EURO Journal on Computational Optimization (2019) 7:123–151
https://doi.org/10.1007/s13675-019-00110-y

ORIGINAL PAPER

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

Vilmar Jefté Rodrigues de Sousa[1] · Miguel F. Anjos[1] · Sébastien Le Digabel[1]

Received: 4 April 2018 / Accepted: 1 March 2019 / Published online: 16 March 2019

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

Mathematics Subject Classification 65K05 · 90C22 · 90C35

✉ Sébastien Le Digabel
sebastien.le.digabel@gerad.ca
http://www.gerad.ca/Sebastien.Le.Digabel

Vilmar Jefté Rodrigues de Sousa
Vilmar.de.sousa@gerad.ca

Miguel F. Anjos
anjos@stanfordalumni.org
http://www.miguelanjos.com

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

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 (sometimes called clusters or colors) such that the sum of the edges with end points in different partitions is maximized.

The max-k-cut problem is equivalent to the minimum k-partition problem (Ghaddar et al. 2011; Wang and Hijazi 2017), and the special case $k = 2$ that is known as the max-cut problem has attracted considerable attention; see, e.g., Barahona et al. (1988), Goemans and Williamson (1995), Krislock et al. (2012), Palagi et al. (2011), and Rendl et al. (2010).

Many industrial applications can be formulated as the max-k-cut problem, including VLSI layout design (Barahona et al. 1988), statistical physics (Liers et al. 2005), and wireless communication problems (Fairbrother et al. 2018; Niu et al. 2017).

The general max-k-cut is known to be $\mathcal{NP}$-complete (Papadimitriou and Yannakakis 1991). Nonetheless, many relaxations (Chopra and Rao 1995; Rendl 2012), heuristics (Ma and Hao 2017), approximations (Frieze and Jerrum 1997; Karger et al. 1998), and exact methods (Anjos et al. 2013; Fairbrother and Letchford 2017; Mitchell 2003) 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 Sect. 4 discusses the test results. Finally, some concluding remarks are made in Sect. 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 Frieze and Jerrum (1997), 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:

$$
\begin{aligned}
(\textit{MkC-SDP}) \quad & \max_{X} && \frac{(k-1)}{k} \sum_{i,j \in V}^{i<j} w_{ij}(1 - X_{ij}) && && (1)\\
& \text{s.t.} && X_{ii} = 1 && \forall i \in V && (2)\\
& && X_{ij} \geq \frac{-1}{k-1} && \forall i, j \in V, i < j && (3)\\
& && X \succeq 0 && && (4)
\end{aligned}
$$

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 (de Klerk et al. 2004; Frieze and Jerrum 1997) and exact methods (Anjos et al. 2013; Ghaddar et al. 2011). In particular, Frieze and Jerrum (1997) extends the max-cut approximation of Goemans and Williamson (1995) to the max-k-cut. In Anjos et al. (2013), the bundleBC algorithm is proposed to solve max-k-cut problems with 60 vertices by combining the SDP branch-and-cut method of Ghaddar et al. (2011) with the principles of the `Biq Mac` algorithm (Rendl et al. 2010). In Anjos et al. (2013), the authors show that their method achieves a dramatic speedup in comparison with Ghaddar et al. (2011), especially when $k = 3$.

1.1.2 Linear formulation

Chopra and Rao (1995) 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:

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

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., Chopra and Rao (1993), Chopra and Rao (1995) and Mitchell (2003). In Chopra and Rao (1993) and Chopra and Rao (1995), 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 Fairbrother and Letchford (2017), 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 Fairbrother et al. (2018). 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 (Fairbrother et al. 2018; Hopcroft and Tarjan 1973; Seidman 1983), and a chordal extension (Heggernes 2006; Wang and Hijazi 2017). The second disadvantage is mitigated by a CPA (Sect. 3) 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 (Ales and Knippel 2016; Chopra and Rao 1995; Fairbrother et al. 2018). In Ales and Knippel (2016), 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 (Ghaddar et al. 2011), 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 Eisenblätter (2002), 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. On the other hand, high computational times are the price to pay for the strength of SDP relaxations (Anjos et al. 2013).

The linear and semidefinite relaxations of the graph partitioning problem where each partition must have about the same cardinality (also known as the k-equipartition problem) are considered in Lisser and Rendl (2003). 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 Sect. 3.1.2). However, for small values of k, the latter outperforms the former.

2 SDP-based inequality

The pioneering work of Shor (1998) leads to an approach to optimize semidefinite programming based on integrating the constraint that restricts the smallest eigenvalue of X to be nonnegative. We refer to this infinite class of valid inequalities as *SDP-based inequalities*.

Optimization problems with an infinite number of constraints are known as semi-infinite programming (SIP) problems. This section briefly reviews SIP and presents a class of SDP-based inequalities that can be used within the LP formulation.

2.1 Semi-infinite formulation of SDP

SIP can be defined as an optimization problem with finitely many variables and infinitely many constraints. The survey paper (Hettich and Kortanek 1993) discusses the theory, algorithms, and applications of semi-infinite programming. The linear semi-infinite programming (LSIP) approach to solve generic SDPs was studied in Krishnan and Mitchell (2001) and Sherali and Fraticelli (2002).

We note that the convex constraint (4) is equivalent to (Helmberg 2000, Theorem 1.1.8):

$$\mu^T X \mu \geq 0 \qquad \forall \mu \in \mathbb{R}^n, \|\mu\| = 1 \tag{9}$$

where $n = |V|$. By replacing (4) by (9) in *MkC-SDP*, we obtain the LSIP formulation of SDP.

There is an infinite number of constraints in (9). In Sherali and Fraticelli (2002), the authors propose the use of the SDP-based inequalities (or semidefinite cuts) as a mechanism to tighten the reformulation linearization technique. Furthermore, in the cut-and-price approach proposed in Krishnan and Mitchell (2006), the authors use the LSIP of the dual SDP formulation for the *max-cut* problem. Their results suggest that this approach is able to solve large-scale instances of SDP.

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 $x_{ij} = \frac{k-1}{k} X_{ij} + \frac{1}{k}$ and $X_{ij} = \frac{k}{k-1} x_{ij} - \frac{1}{k-1}$ 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 Sect. 2.2 to Constraint (9), we derive the following class of inequalities for *MkC-LP*:

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

These SDP-based cuts comprise a relaxation of the underlying semidefinite constraint. In Krishnan and Mitchell (2001), the authors prove that these inequalities ensure that

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.

Fig. 1 Cutting-plane algorithm

the set of linear solutions is feasible for the SDP. In Sect. 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, and 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 Fig. 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 Rodrigues de Sousa et al. (2018), for Constraint (3) in the SDP formulation, and for Constraint (10) proposed in this work.

3.1.1 Separation of combinatorial inequalities

Some valid and facet-inducing inequalities have been proposed in Chopra and Rao (1995) for the *MkC-LP*. Five of these families of constraints are explored computationally in Rodrigues de Sousa et al. (2018), where heuristic and exact methods are proposed. In this work, we replicate the best separation routines of Rodrigues de Sousa et al. (2018) for the following families of inequalities:

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

In Rodrigues de Sousa et al. (2018), 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 Sect. 3.1.4).

3.1.2 Separation of bound inequalities

In Helmberg (2000), 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 the constraints (3) iteratively. The exact separation of constraints (3) can be done in polynomial time with a complete enumeration of all edges $e \in E$ of the graph. For each iteration of the CPA, we add only the $NbIneq$ most violated of these inequalities (see Sect. 3.1.4).

Figure 2 shows data profiles (see explanation in Sect. 4.3.4) for the SDP formulation with and without separation of the bound inequalities for $k \in \{3, 10\}$ for 68 instances of the `Biq Mac` library (see Sect. 4.2). Both methods in Fig. 2 apply the separation of combinatorial inequalities (Sect. 3.1.1) and were solved with `MOSEK` (MOSEK ApS 2015). The difference between *No separation* and *With bound separation* is that the latter method does not separate the constraints (2) in the CPA, i.e., the *No separation* method inserts all the $n(n-1)/2$ constraints in the first iteration of the CPA.

Figure 2 shows that the method that applies the separation obtains better results. For example, for $k = 3$ the *With bound separation* method finds solutions with a gap [see Eq. (12)] of 30% for more than 70% of the problems in less than 10 seconds, while the *No separation* method solves the first instances only after 100 seconds. Moreover, computational tests on instances with $|V| \geq 300$ show that the first iteration of the CPA takes more than 1 h to be completed with the *No separation* method.

3.1.3 Separation of SDP-based inequalities

The family of SDP-based inequalities (10) integrates an infinite number of constraints in the LP relaxation of max-k-cut. Rather than solving the semi-infinite program, we adopt the strategy of generating only suitable constraints by a polynomial time separation routine that is based on the eigenvalues of a symmetric matrix. Let $\hat{x}$ be

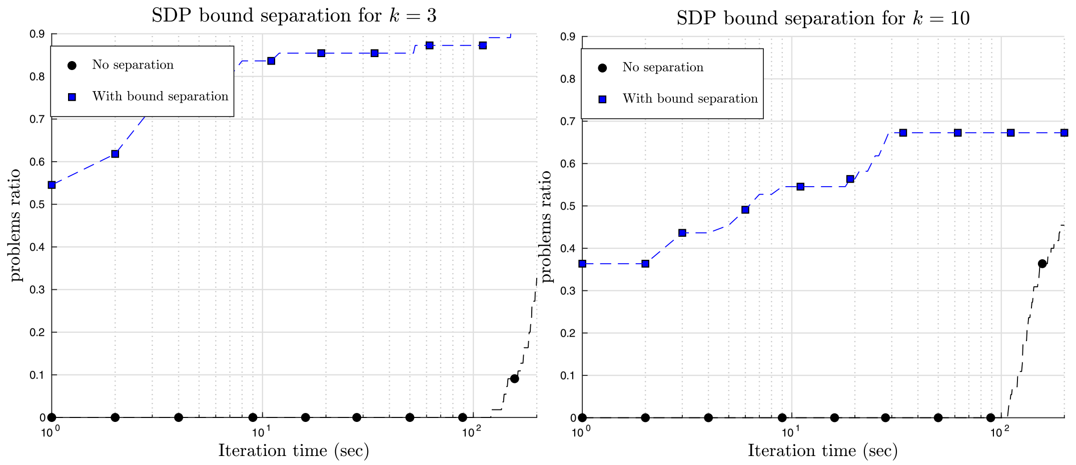

Fig. 2 Separation of bound inequalities (3) in the SDP formulation for a gap of 30%

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 $\hat{x}$:

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

where v_i is the ith entry of the eigenvector $\mathbf{v}$ corresponding to the eigenvalue λ of $\hat{X}$. 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 this eigenvalue separation. We use `Eigen` (Guennebaud et al. 2010) 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 Rodrigues de Sousa et al. (2018), 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 $NbIneq = 100$ for the SDP formulations, similarly to Rodrigues de Sousa et al. (2018).

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 Mitchell et al. (1999), 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. Therefore, we use the variable Ite_{drop} to indicate the interval of CPA iterations that the search is realized. Computational results for $Ite_{drop} \in \{2, 3, 5, 7\}$ show that the SDP and LP formulations are more efficient when the dropping is executed at every third or fifth iteration of the CPA. Therefore, we fix the dropping method at every third iteration of the CPA method ($Ite_{drop} = 3$).

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` (MOSEK ApS 2015). Our computational tests indicated that the default IPM is not efficient so, inspired by the `PDCGM` solver (Gondzio et al. 2016), 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 Gondzio (2012) and Mitchell (2000), 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. Furthermore, IPMs can exploit parallelism easily (Mitchell et al. 1999).

The main change performed in the IPM is that we use the **early termination** technique. We apply the separation routine in a non-optimal solution that is obtained by solving the relaxations approximately with a relative dual termination tolerance ε_T. As shown in Munari and Gondzio (2013), 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 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 Mitchell (2000), 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 Sect. 3.1.4) and decrease ε_T if we have fewer than $NbIneq$ violated constraints. Experimental tests varying the initial $\varepsilon_T \in \{.25, .50, .75, .90\}$ show that on average, when $\varepsilon_T = 0.75$, the SDP and LP formulations obtain, for 50% of the tests, the best results for $k = 3$ and 75% than for $k = 10$.

The second disadvantage is mitigated by occasionally solving the relaxation to optimality. Thus, at each Ite_{opt} iteration of the CPA the relaxations are solved to their optimality without applying the early termination. Computational tests with $Ite_{opt} \in \{1, 2, 5, 7, 9, 12, 15, 20\}$ for the LP and SDP formulations show that the best results are obtained when $Ite_{opt} \in \{2, 5\}$. For example, when $Ite_{opt} \in \{2, 5\}$ and $k = \{3, 10\}$, the SDP formulation solves 80% of instances with a gap inferior to 5% in 10 seconds, while other options cannot solve 50% of instances with the same gap within the same length of time. We fix $Ite_{optLP} = 5$ for the LP formulations (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 Sect. 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 $|V| = 100$

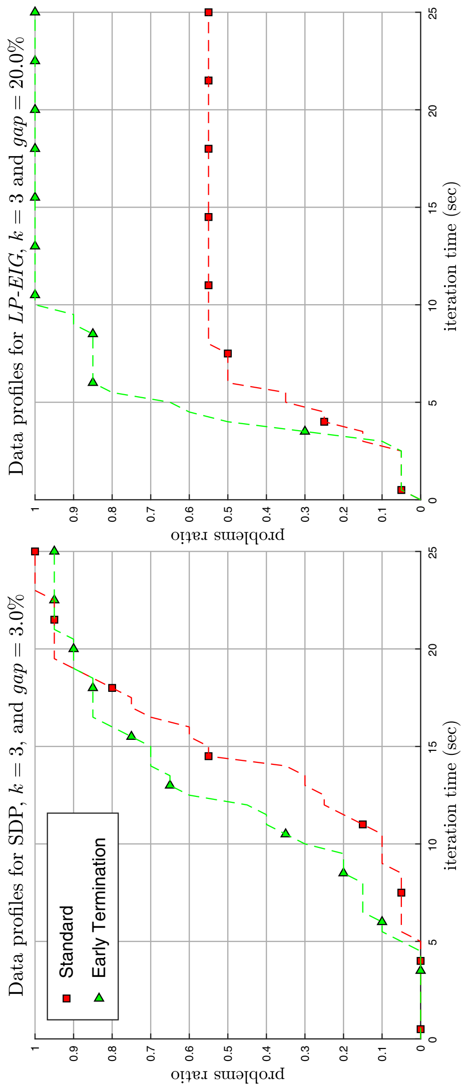

Fig. 3 Study of early termination in IPM

and $k = 3$, and the results can be generalized to other graphs. The gap (12) 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` (MOSEK ApS 2015) on a Linux PC with two Intel(R) Xeon(R) 3.07 GHz processors. We performed tests for $k \in \{3, 4, 6, 7, 10, 0.1|V|\}$ 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 (Mladenović and Hansen 1997).
- *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 (Sect. 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 (Sect. 3.1.2).

4.2 Instances

We consider 228 instances; 68 are from the `Biq Mac` library (Wiegele 2015), and 160 were randomly generated using `rudy` (Rinaldi 2018).

- `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: Six 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.
  - 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: separation routine tables, performance tables, data profiles (Moré and Wild 2009), and performance profiles (Dolan and Moré 2002). We define our comparisons in terms of a set $\mathcal{P}$ of problems, a set $\mathcal{M}$ of optimization algorithms, and a set of partition sizes $\mathcal{K}$.

4.3.1 Separation routine tables

The separation routine tables show the performance of the separation routine of each constraint presented in Sect. 3.1 for each method, after 1 h of CPU time. In these tables, the results are averages over the instances presented in Sect. 4.2. For clarity, we only show results for dense instances (density superior to 50%). The following information is provided in each table:

- Column 1 presents the number of partitions $k \in \{3, 10\}$ allowed.
- Column 2 lists the separation routines:
  - The rows triangle, clique, general clique, wheel and bicycle wheel report results for combinatorial inequalities.
  - The row eigen presents the results for separation routine of the SDP-based inequalities (see Sect. 3.1.3). Therefore, it is applicable only for the *LP-EIG* method.
  - The row SDP bound gives the results for the separation of the SDP bounds (see Sect. 3.1.2).
  - The row entitled `MOSEK` presents the results of the average percentage of time spent to solve the relaxation by `MOSEK` using IPM.
  - The last row of each partition shows the average time, in seconds, of iterations of the CPA and the total number of iterations (ite) performed within 1 h for each method.
- Columns 3 to 8 present, for each method, the percentage of time spent (% time) for each constraint and the average number of inequalities incorporated at each iteration of the CPA ($ineq/ite$). For the row "CPA iterations", the value given under $ineq/ite$ is the total number of CPA iterations executed for each method until the stopping criteria are satisfied.

4.3.2 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 partitions of equal size, $k \in \{3, 10\}$. For each value of k, we provide a table with the following information:

- Column 1 gives the instance name for the `Biq Mac` instances, and the range of the weights for the random instances.
- Columns 2 and 3 give, respectively, the density and dimension ($|V|$) of the instances.
- Columns 4 to 15 report 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.

4.3.3 Data profiles

As observed in our earlier work (Rodrigues de Sousa et al. 2018), data profiles are useful for selecting the best method when a computational time limit is imposed. These profiles show the temporal evolution of methods to a specific gap (gap_{max}). They 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

$$d_m(\beta) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : itime_{p,m} \leq \beta \text{ and } gap_{p,m} \leq gap_{max}\}. \tag{13}$$

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

4.3.4 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.,

$$\rho_m(\alpha) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : UP_{p,m} \leq \alpha\}. \tag{14}$$

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

4.4 Computational results

This section presents and analyzes our computational results. Section 4.4.2 shows the separation routine study tables for the dense instances. Section 4.4.2 shows the performance tables for the `Biq Mac` instances. Section 4.4.3 presents these tables for the random instances. To compare the performance of SDP and *LP-EIG*, we present the data profiles in Sect. 4.4.4 and the performance profiles in Sect. 4.4.5.

4.4.1 Separation routine tables

Tables 1 and 2 show the results of the separation routines for the SDP, LP, and *LP-EIG* methods for dense instances. Table 1 plots the results for instances with mixed-weight edges ($w_e \in [-100, 100]$), and Table 2 presents results for instances that have positive weights ($w_e \in [1, 100]$). The results in both tables demonstrate that the IPM used to solve the relaxations takes on average 86% of the time of each iteration of the CPA. The separation routine of bicycle wheel is most expensive.

For the LP method, triangle followed by wheel and bicycle wheel are the most important inequalities. Moreover, we observe that due to the large number of inequalities that are included at each iteration of the CPA, the LP is the most expensive method when $k = 3$. However, in Sect. 4.4.2, we observe that LP is often the method with the smallest final iteration CPU time (t_{Last}). The reason is that t_{Last} is calculated after dropping unimportant inequalities.

For the SDP method, triangle and SDP bound are the most important inequalities. We observe that the CPA iterations of SDP are more expensive for a large number of partitions ($k = 10$) mostly due to the number of SDP bound inequalities that are violated (added). Results in Table 2 demonstrate that the SDP method includes more SDP bound inequalities and that its CPA iterations are more expensive for instances with positive weight than for mixed-weight instances.

Table 1 Separation routine study for dense instances with weights $w_e \in [-100, 100]$

Partition	Separation routine	*SDP*		*LP*		*LP-EIG*	
		Time	*ineq/ite*	Time	*ineq/ite*	Time	*ineq/ite*
$k = 3$	triangle	0.1%	27	0.1%	572	0.0%	8
	clique	0.1%	4	0.1%	57	0.1%	17
	general clique	0.1%	2	0.1%	56	0.2%	0
	wheel	0.2%	12	0.1%	112	0.2%	52
	bicycle wheel	11.7%	7	7.7%	165	4.4%	17
	eigen	–	–	–	–	0.3%	6
	SDP bound	0.0%	9	–	–	–	–
	MOSEK	88.4%	–	79.2%	–	93.5%	–
	CPA iterations	34.6 s	100 *ite*	48.0 s	76 *ite*	30.5 s	120 *ite*
$k = 10$	triangle	0.0%	8	0.1%	644	0.0%	13
	clique	0.4%	0	0.5%	0	0.9%	0
	general clique	0.5%	0	0.6%	0	1.1%	0
	wheel	0.1%	46	0.1%	141	0.2%	61
	bicycle wheel	7.1%	1	8.2%	214	7.1%	21
	eigen	–	–	–	–	0.4%	6
	SDP bound	0.0%	38	–	–	–	–
	MOSEK	92.1%	–	81.5%	–	88.6%	–
	CPA iterations	65.1 s	56 *ite*	47.3 s	77 *ite*	25.4 s	145 *ite*

For the *LP-EIG* method, wheel and bicycle wheel are the most important inequalities. The *LP-EIG* is the method that performs more CPA iterations in one hour. Therefore, it is the method with the fastest iterations. In general, *LP-EIG* does not include general clique inequalities and just a few SDP-based inequalities are needed at each iteration of the CPA.

For $k = 10$, the LP and *LP-EIG* methods are able to perform almost double the number of CPA iterations for the instances with positive weights than for those of mixed weights.

We point out that the percentages in Tables 1 and 2 do not always add up to 100% because the results in these tables do not include all the procedures of the CPA. For example, the time spent dropping unimportant inequalities (see Sect. 3.2) is not considered.

4.4.2 Performance tables: `Biq Mac` instances

Table 3 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 Separation routine study for dense instances with $w_e \in [1, 100]$

Partition	Separation routine	SDP		LP		*LP-EIG*	
		Time	$ineq/ite$	Time	$ineq/ite$	Time	$ineq/ite$
$k = 3$	triangle	0.1%	19	0.1%	572	0.0%	6
	clique	0.1%	7	0.1%	57	0.1%	14
	general clique	0.1%	3	0.1%	56	0.2%	0
	wheel	0.2%	11	0.1%	112	0.1%	58
	bicycle wheel	10.1%	5	7.7%	165	3.1%	16
	eigen	–	–	–	–	0.3%	6
	SDP bound	0.0%	11	–	–	–	–
	`MOSEK`	87.2%	–	79.2%	–	94.9%	–
	CPA iterations	43.4 s	83 ite	48.0 s	76 ite	30.6 s	119 ite
$k = 10$	triangle	0.0%	9	0.3%	618	0.1%	7
	clique	0.4%	0	0.7%	13	1.6%	0
	general clique	0.4%	0	0.7%	11	1.9%	1
	wheel	0.1%	55	0.4%	121	0.4%	62
	bicycle wheel	6.5%	1	26.4%	118	26.7%	4
	eigen	–	–	–	–	0.6%	4
	SDP bound	0.0%	205	–	–	–	–
	`MOSEK`	92.8%	–	55.4%	–	66.5%	–
	CPA iterations	69.5 s	48 ite	16.0 s	226 ite	14.5 s	249 ite

Moreover, the results show that the SDP-based Constraint (10) improves the final gap by an average of 5% in Table 3.

Table 4 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. For some instances of ising3, the LP method outperforms *LP-EIG* since the LP method executes more iterations of the CPA and adds more inequalities.

4.4.3 Performance tables: *random* instances

Table 5 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 problems that contain both positive and negative edges (mixed weights instances) where the initial SDP is better than the final upper bound of the linear methods. Moreover, for most of the sparse instances, the LP method does not improve the initial upper bound, and for some large instances ($|V| \geq 300$) the combinatorial and SDP-based inequalities included in the LP methods could not improve the initial bound. Therefore, we conclude that for $k = 3$, the linear formulations are not competitive with the SDP.

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

Name	Density	$\|V\|$	SDP				LP				$LP\text{-}EIG$			
			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

Bold value represents for each instance, the value of the method that obtained the best upper bound

Table 4 Performance comparison for `Biq Mac` instances and $k = 10$

Name	Density	$\|V\|$	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

Bold value represents for each instance, the value of the method that obtained the best upper bound

Table 5 Performance comparison for random instances and $k = 3$

Weights	Density	$\|V\|$	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

Bold value represents for each instance, the value of the method that obtained the best upper bound

Table 6 presents the results for $k = 10$. For mixed-weight instances, 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. Table 6 shows that for sparse and positive instances the LP and *LP-EIG* methods have the smallest initial gaps but were unable to improve them.

4.4.4 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 Sects. 4.4.2 and 4.4.3, we saw that LP does not usually improve the initial gap, even after one hour of CPA. Therefore, we have excluded these results.

In Fig. 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.

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.

4.4.5 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 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 0% to 5% (see Fig. 6), because no method could solve the instances with lower gaps, even when we allowed a higher value for $itime$. In Fig. 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.

Table 6 Performance comparison for random instances and $k = 10$

Weights	Density	$\|V\|$	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

Bold value represents for each instance, the value of the method that obtained the best upper bound

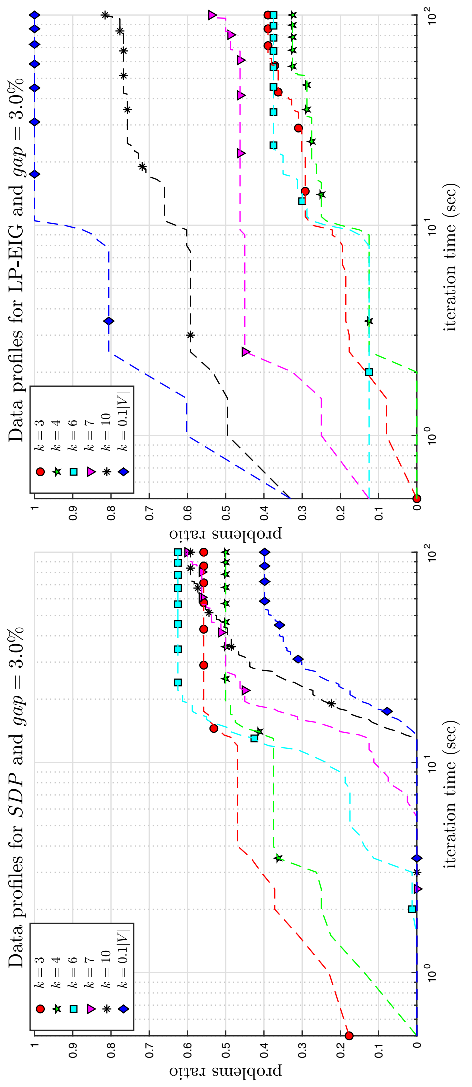

Fig. 4 Data profiles for instances with positive weights for various values of partition size k

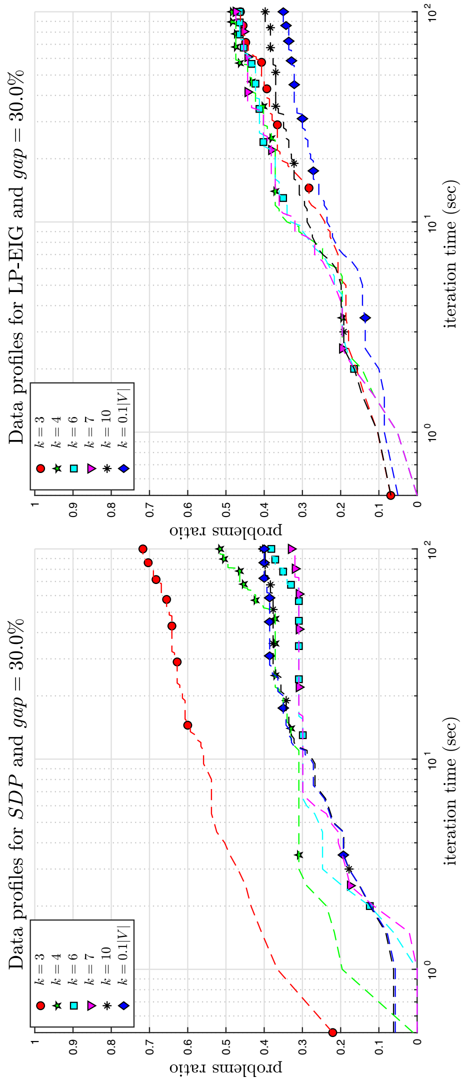

Fig. 5 Data profiles for instances with mixed weights for various values of partition size k

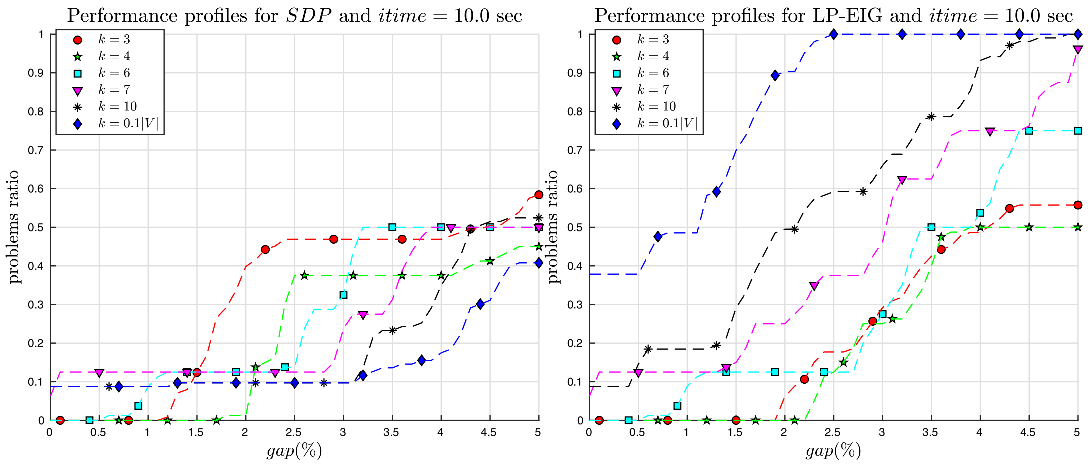

Fig. 6 Performance profiles for instances with positive weights for various values of partition size k

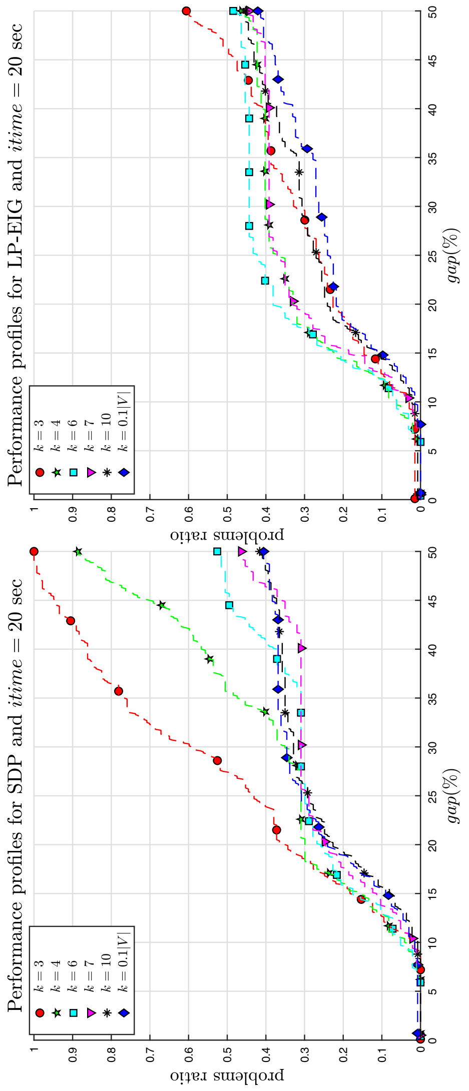

Fig. 7 Performance profiles for instances with mixed weights for various values of partition size k

Table 7 Best methods for each type of problem

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*

4.5 Summary of computational tests

The tables of Sects. 4.4.2 and 4.4.3 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 Sects. 4.4.4 and 4.4.5 indicate that *LP-EIG* is more efficient than SDP for positive weights with $k \geq 7$ and for mixed weights with $k \in \{4, \dots, 10\}$. For $k = 3$, the SDP consistently outperforms the linear formulations.

Table 7 presents a summary of our computational results, indicating the best method 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 7.

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.

Future research involves the implementation of a branch-and-cut algorithm to find the optimal solution of max-*k*-cut using SDP-based inequalities and to study ways of strengthening the SDP-based inequalities, for example, by using combinatorial arguments such as the ones in Avis and Umemoto (2003), Chopra and Rao (1995), Galli et al. (2011), Helmberg and Rendl (1998), and Laurent and Poljak (1996).

Acknowledgements The authors thank the associate editor and two anonymous referees for their helpful comments on an earlier version of this article.

References

Ales Z, Knippel A (2016) An extended edge-representative formulation for the k-partitioning problem. Electron Notes Discrete Math 52(Supplement C):333–342 INOC 2015—7th international network optimization conference

Anjos M F, Ghaddar B, Hupp L, Liers F, Wiegele A (2013) Solving k-way graph partitioning problems to optimality: the impact of semidefinite relaxations and the bundle method. In: Jünger Michael, Reinelt Gerhard (eds) Facets of combinatorial optimization. Springer, Berlin, pp 355–386

Avis D, Umemoto J (2003) Stronger linear programming relaxations of max-cut. Math Program 97(3):451–469

Barahona F, Grötschel M, Jünger M, Reinelt G (1988) An application of combinatorial optimization to statistical physics and circuit layout design. Oper Res 36(3):493–513

Chopra S, Rao MR (1993) The partition problem. Math Program 59(1):87–115

Chopra S, Rao MR (1995) Facets of the k-partition polytope. Discrete Appl Math 61(1):27–48

de Klerk E, Pasechnik DV, Warners JP (2004) On approximate graph colouring and max-k-cut algorithms based on the θ-function. J Comb Optim 8(3):267–294

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91(2):201–213

Eisenblätter A (2002) The semidefinite relaxation of the k-partition polytope is strong. In: Cook WJ, Schulz AS (eds) Integer programming and combinatorial optimization. Lecture notes in computer science, vol 237. Springer, Berlin, pp 273–290

Fairbrother J, Letchford AN (2017) Projection results for the k-partition problem. Discrete Optim 26:97–111

Fairbrother J, Letchford AN, Briggs K (2018) A two-level graph partitioning problem arising in mobile wireless communications. Comput Optim Appl 69(3):653–676

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

Galli L, Kaparis K, Letchford AN (2011) Gap inequalities for non-convex mixed-integer quadratic programs. Oper Res Lett 39(5):297–300

Ghaddar B, Anjos MF, Liers F (2011) A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. Ann Oper Res 188(1):155–174

Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J ACM 42(6):1115–1145

Gondzio J (2012) Interior point methods 25 years later. Eur J Oper Res 218:587–601

Gondzio J, González-Brevis P, Munari P (2016) Large-scale optimization with the primal-dual column generation method. Math Program Comput 8(1):47–82

Guennebaud G, Jacob B, et al (2010) `Eigen`. http://eigen.tuxfamily.org. Accessed Feb 2018

Heggernes P (2006) Minimal triangulations of graphs: a survey. Discrete Math 306(3):297–317

Helmberg C (2000) Semidefinite programming for combinatorial optimization, 1st edn. Konrad-Zuse-Zentrum für Informationstechnik, Berlin

Helmberg C, Rendl F (1998) Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Math Program 82(3):291–315. https://doi.org/10.1007/BF01580072

Hettich R, Kortanek KO (1993) Semi-infinite programming: theory, methods, and applications. SIAM Rev 35(3):380–429

Hopcroft J, Tarjan R (1973) Algorithm 447: efficient algorithms for graph manipulation. Commun ACM 16(6):372–378

Karger D, Motwani R, Sudan M (1998) Approximate graph coloring by semidefinite programming. J ACM 45(2):246–265

Krishnan K, Mitchell JE (2001) Semi-infinite linear programming approaches to semidefinite programming problems. Technical Report 37, Fields Institute Communications Series

Krishnan K, Mitchell JE (2006) A semidefinite programming based polyhedral cut and price approach for the maxcut problem. Comput Optim Appl 33(1):51–71

Krislock N, Malick J, Roupin F (2012) Improved semidefinite bounding procedure for solving max-cut problems to optimality. Math Program 143(1):61–86

Laurent M, Poljak S (1996) Gap inequalities for the cut polytope. Eur J Comb 17(2):233–254

Liers F, Jünger M, Reinelt G, Rinaldi G (2005) Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In: New optimization algorithms in physics. Wiley-VCH Verlag GmbH & Co. KGaA, pp 47–69

Lisser A, Rendl F (2003) Graph partitioning using linear and semidefinite programming. Math Program 95(1):91–101
Ma F, Hao J-K (2017) A multiple search operator heuristic for the max-k-cut problem. Ann Oper Res 248(1):365–403
Mitchell JE (2000) Computational experience with an interior point cutting plane algorithm. SIAM J Optim 10(4):1212–1227
Mitchell JE (2003) Realignment in the National Football League: did they do it right? Naval Res Logist 50(7):683–701
Mitchell JE, Pardalos PM, Resende MGC (1999) Interior point methods for combinatorial optimization. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization, vol 1–3. Springer, Boston, pp 189–297
Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100
Moré JJ, Wild SM (2009) Benchmarking derivative-free optimization algorithms. SIAM J Optim 20(1):172–191
Mosek ApS (2015) MOSEK http://www.mosek.com. Accessed Feb 2018
Munari P, Gondzio J (2013) Using the primal-dual interior point algorithm within the branch-price-and-cut method. Comput Oper Res 40(8):2026–2036
Niu C, Li Y, Qingyang Hu R, Ye F (2017) Femtocell-enhanced multi-target spectrum allocation strategy in LTE-A HetNets. IET Commun 11(6):887–896
Palagi L, Piccialli V, Rendl F, Rinaldi G, Wiegele A (2011) Computational approaches to max-cut. In: Anjos MF, Lasserre JB (eds) Handbook of semidefinite, conic and polynomial optimization: theory, algorithms, software and applications. International series in operations research and management science. Springer, New York, pp 821–847
Papadimitriou CH, Yannakakis M (1991) Optimization, approximation, and complexity classes. J Comput Syst Sci 43(3):425–440
Rendl F (2012) Semidefinite relaxations for partitioning, assignment and ordering problems. 4OR 10(4):321–346
Rendl F, Rinaldi G, Wiegele A (2010) Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Math Program 121(2):307–335
Rinaldi G (2018) Rudy, a graph generator. https://www-user.tu-chemnitz.de/~helmberg/sdp_software.html. Accessed Feb 2018
Rodrigues de Sousa VJ, Anjos MF, Le Digabel S (2018) Computational study of valid inequalities for the maximum *k*-cut problem. Ann Oper Res 265(1):5–27. https://doi.org/10.1007/s10479-017-2448-9
Seidman SB (1983) Network structure and minimum degree. Soc Netw 5(3):269–287
Sherali HD, Fraticelli BMP (2002) Enhancing RLT relaxations via a new class of semidefinite cuts. J Glob Optim 22(1–4):233–261
Shor N Z (1998) Semidefinite programming bounds for extremal graph problems. Springer, Boston, pp 265–298
Wang G, Hijazi H (2017) Exploiting sparsity for the min k-partition problem. arXiv e-prints. arXiv:1709.00485
Wiegele A (2015) Biq mac library—binary quadratic and max cut library. http://biqmac.uni-klu.ac.at/biqmaclib.html. Accessed Feb 2018