ising2	0.1	200	30.22	**3.30**	1129	17	25.25	17.29	143	49	25.25	14.11	150	115
		250	32.31	**4.18**	1334	18	27.78	23.66	196	50	27.78	18.52	220	84
		300	31.93	**4.10**	1250	16	26.33	23.46	134	67	26.33	19.16	348	62
ising3	0.1	200	31.08	**2.14**	1529	17	14.78	11.03	10	320	14.78	9.85	175	115
		250	33.41	**3.73**	1451	17	18.04	15.52	8	349	18.04	13.08	223	84
		300	31.96	**2.53**	1108	16	16.10	13.91	15	316	16.10	12.08	316	64
pm1d	0.9	100	31.15	**16.93**	10	32	44.72	44.72	0	58	44.72	28.42	101	265
pm1s	0.1	300	31.18	**15.81**	4	36	58.14	19.04	2	755	58.14	16.05	25	433

Table 1: Performance comparison for `Biq Mac` instances and $k = 3$.

name	dens.	$\lvert V\rvert$	SDP				*LP*				*LP-EIG*			
			$gap(\%)$				$gap(\%)$				$gap(\%)$			
			start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$
be	0.3	150	73.83	**25.94**	241	24	96.68	92.66	12	161	96.68	60.60	633	34
	0.8	150	73.77	**28.31**	268	22	92.06	91.46	126	50	92.06	46.92	111	153
bqp	0.1	100	76.27	13.62	16	36	68.47	14.05	1	782	68.47	**13.05**	15	544
g05	0.5	100	8.81	4.51	14	14	2.23	**2.23**	0	32	2.23	**2.23**	0	254
ising2	0.1	200	73.65	48.86	1029	14	23.73	16.32	123	60	23.73	**15.49**	156	113
		250	75.23	59.93	942	14	25.35	21.17	174	61	25.35	**17.75**	217	83
		300	75.34	66.30	1038	13	24.36	21.45	121	70	24.36	**17.51**	277	63
ising3	0.1	200	74.78	53.47	1037	14	13.37	**8.48**	14	268	13.37	10.22	148	113
		250	76.76	62.37	971	14	15.84	13.05	13	308	15.84	**12.72**	224	83
		300	76.54	67.63	862	13	15.06	**12.29**	22	299	15.06	12.31	313	61
pm1d	0.9	100	68.77	**20.92**	38	54	86.25	79.01	23	87	86.25	35.06	59	285
pm1s	0.1	300	71.89	18.49	9	26	76.53	18.15	1	811	76.53	**16.87**	19	607

Table 2: Performance comparison for `Biq Mac` instances and $k = 10$.

weight	dens.	$\lvert V\rvert$	SDP				LP				*LP-EIG*			
			$gap(\%)$				$gap(\%)$				$gap(\%)$			
			start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$
[-100, 100]	0.2	100	30.87	**14.45**	11	56	54.78	34.84	1	797	54.78	19.08	85	145
		200	36.33	**24.47**	112	47	55.67	55.67	4	35	55.67	44.39	167	101
		300	39.63	**31.02**	340	35	54.62	54.62	10	35	54.62	54.32	198	55
		500	45.28	**39.36**	531	23	58.00	58.00	9	44	58.00	58.00	9	8
	0.8	100	30.93	**15.59**	16	62	48.64	48.59	2	111	48.64	28.63	114	263
		200	35.65	**25.05**	106	58	48.51	48.51	1	36	48.51	41.96	190	100
		300	37.44	**29.32**	256	46	49.15	49.15	6	35	49.15	48.97	85	53
		500	42.98	**37.67**	420	25	53.18	53.18	199	41	53.18	53.18	10	25
[1, 100]	0.2	100	8.85	**4.66**	8	56	14.07	6.75	1	763	14.07	5.82	65	181
		200	7.17	**5.12**	88	53	10.39	10.39	1	39	10.39	8.89	172	102
		300	6.30	**4.93**	353	33	8.44	8.44	2	37	8.44	8.40	201	58
		500	5.45	**4.78**	515	23	6.84	6.84	10	42	6.84	6.84	10	8
	0.8	100	2.60	**1.37**	17	53	3.90	3.90	0	16	3.90	3.10	59	347
		200	2.13	**1.51**	109	51	2.86	2.86	1	28	2.86	2.63	189	93
		300	1.74	**1.36**	227	44	2.27	2.27	2	31	2.27	2.25	396	56
		500	1.61	**1.40**	495	25	1.99	1.99	10	32	1.99	1.99	10	28

Table 3: Performance comparison for random instances and $k = 3$.

4.4.2 Performance tables: *random* instances

Table 3 shows the performance of SDP, LP, and ***LP-EIG*** on the random instances when $k = 3$. Similarly to the `Biq Mac` problems, the SDP outperforms the linear methods, especially for the mixed-weight problems where the initial SDP is better than the final upper bound of the linear methods. Moreover, for most of the sparse instances, LP does not improve the initial upper bound. We conclude that for $k = 3$, the linear formulations are not competitive with the SDP.

Table 4 presents the results for $k = 10$. For mixed-weight problems the SDP has stronger bounds but their computation is expensive. For positive weights, ***LP-EIG*** usually gives the smallest gap and a competitive iteration time.

4.4.3 Data profiles

This section shows data profiles for SDP and ***LP-EIG*** for a specified gap. We plot the results for $k \in \{3, 4, 6, 7, 10, 0.1|V|\}$ for each method. In Sections 4.4.1 and 4.4.2 we saw that LP does not usually improve the initial gap, even after one hour of CPA. Therefore, we have excluded these results.

In Figure 4, we present the data profiles for instances with positive weights, i.e., all 80 problems of the family $pRnd$ and 10 from g05. Figure 5 displays the results for instances with mixed weights, i.e., 80 instances from $nRnd$, 20 from be, and 10 from bqp, pm1s, and pm1d.

weight	dens.	$\lvert V\rvert$	SDP				*LP*				*LP-EIG*			
			$gap(\%)$				$gap(\%)$				$gap(\%)$			
			start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$	start	stop	t_{Last}	$\#_{ite}$
[-100, 100]	0.2	100	70.22	**16.14**	31	43	100.41	40.00	1	905	100.41	17.46	70	178
		200	78.93	**31.98**	749	14	104.32	104.32	1	39	104.32	56.00	161	104
		300	83.19	**55.63**	846	14	102.85	102.85	2	41	102.85	70.41	255	57
		500	88.09	**74.77**	860	13	104.56	104.56	9	45	104.56	95.97	479	23
	0.8	100	71.24	**20.09**	56	59	94.43	67.68	17	176	94.43	34.89	62	290
		200	76.37	**37.61**	780	16	93.22	93.22	1	39	93.22	54.52	179	99
		300	77.90	**52.80**	662	16	92.82	92.82	2	43	92.82	63.77	275	59
		500	85.68	**73.02**	783	14	98.92	98.92	9	42	98.92	90.90	539	24
[1, 100]	0.2	100	27.19	0.12	18	11	0.12	**0.12**	0	18	0.12	**0.12**	0	17
		200	17.64	7.29	905	15	0.48	**0.48**	1	34	0.48	**0.48**	1	18
		300	14.17	10.06	943	15	1.43	**1.43**	2	38	1.43	**1.43**	2	10
		500	10.84	9.52	876	13	2.94	**2.94**	10	62	2.94	**2.94**	10	6
	0.8	100	5.99	2.11	33	17	4.27	3.24	9	662	4.27	**1.60**	31	437
		200	4.30	2.79	170	17	5.05	5.04	4	49	5.05	**2.24**	121	116
		300	3.37	2.66	227	16	3.91	3.91	2	30	3.91	**2.56**	223	57
		500	2.93	**2.55**	794	14	3.31	3.31	10	35	3.31	3.31	10	23

Table 4: Performance comparison for random instances and $k = 10$.

Positive weights. Figure 4 presents the data profiles for $gap = 3\%$ and positive weights. *LP-EIG* outperforms SDP when $k \geq 7$, especially for iterations that take less than 10 s. For example, for $k = 10$ and $itime$ =10 s *LP-EIG* solves approximately 80% of the problems while SDP does not solve any.

For $k \in \{4, 6\}$ *LP-EIG* can solve more problems in the first five seconds, but for more expensive iterations SDP can solve more problems. For $k = 3$ SDP consistently outperforms *LP-EIG*.

Mixed weights. Figure 5 presents data profiles for $gap = 30\%$ and mixed weights. For $k \geq 4$ *LP-EIG* has a slight advantage over SDP for iterations that take less than 5 s. However, neither method is satisfactory: they solve only 40% of the instances in 100 s. For $k = 3$, SDP is better than *LP-EIG*; it solves more than 50% of the instances within 10 s.

4.4.4 Performance profiles

This section shows the performance profiles of SDP and *LP-EIG*. We again exclude the LP method.

Positive weights. Figure 6 shows the performance profiles for positive weights and a time of 10 s (we consider only iterations that take less than 10 s). For $k \leq 6$ SDP

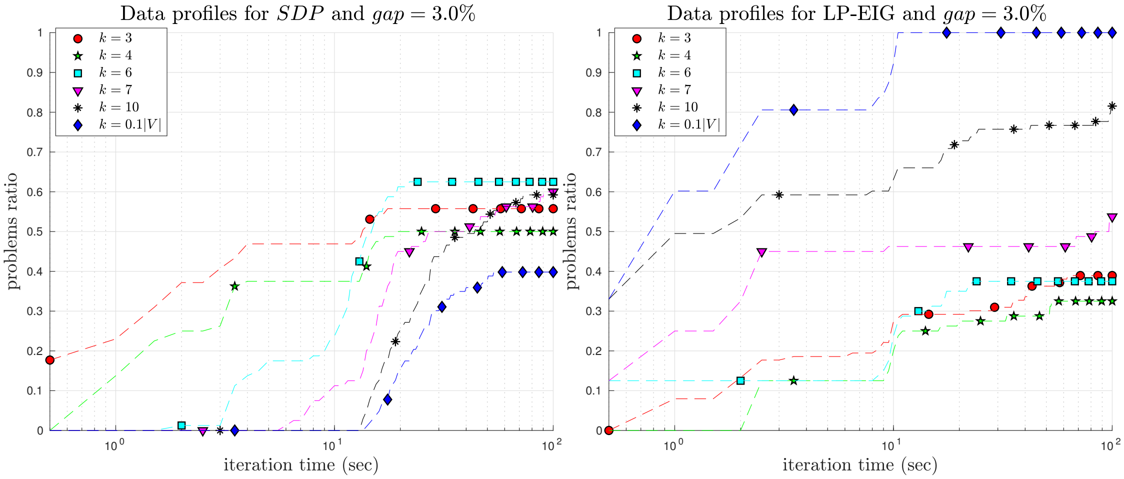

Figure 4: Data profiles for instances with positive weights for various values of partition size k.

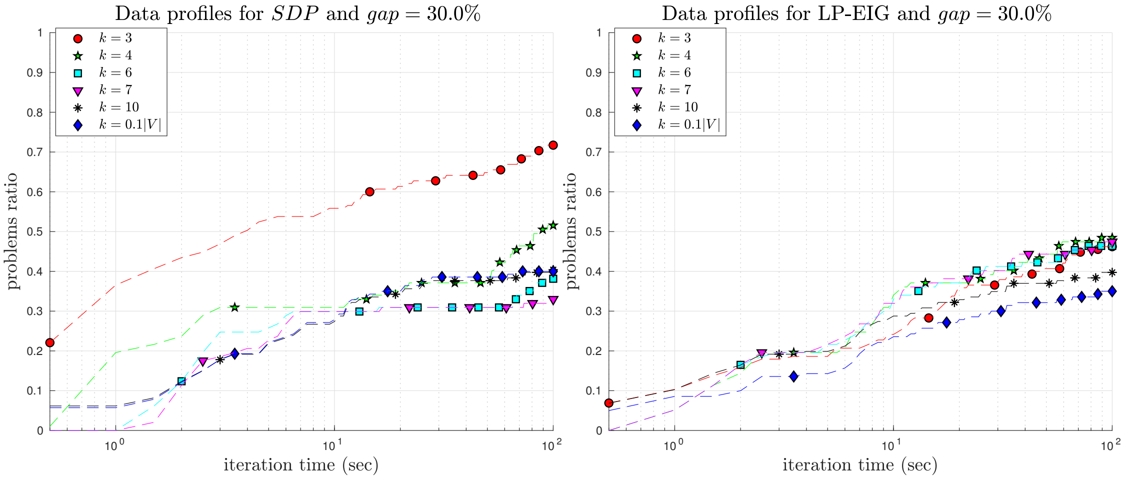

Figure 5: Data profiles for instances with mixed weights for various values of partition size k.

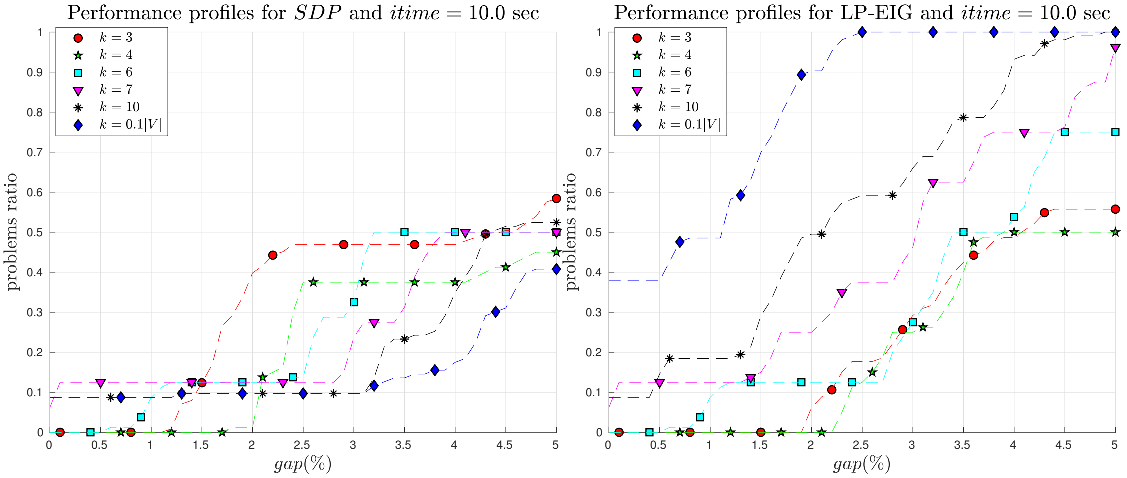

Figure 6: Performance profiles for instances with positive weights for various values of partition size k.

outperforms *LP-EIG*, especially for $gap \leq 3.5\%$. However, for $k \geq 7$ this is reversed. In particular, for $k = 10$ *LP-EIG* solves all the instances with a gap below 2.5%, whereas SDP solves only 10% of the instances.

Mixed weights. Figure 7 shows the performance profiles for a time of 20 s and mixed weights. Here, the gap goes from 0% (optimality) to 50% rather than 5% (see Figure 6), because no method could solve the instances with lower gaps, even when we allowed a higher value for $itime$. In Figure 7 we observe that for $k = 3$ SDP outperforms *LP-EIG*, but the latter is more efficient for $k \in \{4, \ldots, 7\}$. For $k \geq 10$ the two methods have similar performance.

4.5 Summary of computational tests

The tables of Sections 4.4.1 and 4.4.2 show that for $k = 3$ the SDP formulation consistently obtains the best results. However, for $k = 10$ *LP-EIG* outperforms SDP for some sparse mixed-weight instances and for positive-weight instances.

The data and performance profiles in Sections 4.4.3 and 4.4.4 indicate that *LP-EIG* is more efficient than SDP for positive weights with $k \geq 7$ and for mixed weights with $k \in \{4, \ldots, 10\}$. For $k = 3$ the SDP consistently outperforms the linear formulations.

Table 5 presents a summary of our computational results, indicating the best method for each type of problem.

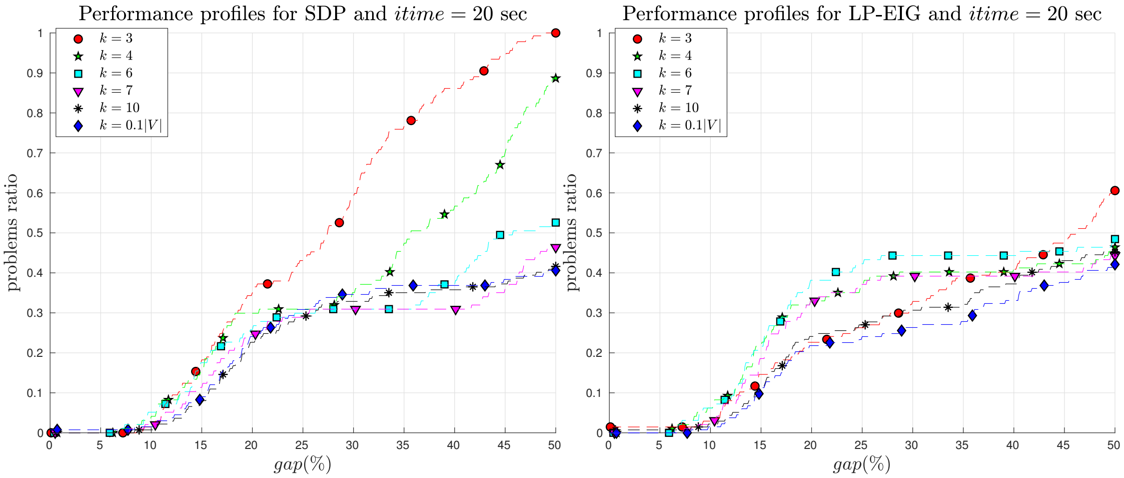

Figure 7: Performance profiles for instances with mixed weights for various values of partition size k.

Type of instance		Partition size	
weight	**density**	$k \leq 6$	$k \geq 7$
mixed	Sparse	SDP or *LP-EIG*	SDP or *LP-EIG*
	Dense	SDP or *LP-EIG*	SDP or *LP-EIG*
positive	Sparse	SDP	*LP-EIG*
	Dense	SDP	*LP-EIG*

Table 5: Best method(s) for each type of problem.

5 Discussion

We have proposed a family of SDP-based constraints (10) to strengthen the LP relaxation of the max-k-cut problem. The constraint matrix has an infinite number of rows. Therefore, we use an exact method based on eigenvalues to separate the linear solutions.

To investigate the strength of the proposed constraint, we use a CPA that relies on the early termination of an IPM, and we study the performance of the SDP and LP relaxations for various values of k and problem types. Both relaxations are strengthened by combinatorial facet-defining inequalities.

To guarantee a fair comparison, we use three benchmarks: performance tables, data profiles, and performance profiles. Our results are summarized in Table 5.

We conclude that the early termination of the IPM is effective for both the SDP and LP relaxations in the CPA. Moreover, the SDP-based constraint strengthens the LP relaxation, especially for dense instances. *LP-EIG* outperforms SDP for problems with positive weights and $k \geq 7$. Additionally, the new linear formulation is competitive for sparse instances with mixed weights.

References

[1] Z. Ales and A. Knippel. An extended edge-representative formulation for the k-partitioning problem. *Electronic Notes in Discrete Mathematics*, 52(Supplement C):333–342, 2016. INOC 2015 - 7th International Network Optimization Conference.

[2] M. F. Anjos, B. Ghaddar, L. Hupp, F. Liers, and A. Wiegele. Solving k-way graph partitioning problems to optimality: The impact of semidefinite relaxations and the bundle method. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 355–386. Springer Berlin Heidelberg, 2013.

[3] Mosek ApS. MOSEK. `http://www.mosek.com`, 2015.

[4] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[5] S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59(1):87–115, 1993.

[6] S. Chopra and M. R. Rao. Facets of the k-partition polytope. *Discrete Applied Mathematics*, 61(1):27–48, 1995.

[7] 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.

[8] V. J. Rodrigues de Sousa, M. F. Anjos, and S. Le Digabel. Computational Study of Valid Inequalities for the Maximum k-Cut Problem. Technical Report G-2016-17, Les cahiers du GERAD, 2016. To appear in *Annals of Operations Research*.

[9] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[10] A. Eisenblätter. The semidefinite relaxation of the k-partition polytope is strong. In W. J. Cook and A. S. Schulz, editors, *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 273–290. Springer, Berlin, Heidelberg, 2002.

[11] J. Fairbrother and A. N. Letchford. Projection results for the k-partition problem. *Discrete Optimization*, 26:97–111, 2017.

[12] J. Fairbrother, A. N. Letchford, and K. Briggs. A two-level graph partitioning problem arising in mobile wireless communications. *Computational Optimization and Applications*, 69(3):653–676, 2018.

[13] A. Frieze and M. Jerrum. Improved approximation algorithms for maxk-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

[14] 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.

[15] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[16] J. Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218:587–601, 2012.

[17] J. Gondzio, P. González-Brevis, and P. Munari. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation*, 8(1):47–82, 2016.

[18] G. Guennebaud, B. Jacob, et al. `Eigen`. `http://eigen.tuxfamily.org`, 2010.

[19] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[20] C. Helmberg. *Semidefinite Programming for Combinatorial Optimization*. Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Berlin-Dahlem, Germany, 1st edition, 2000.

[21] R. Hettich and K. O. Kortanek. Semi-infinite programming: Theory, methods, and applications. *SIAM Review*, 35(3):380–429, 1993.

[22] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[23] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998.

[24] K. Krishnan and J. E. Mitchell. Semi-infinite linear programming approaches to semidefinite programming problems. Technical Report 37, Fields Institute Communications Series, 2001.

[25] K. Krishnan and J. E. Mitchell. A semidefinite programming based polyhedral cut and price approach for the maxcut problem. *Computational Optimization and Applications*, 33(1):51–71, 2006.

[26] N. Krislock, J. Malick, and F. Roupin. Improved semidefinite bounding procedure for solving max-cut problems to optimality. *Mathematical Programming*, 143(1):61–86, 2012.

[27] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In *New Optimization Algorithms in Physics*, pages 47–69. Wiley-VCH Verlag GmbH & Co. KGaA, 2005.

[28] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95(1):91–101, 2003.

[29] F. Ma and J.-K. Hao. A multiple search operator heuristic for the max-k-cut problem. *Annals of Operations Research*, 248(1):365–403, 2017.

[30] J. E. Mitchell. Computational experience with an interior point cutting plane algorithm. *SIAM Journal on Optimization*, 10(4):1212–1227, 2000.

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

[32] J. E. Mitchell, P. M. Pardalos, and M. G. C. Resende. Interior point methods for combinatorial optimization. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume 1–3*, pages 189–297. Springer US, Boston, MA, 1999.

[33] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[34] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[35] P. Munari and J. Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, 40(8):2026–2036, 2013.

[36] C. Niu, Y. Li, R. Qingyang Hu, and F. Ye. Femtocell-enhanced multi-target spectrum allocation strategy in LTE-A HetNets. *IET Communications*, 11(6):887–896, 2017.

[37] L. Palagi, V. Piccialli, F. Rendl, G. Rinaldi, and A. Wiegele. Computational approaches to max-cut. In M. F. Anjos and J. B. Lasserre, editors, *Handbook of Semidefinite, Conic and Polynomial Optimization: Theory, Algorithms, Software and Applications*, International Series in Operations Research and Management Science, pages 821–847. Springer, New York, 2011.

[38] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[39] F. Rendl. Semidefinite relaxations for partitioning, assignment and ordering problems. *4OR*, 10(4):321–346, 2012.

[40] F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121(2):307–335, 2010.

[41] G. Rinaldi. Rudy, a graph generator. `https://www-user.tu-chemnitz.de/~helmberg/sdp_software.html`, 2018.

[42] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[43] G. Wang and H. Hijazi. Exploiting sparsity for the min k-partition problem. ArXiv e-prints, September 2017. arXiv:1709.00485.

[44] A. Wiegele. Biq mac library - Binary quadratic and max cut library. `http://biqmac.uni-klu.ac.at/biqmaclib.html`, 2015.