UNIVERSITÉ DE MONTRÉAL

GLOBAL OPTIMIZATION OF THE MAXIMUM K-CUT PROBLEM

VILMAR JEFTÉ RODRIGUES DE SOUSA
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
MAI 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

GLOBAL OPTIMIZATION OF THE MAXIMUM K-CUT PROBLEM

présentée par : RODRIGUES DE SOUSA Vilmar Jefté
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

Mme LAHRICHI Nadia, Ph. D., présidente
M. ANJOS Miguel F., Ph. D., membre et directeur de recherche
M. LE DIGABEL Sébastien, Ph. D., membre et codirecteur de recherche
M. PERRIER Michel, Ph. D., membre
M. KRISLOCK Nathan, Ph. D., membre externe

DEDICATION

To the Almighty God
and to my family . . .

ACKNOWLEDGEMENTS

First and foremost I gratefully acknowledge the support and guidance of my supervisors Miguel F. Anjos and Sébastien Le Digabel. I'm infinitely thankful for your time, patience, ideas and corrections. Without you, the present study could not have been completed. I also want to thank Jacek Gondzio and all the members of my jury for their helpful insights.

To my family, *muito obrigado* for your unconditional love, support, and encouragement during the challenges. Especially, my wife, Elisama, thank you for being by my side throughout this process and for understanding my craziness.

To all my friends at GERAD: thank you, *merci*, *obrigado*, *gracias*, *shukraan*, *Mamnoun*... You're a great group of friends; the tennis, volley, soccer, lunches, and practice of presentations made these three years pass faster and you kept me sane. I would also like to thank all the staff of GERAD and Polytechnique Montréal for their daily assistance.

RÉSUMÉ

Le problème de la k-coupe maximale (max-k-cut) est un problème de partitionnement de graphes qui est un des représentatifs de la classe des problèmes combinatoires $\mathcal{NP}$-difficiles. Le max-k-cut peut être utilisé dans de nombreuses applications industrielles. L'objectif de ce problème est de partitionner l'ensemble des sommets en k parties de telle façon que le poids total des arrêtes coupées soit maximisé.

Les méthodes proposées dans la littérature pour résoudre le max-k-cut emploient, généralement, la programmation semidéfinie positive (SDP) associée. En comparaison avec les relaxations de la programmation linéaire (LP), les relaxations SDP sont plus fortes mais les temps de calcul sont plus élevés. Par conséquent, les méthodes basées sur la SDP ne peuvent pas résoudre de gros problèmes. Cette thèse introduit une méthode efficace de branchement et de résolution du problème max-k-cut en utilisant des relaxations SDP et LP renforcées.

Cette thèse présente trois approches pour améliorer les solutions du max-k-cut. La première approche se concentre sur l'identification des classes d'inégalités les plus pertinentes des relaxations de max-k-cut. Cette approche consiste en une étude expérimentale de quatre classes d'inégalités de la littérature : clique, general clique, wheel et bicycle wheel. Afin d'inclure ces inégalités dans les formulations, nous utilisons un algorithme de plan coupant (CPA) pour ajouter seulement les inégalités les plus importantes . Ainsi, nous avons conçu plusieurs procédures de séparation pour trouver les violations. Les résultats suggèrent que les inégalités de wheel sont les plus fortes. De plus, l'inclusion de ces inégalités dans le max-k-cut peut améliorer la borne de la SDP de plus de 2%.

La deuxième approche introduit les contraintes basées sur formulation SDP pour renforcer la relaxation LP. De plus, le CPA est amélioré en exploitant la technique de terminaison précoce d'une méthode de points intérieurs. Les résultats montrent que la relaxation LP avec les inégalités basées sur la SDP surpasse la relaxation SDP pour de nombreux cas, en particulier pour les instances avec un grand nombre de partitions $(k \geq 7)$.

La troisième approche étudie la méthode d'énumération implicite en se basant sur les résultats des dernières approches. On étudie quatre composantes de la méthode. Tout d'abord, nous présentons quatre méthodes heuristiques pour trouver des solutions réalisables : l'heuristique itérative d'agrégation, l'heuristique d'opérateur multiple, la recherche à voisinages variables, et la procédure de recherche aléatoire adaptative gloutonne. La deuxième procédure analyse les stratégies dichotomiques et polytomiques pour diviser un sous-problème. La troisième composante étudie cinq règles de branchement. Enfin, pour la sélection des nœuds de l'arbre de branchement, nous considérons

les stratégies suivantes : meilleur d'abord, profondeur d'abord, et largeur d'abord. Pour chaque stratégie, nous fournissons des tests pour différentes valeurs de k. Les résultats montrent que la méthode exacte proposée est capable de trouver de nombreuses solutions.

Chacune de ces trois approches a contribué à la conception d'une méthode efficace pour résoudre le problème du max-k-cut. De plus, les approches proposées peuvent être étendues pour résoudre des problèmes génériques d'optimisation en variables mixtes.

ABSTRACT

In graph theory, the maximum k-cut (max-k-cut) problem is a representative problem of the class of $\mathcal{NP}$-hard combinatorial optimization problems. It arises in many industrial applications and the objective of this problem is to partition vertices of a given graph into at most k partitions such that the total weight of the cut is maximized.

The methods proposed in the literature to optimally solve the max-k-cut employ, usually, the associated semidefinite programming (SDP) relaxation in a branch-and-bound framework. In comparison with the linear programming (LP) relaxation, the SDP relaxation is stronger but it suffers from high CPU times. Therefore, methods based on SDP cannot solve large problems. This thesis introduces an efficient branch-and-bound method to solve the max-k-cut problem by using tightened SDP and LP relaxations.

This thesis presents three approaches to improve the solutions of the problem. The first approach focuses on identifying relevant classes of inequalities to tighten the relaxations of the max-k-cut. This approach carries out an experimental study of four classes of inequalities from the literature: clique, general clique, wheel and bicycle wheel. In order to include these inequalities, we employ a cutting plane algorithm (CPA) to add only the most important inequalities in practice and we design several separation routines to find violations in a relaxed solution. Computational results suggest that the wheel inequalities are the strongest by far. Moreover, the inclusion of these inequalities in the max-k-cut improves the bound of the SDP formulation by more than 2%.

The second approach introduces the SDP-based constraints to strengthen the LP relaxation. Moreover, the CPA is improved by exploiting the early-termination technique of an interior-point method. Computational results show that the LP relaxation with the SDP-based inequalities outperforms the SDP relaxations for many instances, especially for a large number of partitions $(k \geq 7)$.

The third approach investigates the branch-and-bound method using both previous approaches. Four components of the branch-and-bound are considered. First, four heuristic methods are presented to find a feasible solution: the iterative clustering heuristic, the multiple operator heuristic, the variable neighborhood search, and the greedy randomized adaptive search procedure. The second procedure analyzes the dichotomic and polytomic strategies to split a subproblem. The third feature studies five branching rules. Finally, for the node selection, we consider the following strategies: best-first search, depth-first search, and breadth-first search. For each component, we provide computational tests for different values of k. Computational results show that the proposed exact method is able to uncover many solutions.

Each one of these three approaches contributed to the design of an efficient method to solve the max-k-cut problem. Moreover, the proposed approaches can be extended to solve generic mix-integer SDP problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

$G = (V, E)$	A graph
E	Set of edges in a graph
V	Set of vertices in a graph
$n = \|V\|$	Number of vertices in set V
$\|E\|$	Number of edges in set E
$\mathbb{R}$	Real set
S_n	Symmetric $n \times n$ Matrix
S_n^+	Symmetric Positive Semidefinite $n \times n$ Matrix

BeFS	Best First Search
BrFS	Breath First Search
CPA	Cutting Plane Algorithm
CPU	Central Processing Unit
DC	Dynamic Convexized Method
DFS	Depth First Search
GHz	Gigahertz
GMSW	Greedy Heuristic for Multiple Sizes of Wheel
GPP	Graph Partitioning Problem
GRASP	Greedy Randomized Adaptive Search Procedure
ICH	Iterative Clustering Heuristic
IPM	Interior Point Method
LP	Linear Programming
LSIP	Linear Semi-Infinite Programing
max-k-cut	Maximum k-Cut
MOH	Multiple Operator Heuristic
$\mathcal{NP}$	Nondeterministic Polynomial
PC	Personal Computer
SDP	Semidefinite Programming
SBC	Semidefinite Branch-and-Cut algorithm
SIP	Semi-Infinite Programing
VLSI	Very-Large-Scale-Integrated
VNS	Variable Neighborhood Search

CHAPTER 1 INTRODUCTION

The maximum k-cut (max-k-cut) problem is a graph partitioning problem that is a representative problem of the class of $\mathcal{NP}$-complete combinatorial optimization problems. Although it has several industrial applications, there is not a lot of computational studies proposed in the literature and the existing exact methods are very limited. Therefore, the main objective of this thesis is to investigate and propose new mathematical optimization techniques to design an efficient method to solve the problem to optimality.

1.1 Background

Operations research is a scientific approach that applies advanced analytical methods to support the decision making process and to optimize systems. In order to deal with a problem, it is typically necessary to create an abstraction of the real-world situation. A mathematical model is used as an attempt to describe the system.

A wide variety of real-world problems can conveniently be described by a graph, for example, the famous traveling salesman problem [69] and the seminal problem of the Königsberg bridges [20]. In graph theory, a graph $G = (V, E)$ is defined by a set of vertex V and a set of edges E.

Among all types of graphs, we consider the connected-weighted-simple graphs where V is a finite set and the edges are undirected with weight w_{ij} for all $(i, j) \in E$. Moreover, multiple edges or loops are disallowed.

Many operations can be performed on a graph, and one of them is a cut. In general, the graph is k-cut if the vertex set V is partitioned into at most k partitions in such a way that the intersection of two different partitions $P_i \subseteq V$ and $P_j \subseteq V$ is an empty set, $P_i \cap P_j = \emptyset$ for $i \neq j$ for each $i, j \in \{1, 2, \ldots, k\}$. In addition, all vertices inside the k partitions should also be in the vertex set $V = P_1 \cup P_2 \cdots \cup P_k$. The value of k-cut is the sum of weights of all edges joining different partitions. Calculating the optimal cut is a graph partitioning problem.

1.2 The maximum k-cut problem

The objective of the max-k-cut problem is to partition the vertex set of a graph into at most k partitions such that the total weight of the k-cut is maximized. The problem arises in many applications, e.g., VLSI circuit design, wireless communication [54] and sports team scheduling [60]. Some of these applications are reviewed in Chapter 2. Figure 1.1 illustrates the max-k-cut for a graph with

$|V| = 7$, $|E| = 11$ and $k = 3$.

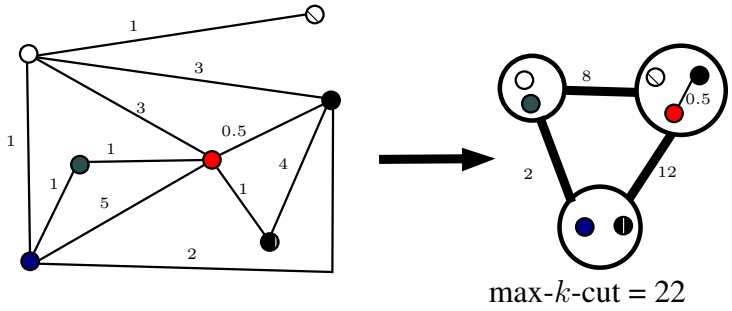

Figure 1.1 Illustration of a max-k-cut problem in a graph for $k = 3$.

For the max-k-cut, the special case with $k = 2$ is known as the max-cut problem. Due to the equivalence of this problem with unconstrained binary quadratic optimization, the max-cut has received most attention in the literature, see e.g. [51, 74]. In our investigation, we focus on problems with $k \geq 3$.

The max-k-cut is equivalent to the minimum k-partition problem [31]. In the minimum k-partition, the task is to minimize the total weight of the edges joining vertices in the same partition. The k-way equipartition problem is another related problem. In the k-way equipartition, we add some constraints that force the partitions to be the same (or almost) sizes.

In an unweighted graph, the max-k-cut can also be used to provide bounds for the chromatic number. The smallest number of colors needed to color the vertices of a graph in such a way that two adjacent vertices (joined by an edge) cannot have the same color.

1.3 Objective and outline

The objective of this thesis is to design an efficient solver for the max-k-cut problem. Hence, we investigate the following three objectives in next chapters. First, computational study of some proposed facet-defining inequalities to identify the most relevant in practice. Second, investigate ways of strengthen even more the mathematical model of the max-k-cut. Third, design an exact method to solve the problem to optimality.

In the first chapter of this thesis, we present the state-of-art of max-k-cut with applications, formulations, and methods proposed in the literature. Chapter 3 details the organization of the research. The Chapters 4, 5, and 6 are the core of the thesis, where we present the two articles and one chapter about the branch-and-bound algorithm where we present an exact method to obtain the global optimal solution for the max-k-cut problem. Finally, we conclude our work in Chapter 8.

CHAPTER 2 CRITICAL LITERATURE REVIEW

This chapter presents the state-of-art of the max-k-cut problem. Section 2.1 presents the most important applications. Section 2.2 review integer formulations and their relaxations. In Section 2.3, we present some valid inequalities that are proposed in the literature and Section 2.4 presents approximations, heuristics and exact methods designed to solve the max-k-cut problem.

2.1 Applications

Several industrial optimization problems are formulated as max-k-cut, some of the most important are described below.

Statistical physics. In [6], two classical applications of max-k-cut are presented: the statistical physics and very-large-scale-integrated (VLSI) circuit design. The ground states or minimum energy configuration of sping glasses are determined in statistical physics. The ground state can be found by minimizing the energy interaction between magnetic atoms in the sping glass. The energy between two atoms depends on their orientations and their distance. A simplified version of 1-dimensional direction is studied in [6, 52] where the problem is formulated as a max-cut problem.

(VLSI) circuit design. This problem arises in the phase of layer assignment in the construction of chip where wires belonging to different net may be crossed [6, 8]. Additional cost and often failure of the board happens when two wires cross in the same layer. Therefore, it is desirable to find a layer assignment that minimizes cost and failure. In [6], the wires are considered as the vertices, the crossing as the edges, and the layers as the partitions of a max-k-cut problem. This application can be generalized to modular design where a system is divided into smaller parts called modules in order to maximize independence between modules [29].

Team Realignment. A very common problem that is formulated as a k-equipartition problem is the team scheduling. In [60, 72] the authors study the realignment problem for a football league where the objective is to minimize the sum of intra-divisional distance. In this application, the cities are the vertices of the graph, the distances between cities correspond to the weight of edges and the number of division are the k partitions. An interesting application related to team scheduling is proposed in [28] to assign doctors for victims in emergency situations.

Wireless communication. In the case of wireless communications, the wireless network designs are the hardest problems. Most investigations are related to the problem of frequency assignment where the aim is to minimize interferences between devices see e.g. [22, 86]. However, in [54] the objective is to assign a group of exchange stations in such a way that as much traffic as possible can be routed inside these clusters.

Other applications. Many other cluster problems can be formulated as a max-k-cut problem such as parallel computing [44], floor planning [12], and others assignment problems like the ones studied in the capacitated version of max-k-cut [29]: the placement of television commercials and the placement of containers.

2.2 Formulations of the max-k-cut

The max-k-cut is known to be an $\mathcal{NP}$-hard combinatorial optimization [71]. In order to computationally solve the problem some integer programming formulations are proposed in the literature and their relaxations can be separated into two main groups: linear programming (LP) relaxation and semidefinite programming (SDP) relaxation.

2.2.1 Integer programming formulations

The max-k-cut has essentially three different integer formulations: One formulation with edge-only variables, another with both node-and-edge variables and a third with node-only variables. We present all of these formulations.

Edge-only formulation

A 0-1 edge formulation is proposed in [10], where the integer variable x_{ij} for each $i, j \in V$ is defined as:

$$x_{ij} = \begin{cases} 0 & \text{if edge } (i,j) \text{ is cut,} \\ 1 & \text{otherwise.} \end{cases}$$

Hence, from an edge perspective, the max-k-cut is formulated as:

$$
\begin{aligned}
(EDGE) \quad \max_{x} \quad & \sum_{i,j\in V, i<j} w_{ij}(1-x_{ij}) && && (2.1)\\
\text{s.t.} \quad & x_{ih}+x_{hj}-x_{ij}\leq 1 && \forall i,j,h\in V, && (2.2)\\
& \sum_{i,j\in Q, i<j} x_{ij}\geq 1 && \forall Q\subseteq V \text{ with } |Q|=k+1, && (2.3)\\
& x_{ij}\in\{0,1\} && \forall i,j\in V. && (2.4)
\end{aligned}
$$

where Constraint (2.2) and (2.3) are called triangle and clique inequalities, respectively. Triangle Inequalities (2.2) correspond to the logical conditions that if the edges (i,h) and (h,j) are not cut then edge (i,j) cannot be cut. The Constraints (2.3) impose that at least one edge in a clique with $k+1$ vertices cannot be cut.

Node-and-edge formulation

Also in [10], the authors present the node-and-edge formulation that has $|V|k+|E|$ variables and constraints. For each $v,i,j\in V$, for each $(i,j)\in E$ and $p\in\{1,\ldots,k\}$. The binary variables are:

$$
x_{ij}=\begin{cases}0 & \text{if edge } (i,j) \text{ is cut,}\\ 1 & \text{otherwise.}\end{cases}
\qquad
y_{vp}=\begin{cases}1 & \text{if vertex } v \text{ is in partition } p,\\ 0 & \text{otherwise.}\end{cases}
$$

Therefore, the node-and-edge integer formulations of max-k-cut is defined as:

$$
\begin{aligned}
(\textit{No-Ed}) \quad \max_{x} \quad & \sum_{(i,j)\in E, i<j} w_{ij}(1-x_{ij}) && && (2.5)\\
\text{s.t.} \quad & \sum_{p=1}^{k} y_{vp}=1 && \forall v\in V\,, && (2.6)\\
& x_{ij}\geq y_{ip}+y_{jp}-1 && \forall((i,j)\in E, p=1,\ldots,k), && (2.7)\\
& x_{ij}\leq y_{ip}-y_{jp}+1 && \forall((i,j)\in E, p=1,\ldots,k), && (2.8)\\
& x_{ij}\leq -y_{ip}+y_{jp}+1 && \forall((i,j)\in E, p=1,\ldots,k), && (2.9)\\
& x_{ij}\in\{0,1\} && \forall(i,j)\in E, && (2.10)\\
& y_{vp}\in\{0,1\} && \forall(v\in V, p\in\{1,\ldots,k\}). && (2.11)
\end{aligned}
$$

where Inequalities (2.8)-(2.9) can be removed when all edges weight are non-negative.

Node-only formulation

The last formulation is based on (vertex) node variables. In [25], the authors mention that just allowing the variables to be $x_i \in \{1, \ldots, k\}$ for each $i \in V$ does not generate useful integer formulations. Instead, if x_i is one of the k vectors $a_1, a_2, \ldots, a_k$ in $\mathbb{R}^{k-1}$ such that the dot product of this vector is:

$$a_i \cdot a_j = \frac{-1}{k-1}$$

it provides a convenient quadratic formulation. In Lemma 4 of [25], it is proved that the value $-1/(k-1)$ gives the best angle separation for k vectors. Then, for $x_i \in \{a_1, \ldots, a_k\}$ we have:

$$x_i x_j = \begin{cases} \frac{-1}{k-1} & \text{if } x_i \neq x_j \text{ (i.e, vertices } i \text{ and } j \text{ are in different partitions),} \\ 1 & \text{if } x_i = x_j. \end{cases}$$

Finally, we end up with the following quadratic formulation of the max-k-cut:

$$(NODE) \quad \max_{x} \quad \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - x_i x_j) \tag{2.12}$$

$$\text{s.t.} \quad x_i \in \{a_1, a_2, \ldots, a_k\} \qquad \forall i \in V. \tag{2.13}$$

2.2.2 Linear programming relaxation

Linear programs are the main field of operational research. In a linear programming formulation, all parameters of the model are known with certainty, all the variables are real, and all expressions (constraints and the objective function) are linear, i.e., a linear programming respects the following three assumptions: deterministic property, divisibility, and linearity [18].

The simplex algorithm [13] was, during many years, the unique method available to solve the LP. Although simplex is, in theory, a non-polynomial algorithm (it can make exponential steps to attain optimality), it is, in practice, very reliable and efficient. Nowadays, the polynomial interior point method (IPM) has shown to be a good alternative to the simplex method [34, 58]. In fact, in [34, 61] the authors show that the IPM enables the solution of many large-scale real-life problems and that IPM can exploit parallelism quite well.

Typically, a linear formulation is derivate from an integer programming formulation by relaxing the integrality constraint, i.e., in the LP we replace the constraint $x \in \{0, 1\}$ by its relaxed form $0 \leq x \leq 1$.

Edge-only relaxation. By relaxing the Integrality Constraint (2.4) in the $EDGE$ formulation by

$$0 \leq x_{ij} \leq 1 \;\; \forall i, j \in V. \tag{2.14}$$

we end up with the linear edge formulation of max-k-cut. This LP relaxation is used in the k-equipartition problems, see e.g. [60, 72] and in [85] the authors use this relaxation to exploit sparsity in the max-k-cut problem. They show that it is not necessary to add dummy edges (of zero weight) if the graph is chordal.

We can notice that this formulation suffers from huge number inequalities: in a complete graph it can have $3\binom{|V|}{3}$ triangle Inequalities (2.2) and $\binom{|V|}{k+1}$ clique Inequalities (2.3).

The edge-only formulation also suffers for not exploiting the structure of graphs like sparsity. In general, a chordal extension [39] of a sparse graph still adds a lot of dummy edge.

Node-and-edge relaxation. Replacing the Constraint (2.10) and (2.11) by their correspondent linear relaxations defines the linear node-and-edge formulation of the max-k-cut problem. This formulation is applied in the bounding procedure of [22] for a two-level graph partitioning problem. In [22] the authors show that this relaxation is very weak and it suffers from symmetry. This formulation is further improved by the so-called representative formulations [2] where a representative variable is added to break symmetry.

Both linear formulations presented above are investigated in Chapter 5 where we present some advantages and drawbacks. In summary, the LP formulation can rapidly be solved, but the bounds of LP are not strong [31], i.e., the linear solution is far from being an optimal solution.

2.2.3 Semidefinite programming relaxation

Semidefinite programming is one of the most active research areas in optimization [70]. In [73] the author points out that some $\mathcal{NP}$-complete combinatorial problems can benefit from semidefinite optimization because it is solved in polynomial time (with fixed precision), and it can handle quadratic constraints in its model.

SDP refers to the problem of optimizing a linear function over the intersection of the cone of positive semidefinite (*psd*) matrices with an affine set. A matrix M is symmetric ($M \in S_n$) if $M^T = M$ and, in particular, M is a square matrix with dimension n. The matrix $M \in S_n$ is *psd* ($M \in S_n^+, M \succeq 0$) if:

$$\mu^T M \mu \geq 0 \qquad \forall \mu \in \mathbb{R}^n. \tag{2.15}$$

It is known that $M \in S_n^+$ if and only if all principal submatrix of M are also psd. Moreover, a diagonal element m_{ii} from M is always among the elements of largest absolute value, i.e. for all $i \in \{1, \ldots, n\}$ the element $m_{ii} = \max\{|m_{ij}| : j \in \{1, \ldots, n\}\}$. Hence, if $m_{ii} = 0$ all elements in column i and row i of $M \in S_n^+$ are also 0. Others properties of *psd* are detailed in [40].

In SDP, the matrix variable $X \in S_n^+$ replaces the vector $x \in \mathbb{R}_+^n$ of variables in the LP formulation, i.e., the cone of the nonnegative orthant $x \geq 0$ in the LP is replaced by the cone of semidefinite matrices $X \succeq 0$. The linear programming is a special case of SDP where all non-diagonal elements of X are zero. Another similarity with LP is that SDP is convex, and duality theory of LP generalizes naturally to SDP, in particular, the characterization of optimality is ensured if there exist feasible points in the interior of primal and dual formulations [41].

Since the studies of [3, 66], the *interior point method* (IPM) is the most efficient method for solving semidefinite programming. The IPM iterates inside the SDP cone and it converges very fast by using Newton's method. However, the computation of each iteration is often too high for practical applications with many constraints.

Others, less accurate methods, are proposed to solve the SDP faster. The *spectral bundle method* proposed in [43] reformulate the SDP as an eigenvalue optimization problem, in this way, the bundle method avoid the expensive calculation of Cholesky factorization by using a first order method that depends only on the calculation of eigenvalues and eigenvectors. However, for the spectral bundle, there is no polynomial bound on the number of arithmetic operations.

Another way of solving the SDP is by transforming SDP in a *semi-infinite programming* problem [49, 50]. In summary, this approach replaces the cone $X \succeq 0$ by Constraint (2.15). In Chapter 5 we study in detail this approach to formulate a new family of inequalities for the linear edge-only relaxation of the max-k-cut problem.

Node-only relaxation. The SDP relaxation is based on the quadratic node-only formulation of the max-k-cut ($NODE$). To obtain a relaxation of the quadratic formulation we replace the constraint (2.13) to:

$$x_i \in B_n \quad \forall i \in V. \tag{2.16}$$

where B_n is the unit sphere in n dimensions. Thus, to avoid $x_i \cdot x_j = -1$ we add the constraint $x_i \cdot x_j \geq \frac{-1}{k-1}$. Thereby, replacing $x_i \cdot x_j$ for X_{ij} gives the SDP formulation proposed in [25]:

$$(SDP) \quad \max_{X} \quad \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - X_{ij}) \tag{2.17}$$

$$\text{s.t.} \quad X_{ii} = 1 \qquad \forall i \in V, \tag{2.18}$$

$$X_{ij} \geq \frac{-1}{k-1} \qquad \forall i, j \in V, i < j, \tag{2.19}$$

$$X \succeq 0. \tag{2.20}$$

Although this formulation has only a few constraints, the $\frac{n(n-1)}{2}$ Constraints (2.19) can impact a lot the computing time of the SDP. For instance, in [40] the author indicates that it is more efficient to start only with Constraint (2.18) and to separate $X_{ij} \geq \frac{-1}{k-1}$ successively in a cutting plane algorithm.

2.2.4 SDP versus LP relaxations

Both SDP and LP formulations have their strength and weakness. We see that IPM and simplex methods are well suitable to provide fast solutions for large linear problems, but in [31] the authors affirm that the LP relaxations are weak and it could result in the enumeration of all the solutions in a branch-and-bound method.

In [19] the authors study the relation between the LP and SDP polytopes. They show that SDP relaxations of max-k-cut violates at most $\sqrt{2} - 1$ of all triangle Constraint (2.2) and it violates at most $\frac{1}{2}$ of all clique constraints, in comparison with a violation of 1 for the LP relaxation.

Moreover, in [4] the authors affirm that computationally speaking, the strength of the SDP relaxations come with the cost of expensive running times.

2.3 Strengthening max-k-cut formulations

A relaxation can be tightened, without removing any feasible solution, by the addition of cutting planes (also called valid inequalities). In order to strength even more the LP and the SDP relaxations of the max-k-cut problem, some researchers have proposed some valid inequalities.

A cutting plane algorithm (CPA) is designed to solve a problem by, iteratively, adding only the most important inequalities. Therefore, this section presents some valid inequalities and introduces the CPA.

2.3.1 Valid inequalities

Typically, the triangle (2.2) and clique (2.3) inequalities of the edge-only formulation are also used to strengthen the SDP formulations. For example, in [81], it is shown that triangle inequalities are stronger than clique inequalities in the SDP relaxation.

In [9], the authors propose several valid and facet-defining inequalities for k-partition problem polytopes. Particularly, they use the edge-only linear relaxation to demonstrate the validity of the following inequalities:

- the General clique inequalities are generalizations of the Clique Inequalities 2.3, and they are valid if it has more than k vertices and it is facet-defining if the size of this inequality (number of vertices in it) is not a module of k,
- the wheel inequalities are facet-defining if the main cycle in the structure of this inequality is odd and if $k \geq 4$, and
- the bicycle wheel inequalities are facet-defining for $k \geq 4$ if the cycle is odd.

Several valid and facets-defining inequalities for the max-cut problem is studied in [16]. The results of [16] are generalized for the max-k-cut problem in [10] where some sufficient condition for hypermetric, cycle and anti-web inequalities are proposed.

In [21], a new family of inequality for the node-and-edge relaxation called projected clique inequalities is obtained by the projection of edge-only formulation to node-and-edge formulation. Computational results show that the new inequalities are of practical use in the case of large sparse graphs.

In Chapter 4 we review in more detail some of these valid inequalities, especially the ones proposed in [9].

2.3.2 Cutting plane algorithm

A CPA is designed to find optimal solutions of different types of problems or to tighten bounds [69, 87]. The CPA is a standard technique applied to problems when the number of inequalities is too large to be explicitly represented in the model [36]. For the max-k-cut, the CPA is, usually, used to separate triangle and clique inequalities see e.g. [4, 31].

In summary, the CPA starts with a basic relaxation. At each iteration, it solves the problem, then some separation routines search for violated cut planes (inequalities) and the most violated constraints are added to the relaxation. Then, some unimportant constraints (with large slack variables)

can also be removed. Finally, if some cut planes were added a new iteration is started, otherwise, we stop the CPA.

Separation routines are exact or heuristic methods applied to search for violated inequalities in a relaxed solution. For the max-k-cut problem, in [4, 31] a greedy heuristic is applied to search for clique inequalities and an exact enumeration to search for triangle inequalities. In [15] the authors show that the separation of odd cycles can be done in polynomial time for wheel and bicycle wheel inequalities. For the projected inequalities of [21], the author proposes some exact and heuristic methods for the searching.

A variation of the CPA that uses IPM solvers is applied to the `PDCGM` solver [35] and in [58]. The technique employed is called the *early-termination*, where the IPM is stopped before it attains optimality to search for cut planes that are violated. Normally, the inequalities are stronger when the early-termination is applied [61].

A formal and more detailed description of CPA is provided in Chapters 4 and 5.

2.4 Methods to solve the max-k-cut problem

In this section, we present some approximations, heuristics and exact methods proposed in the literature to solve the max-k-cut problem.

2.4.1 Approximations methods

In [33] the famous 0.878-approximation algorithms for the max-cut problem is proposed. The main idea of this approximation is to solve an SDP formulation and use a randomized rounding heuristic to obtain a good solution. In this approximation, a random hyperplane that passes through the origin is chosen, and we partition a vertex $v_i \in V$ according to the side of the hyperplane that it falls.

In [25] and [47] the authors extend the approach of [33] to the max-k-cut. In [25], it is shown that there exist a sequence of constants α_k such that:

$$H(w(\mathcal{P}_k)) \geq \alpha_k w(\mathcal{P}_k^*)$$

where $\mathcal{P}_k^*$ denotes the optimal solution, $w(\mathcal{P}_k^*)$ is the value of $\mathcal{P}_k^*$, and H denotes the expected value of their approximations. In [25, Theorem 1], the authors show that the constants α_k satisfies :

- $\alpha_k > \frac{k-1}{k}$, and

- $\alpha_2 \geq 0.878567$, $\alpha_3 \geq 0.800217$, $\alpha_4 \geq 0.850304$, $\alpha_5 \geq 0.874243$, $\alpha_{10} \geq 0.926642$, and $\alpha_{100} \geq 0.990625$.

For small values of k, the SDP approximation proposed in [14] gives better results than [25]. For example, if $k = 3$ they improved the bound to 0.832718 and for $k = 4$ to 0.857487. In [14] the rounding procedure of [25] is applied in a lifted matrix of the SDP solutions.

2.4.2 Heuristic methods

As opposed to exact and approximations method where the first guarantees optimality and the second a solution that cannot be worse than an α approximation, the heuristic and meta-heuristic methods attempt to yield good solutions with any guarantee. Typically, the heuristic is much faster than exact methods, and the heuristic solutions are normally better than the ones of approximation methods.

For the max-cut ($k = 2$) problem, many heuristic algorithms have been proposed (see e.g., [89, 48]). For the max-k-cut, on the other hand, there are much fewer methods proposed. Therefore, we present all heuristic that we are aware of.

An iterative clustering heuristic (ICH) was proposed in [31] to finds feasible solutions from an SDP solution. Their computational results show that ICH provides better solutions than those obtained by [25]. The ICH works by building subgraphs based on the information from an SDP solution (X^*).

In [90] it is introduced a multi-start-type algorithm called dynamic convexized (DC) method where a local search algorithm is applied to a dynamically updated auxiliary function.

In [56] the authors present a multiple operator heuristic (MOH). The MOH is an iterative method that applies five search operators in three search phases. They show that their MOH provides better bounds in less time than DC method in 90% of their tests.

A more detailed review of some of these heuristic methods is presented in Chapter 6, where we also compare their efficiency.

2.4.3 Exact methods

Exact methods find optimal solutions for the max-k-cut problem. Most of the methods are based on the branch-and-bound algorithm. Both algorithms use a tree search strategy to implicitly enumerate all possible solutions. The difference between these methods is that the branch-and-cut uses a cutting plane algorithm in all nodes of the three search. We investigate in details the branch-and-bound framework in Chapter 5.

In [60, 59] a branch-and-cut algorithm based on edge-only relaxation is applied to the k-way equipartition problem to solve the realignment problem of a football league where $k = 8$ and $|V| = 32$. The branch-and-cut uses an LP formulation that is reinforced by triangle and clique inequalities. For problems with 100 to 500 vertices, they have found a percentage gap inferior to 2.5% for $k = 4$.

A branch-and-bound algorithm with cutting plane algorithm in the root node (called cut-and-branch algorithm) is applied in [22] for the two-level graph partitioning problem. The linear relaxation is based on the node-and-edge formulations of max-k-cut. The input graph is simplified by a preprocessing stage, and the LP relaxation is tightened by the triangle, clique, and general clique based inequalities. Computational results found the optimal solution for problems with $|V| = 100$ and $k \in \{2, 3, 4\}$ for sparse graphs.

A branch-and-bound method based on the edge-only formulation of the max-k-cut is investigated in [85], where it is shown that a chordal graph can be formulated with only $|E|$ variables instead of $\frac{n(n-1)}{2}$. In [85], sparse problems with $|V| = 200$ and $k \in \{3, 4\}$ are solved in few seconds.

In [31] the semidefinite relaxation is used in a branch-and-cut algorithm called SBC. The SBC uses triangle and clique inequalities, and the SDP is solved by the spectral bundle method, moreover, the ICH heuristic is applied to get feasible solutions. The SBC algorithm computes optimal solutions for dense graphs with 60 vertices, and for sparse graphs with 100.

The `BundleBC` algorithm proposed in [4] is an improvement of the SBC method. `BundleBC` extends the ideas of the `Biq Mac` solver [74] to max-k-cut. The importance of separating clique inequalities with triangle inequalities is also investigated in `BundleBC`. Feasible solutions are computed via the approximation algorithm proposed in [25]. Computational results show that the use of the clique inequalities reduces the number of subproblems analyzed in the branch-and-bound tree, and the `BundleBC` is shown to be faster than SBC. The `BundleBC` is able to solve some dense instances with 70 nodes and sparse with 100, for $k \in \{3, 5, 7\}$.

2.5 Additional reviews

To close this chapter we refer to the reader the additional literature review on the cutting plane algorithm in Chapter 4. In Chapter 5, we review the semi-infinite formulations especially for the linear formulation of SDP. Finally, in Chapter 6, we review in detail all the components of the branch-and-bound method.

CHAPTER 3 ORGANIZATION OF THE THESIS

In this chapter, we present the organization of the research done as part of this thesis. The motivations and the results initially hoped for are also explained.

In the previous chapter, we presented the state-of-art of the max-k-cut literature. We observe that some methods are proposed to exactly solve the max-k-cut. In particular, the SBC [31] and the `BundleBC` [4] solvers use the SDP formulations of max-k-cut to obtain upper bounds in a branch-and-cut framework. However, both methods are unable to solve instances with more than 70 vertices.

Several studies show that, on one hand, the SDP yields good bounds, but, on the other hand, it suffers from expensive running times [4, 19]. Based on this, we aim to design a formulation that provides a trade-off between quality and time of solution, and we apply this formulation within an efficient branch-and-bound algorithm.

In Chapters 4 and 5, we report our articles [76] and [77]. In Chapter 6, we present an investigation of an exact method for the max-k-cut. The chapters are presented in a chronological way. Each part of the thesis focuses on improving solutions to the max-k-cut problem. We present the steps performed at each one of the three parts of this thesis.

The first article [76] was accepted to be published on *Annals of Operations Research* in March 2017, and is reported in Chapter 4. This article investigates some families of inequalities to strengthen the max-k-cut relaxation. These inequalities studied were proposed by [9] and they are known to be facet-defining. Specifically, we considered the five following classes of cutting planes: the Triangle, Clique, General clique, Wheel and Bicycle wheel inequalities. Due to the fact that each family of inequality can have a huge number of rows, we design a cutting plane algorithm (CPA) to add only the most important inequalities. Therefore, for each class, we study several separation methods. For example, for the wheel and bicycle wheel inequalities, we study four separation methods for finding violated inequalities in an iteration of CPA. In order to investigate the impact of each class, we create ten combination of methods that alternate the activation and deactivation of each family. Moreover, we study dense and sparse graphs, and we compare our combinations using different benchmarks. Hence, we intend to identify the most important classes of inequalities to be prioritized in the max-k-cut formulation.

The second article [77] submitted in April 2018 on *EURO Journal on Computational Optimization* is presented in Chapter 5. We proposed a family of inequalities to strengthen the LP relaxation based on the SDP formulation. These inequalities have infinity rows and we design an exact sepa-

ration routine based on eigenvalue of the relaxed solution. Moreover, we employ a modified CPA to find deeper cuts. In order to compare the SDP with the new LP formulation we test both methods for several instances.

Chapter 6 focuses on designing an efficient exact method to obtain global solutions of the max-k-cut problem. This chapter is the conclusion of our study where we include ideas of the two previous works [76, 77] in a branch-and-bound algorithm. We investigate five of the most important components of the branch-and-bound method: upper-bound, feasibility, splitting, branching, and node selection. For each procedure, we computationally study several strategies. For example, in the branching rule procedure we study five different rules for selecting a variable in a node of the branch-and-bound tree. Moreover, we show the evolution of the LP formulations (from [76] to [77]) and we also compare both LP and SDP formulations in the branch-and-bound scheme for a variety of instances. This way, an efficient exact method is designed.

CHAPTER 4 ARTICLE 1: COMPUTATIONAL STUDY OF VALID INEQUALITIES FOR THE MAXIMUM K-CUT PROBLEM

Authors: Vilmar J. Rodrigues de Sousa, Miguel F. Anjos, Sébastien Le Digabel.
Accepted for publication: Annals of Operations Research

Abstract We consider the maximum k-cut problem that consists in partitioning the vertex set of a graph into k subsets such that the sum of the weights of edges joining vertices in different subsets is maximized. We focus on identifying effective classes of inequalities to tighten the semidefinite programming relaxation. We carry out an experimental study of four classes of inequalities from the literature: clique, general clique, wheel and bicycle wheel. We considered 10 combinations of these classes and tested them on both dense and sparse instances for $k \in \{3, 4, 5, 7\}$. Our computational results suggest that the bicycle wheel and wheel are the strongest inequalities for $k = 3$, and that for $k \in \{4, 5, 7\}$ the wheel inequalities are the strongest by far. Furthermore, we observe an improvement in the performance for all choices of k when both bicycle wheel and wheel are used, at the cost of 72% more CPU time on average when compared with using only one of them.

Keywords. Maximum k-Cut, Graph Partitioning, Semidefinite Programming, Computational Study.

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

4.1 Introduction

Graph partitioning problems (GPPs) are an important class of combinatorial optimization. There are various types of GPP based on the number of partitions allowed, the objective function, the types of edge weights, and the possible presence of additional constraints such as restrictions on the number of vertices allowed in each partition. This work considers the GPP known as the maximum k-cut (max-k-cut) problem: Given a connected graph $G = (V, E)$ with edge weights w_{ij} for all $(i, j) \in E$, partition the vertex set V into at most k subsets so as to maximize the total weight of cut edges, i.e., the edges with end points in two different subsets. This problem is sometimes also called minimum k-partition problem [31]. The special case of max-k-cut with $k = 2$ is known as the max-cut problem and has received most attention in the literature, see e.g. [6, 16, 33, 51, 70, 74].

Graph partitioning and max-k-cut problems have myriad applications as VLSI layout design [6], sports team scheduling [60], statistical physics [52], placement of television commercials [29], network planning [19], and floorplanning [12].

The unweighted version of max-k-cut is known to be $\mathcal{NP}$-complete [71]. Hence, researchers have proposed heuristics [11, 56], approximation algorithms [11, 25], and exact methods [4, 31] for solving the max-k-cut. In particular the approximation method proposed in [25] extends the famous semidefinite programming (SDP) results for max-cut in [33] to max-k-cut. The authors of [14] subsequently improved the approximation guarantees for small values of k.

Ghaddar et al. [31] propose the SBC algorithm. It is a branch-and-cut algorithm for the minimum k-partition problem based on the SDP relaxation of max-k-cut. They found experimentally that for $k > 2$ the SDP relaxation yields much stronger bounds than the LP relaxation, for both sparse and dense instances. This algorithm was improved in [4] which introduced the bundleBC algorithm to solve max-k-cut for large instances by combining the approach of [31] with the design principles of `Biq Mac` [74]. The results in [4] confirm that, computationally speaking, the SDP relaxations often yield stronger bounds than LP relaxations.

The matrix lifting SDP relaxation of the GPP, and the use the triangle and clique inequalities to strengthen it, were studied in [81, 83]. In particular [81] shows that this relaxation is equivalent to the relaxation in [25] and is as competitive as any other known SDP bound for the GPP. The results also show that the triangle inequalities are stronger than the clique (independent set) inequalities.

Using a similar approach as [81], the authors in [84] propose several new bounds for max-k-cut from the maximum eigenvalue of the Laplacian matrix of G. Moreover, [84] shows that certain perturbations in the diagonal of the Laplacian matrix can lead to stronger bounds. In [67] the author proposes a more robust bound than the one proposed in [84] by using the smallest eigenvalue of the adjacency matrix of G.

In [21] the authors project the polytopes associated with the formulation with only edge variables into a suitable subspace to obtain the polytopes of the formulation that has both node and edge variables. They then derive new valid inequalities and a new semidefinite programming relaxation for the max-k-cut problem. The authors suggest that the new inequalities may be of practical use only in the case of large sparse graphs.

The objective of this work is to carry out a computational study of valid inequalities that strengthen the SDP relaxation for the max-k-cut problem, and to identify the strongest ones in practice. Valid and facet-inducing inequalities for k-partition were studied in [9, 10]. We focus our analysis on the following inequalities: Triangle, Clique, General clique, Wheel and Bicycle wheel. To the best of our knowledge, no research has specifically focused on comparing the strength of the range of

inequalities considered here.

This paper is organized as follows. Sections 5.1.1 and 4.1.2 review the SDP formulation and relaxation of the max-k-cut problem. Section 5.2 presents the classes of inequalities and discusses the separation methods used. Section 4.3 describes the cutting plane algorithm used, and Section 5.4 presents and discusses the test results. The conclusions from this study are provided in Section 6.7.

4.1.1 Problem formulation

We assume without loss of generality that the graph $G = (V, E)$ is a complete graph because missing edges can be added with a corresponding weight of zero. Let $k \geq 2$ and define the matrix variable $X = (X_{ij})$, $i, j \in V$, as:

$$X_{ij} = \begin{cases} \frac{-1}{k-1} & \text{if vertices } i \text{ and } j \text{ are in different partitions of the } k\text{-cut of } G, \\ 1 & \text{otherwise.} \end{cases}$$

The max-k-cut problem on G can be expressed as:

$$\max_{X} \quad \sum_{i,j \in V, i<j} w_{ij} \frac{(k-1)(1-X_{ij})}{k} \tag{4.1}$$

$$\text{s.t.} \quad X_{ii} = 1 \qquad \forall i \in V, \tag{4.2}$$

$$X_{ij} \in \left\{ \frac{-1}{k-1}, 1 \right\} \qquad \forall i, j \in V, i < j, \tag{4.3}$$

$$X \succeq 0. \tag{4.4}$$

We refer to this formulation as (MkP). This formulation was first proposed by [25]. As mentioned in [25, Lemma 4], the value $-1/(k-1)$ gives the best angle separation for k vectors.

4.1.2 Semidefinite relaxation

Replacing Constraint (4.3) by $\frac{-1}{k-1} \leq X_{ij} \leq 1$ defines the SDP relaxation. However, the constraint $X_{ij} \leq 1$ can be removed since it is enforced implicitly by the constraints $X_{ii} = 1$ and $X \succeq 0$.

Therefore the SDP relaxation, denoted (RMkP), is:

$$\max_{X} \quad \sum_{i,j\in V, i<j} w_{ij} \frac{(k-1)(1-X_{ij})}{k} \tag{4.5}$$

$$\text{s.t.} \quad X_{ii} = 1 \qquad \forall i \in V\,, \tag{4.6}$$

$$X_{ij} \geq \frac{-1}{k-1} \qquad \forall i,j \in V, i<j, \tag{4.7}$$

$$X \succeq 0. \tag{4.8}$$

4.2 Formulation and separation of inequalities

The SDP relaxation (RMkP) can be tightened by adding valid inequalities, i.e., cutting planes that are satisfied for all positive semidefinite matrices that are feasible for problem (MkP) but are violated by solution X' of the (current) SDP relaxation. In this section we present the classes of inequalities from [9, 10] that we consider adding to tighten the SDP relaxation. For each class, we describe its properties, its formulation in terms of the SDP variable X, and the separation routine used.

The first two classes of inequalities (Triangle and Clique) were used in the SBC branch-and-cut method [31] and in the matrix lifting approach [81, 83]. We are unaware of the other classes having been computationally tested or used.

4.2.1 Triangle inequalities

Triangle inequalities are based on the observation that if vertices i and h are in the same partition, and vertices h and j in the same partition, then vertices i and j necessarily have to be in the same partition. The resulting $3\binom{|V|}{3}$ triangle inequalities are:

$$X_{ih} + X_{hj} - X_{ij} \leq 1 \;\; \forall i,j,h \in V. \tag{4.9}$$

Separation of triangle inequalities

As pointed out in [4], the enumeration of all triangle inequalities is computationally inexpensive, even for large instances. Therefore, a complete enumeration is done to find triangles in the graph G that violate (4.9). The tolerance used to detect a violation is 10^{-5}.

4.2.2 Clique inequalities

The $\begin{pmatrix} |V| \\ k+1 \end{pmatrix}$ clique inequalities ensure that for every subset of $k+1$ vertices, at least two of the vertices belong to the same partition. We express them as:

$$\sum_{i,j \in Q, i<j} X_{ij} \geq -\frac{k}{2} \ \forall Q \subseteq V \text{ with } |Q| = k+1. \tag{4.10}$$

Separation of clique inequalities

The exact separation of clique is $\mathcal{NP}$-hard in general, and the complete enumeration is intractable even for small values of k [31]. We use the heuristic described in [31] to find a violated clique inequality for a solution X' of the SDP relaxation. This heuristic is described here as Algorithm 1, and returns up to $|V|$ clique inequalities. We use a tolerance $\varepsilon = 10^{-5}$.

for *all* $j \in V$ **do**
  $C = \{j\}$;
  while $|C| \leq k$ **do**
    Select $u \in V \backslash C$ such that $u \in \left\{ \arg\min_{v} \sum_{i \in C} X'_{i,v} \mid v \in V \backslash C \right\}$;
    $C = C \cup \{u\}$;
  end
  if $\left(\sum_{i,z, \in C, i<z} X'_{iz} < -\frac{k}{2} - \varepsilon \right)$ **then**
    $Q = C$;
    Construct an inequality of type (4.10);
  end
end

Algorithm 1: Heuristic to find violated clique inequalities.

4.2.3 General clique inequalities

The General clique inequalities in [9] are a generalization of the Clique inequality (4.10).

Let $Q = (V(Q), E(Q))$ be a clique of size $p := tk + q$ where $t \geq 1$ and $0 \leq q < k$, i.e., t is the largest integer such that $tk \leq p$. In terms of the SDP variable X, the general clique inequalities have the form:

$$\sum_{i,j \in Q} X_{ij} \geq \frac{k}{k-1} \left[z + w \left(\frac{k-1}{k} - 1 \right) \right] \ \forall Q \subseteq V, |Q| = p. \tag{4.11}$$

where $z = \frac{1}{2}t(t-1)(k-q) + \frac{1}{2}t(t+1)q$ and $w = \frac{p(p-1)}{2}$. It is proved in [9] that (4.11) is valid for the polytope of the max-k-cut problem if $p \geq k$, i.e., $t \geq 1$. Furthermore, it is facet-defining if and only if $1 \leq q \leq k-1$, $t \geq 1$, i.e., p is not an integer multiple of k.

Separation of general clique inequalities

Because the clique inequalities are included in general clique, it is clear that enumeration is also impracticable for general clique. Thus, we propose a heuristic to find the most violated general clique inequalities for X'.

Our heuristic is a small modification of the one used to separate the clique inequalities (Algorithm 1). Specifically:

- The argument in the `WHILE` condition becomes $|C| \leq p-1$.
- The inequality in the `IF` condition is replaced by

$$\left(w(j) < \frac{k}{k-1}\left[z + w\left(\frac{k-1}{k} - 1 \right) \right] - \varepsilon \right).$$

The parameter p may vary with the number of partitions k. After preliminary tests using $p \in \{5, 7, 8, 10, 11, 13\}$ for $k = 3$, $p \in \{6, 7, 9, 11, 13, 14\}$ for $k = 4$, $p \in \{7, 9, 11, 13, 16, 18\}$ for $k = 5$, and $p \in \{9, 11, 13, 16, 18, 20\}$ for $k = 7$, we settled on with the following choices: $p = 5$ for $k = 3$; $p = 6$ for $k = 4$; $p = 7$ for $k = 5$; and $p = 11$ for $k = 7$.

4.2.4 Wheel inequalities

In [9], the q-wheel is defined as $W_q = (V_q, E_q)$, where

- $V_q = \{v, v_i | i = 1, 2, ..., q\}$
- $E_q = \bar{\mathrm{E}} \cup \hat{\mathrm{E}}$, where $\bar{\mathrm{E}} = \{(v, v_i), i = 1, 2, ..., q\}$ and $\hat{\mathrm{E}} = \{(v_i, v_{i+1}), i = 1, 2, ..., q\}$.

where $v \in V$, and all indices greater than q are modulo q. For $q = 6$, W_6 is illustrated in Figure 4.1.

Let W_q be a subgraph of G for $q \geq 3$. In terms of the SDP variable X, the wheel inequality is defined as:

$$\sum_{i=1}^{q} X_{v,v_i} - \sum_{i=1}^{q} X_{v_i,v_{i+1}} \leq \left\lfloor \frac{1}{2}q \right\rfloor \frac{k}{k-1}. \tag{4.12}$$

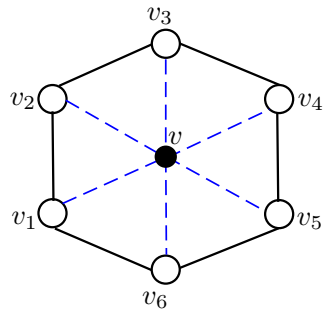

Figure 4.1 Example of wheel with $q = 6$.

The idea of this inequality is that vertex v can be in the same partition as vertices v_i and v_{i+1} if edge $e = (v_i, v_{i+1})$ is not cut. It is proved in [9] that (4.12) is a valid inequality for max-k-cut, and that it defines a facet if q is odd and $k \geq 4$.

Separation of wheel inequalities

By Inequality (4.12), a q-wheel is violated by X' if:

$$\sum_{i=1}^{q} X'_{v,v_i} - \sum_{i=1}^{q} X'_{v_i,v_{i+1}} - \left(\left\lfloor \frac{1}{2} q \right\rfloor \frac{k}{k-1} \right) > 0. \tag{4.13}$$

For each vertex $v \in V$, we wish to maximize the left-hand side of Inequality (4.13) to find the most violated wheel inequality.

In [15] it is shown that the odd wheel inequalities can be separated in polynomial-time. We also considered four heuristics, for a total of five methods tested:

- Exact algorithm [15].
- The GRASP meta-heuristic [23].
- A genetic algorithm (GA) from [80].
- The K-step greedy lookahead [78] with K=2.
- Our own greedy heuristic for multiple sizes of wheel (GMSW), described in Algorithm 2.

The GMSW heuristic achieved, on average, the best ratio between CPU time and performance for all tests. For instance, the performance ratio (see Section 5.4.3) was less than 0.03% higher than the exact method. In addition, its CPU time was in average 45% less. In GSMW, the parameter

for *all* $j \in V$ **do**
  $i = 1$;
  $v_i = j$;
  $W = \{v_i\}$;
  while $i \leq \mathrm{NbIte}$ **do**
    $q = |W|$;
    $z = 0$;
    if $q \geq 3$ **then**
      Select $v \in V \backslash W$ s.t.:
      $z = \max_v \sum_{i=1}^{q} X'_{v,v_i} - \sum_{i=1}^{q} X'_{v_i,v_{i+1}} - \left(\lfloor \frac{1}{2} q \rfloor \frac{k}{k-1} \right)$;
    end
    if $z > 0$ **then**
      Build Inequality (4.12) with W and v;
      $j = j + 1$;
      $i = \mathrm{NbIte} + 1$;
    else
      Select $u \in V \backslash W$ such that $u \in \{\arg \min_v X'_{v_q,v} + X'_{v,v_1} \mid v \in V \backslash W\}$;
      $i = i + 1$;
      $v_i = \{u\}$;
      $W = W \cup \{v_i\}$;
    end
  end
end

Algorithm 2: The GMSW heuristic to find violated wheel inequalities.

$\mathrm{NbIte} \leq |V| - 1$ is the maximum size allowed for the wheel. We obtained the best results using $\mathrm{NbIte} = 2k$ when $2k \leq |V| - 1$.

4.2.5 Bicycle wheel inequalities

The fourth class of inequalities corresponds to a q-bicycle wheel $BW_q = (V_q, E_q)$ subgraph of G, where:

$$
\begin{aligned}
&V_q = \{u_1, u_2\} \cup \{v_i, i = 1, 2, ..., q\}, && E_q = E_1 \cup E_2 \cup \{\bar{\mathrm{e}}\}\ , \\
&E_1 = \{(u_i, v_j), i = 1, 2, j = 1..., q\}, && E_2 = \{(v_j, v_{j+1}), j = 1..., q\}\ , \\
&\bar{\mathrm{e}} = \{(u_1, u_2)\}, &&
\end{aligned}
$$

where vertices $v, u \in V$. For $q = 5$, BW_5 is illustrated in Figure 4.2.

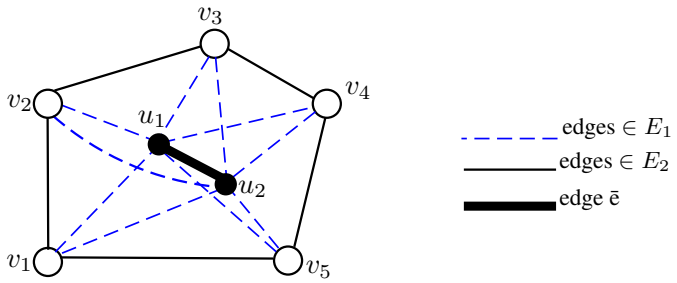

Figure 4.2 Example of bicycle wheel with $q = 5$.

For BW_q with $q \geq 3$, the corresponding bicycle wheel inequality is defined as:

$$\sum_{i=1}^{2}\sum_{j=1}^{q} X_{u_i,v_j} - \sum_{j=1}^{q} X_{v_j,v_{j+1}} - X_{u_1,u_2} \leq \frac{k}{k-1}\left[2\lfloor\frac{1}{2}q\rfloor - q + 1\right] + q - 1. \tag{4.14}$$

where $v \in V$ and all indices greater than q are modulo q. It is proved in [9] that the bicycle wheel inequality is a valid inequality to (MSkP), and for $k \geq 4$ it induces a facet if q is odd and BW_q is an induced subgraph of G.

The idea behind bicycle wheel inequalities is that at least one edge $e \in E_1$ must be cut for each edge from $\bar{e}$ or E_2 that is cut.

Separation of bicycle wheel inequalities

In [15] is pointed out that the separation for the odd bicycle wheel inequalities (4.14) can also be done in polynomial time, by adapting the approach in [30].

As we did for wheel separation, we tested also some heuristic method to find violated inequalities of the form (4.14) for a given X'.

The basic idea of those heuristics is:

- Search for a cycle that minimizes the sum $\left(\sum_{j=1}^{j\leq q} X_{v_j,v_{j+1}}\right)$.
- Find the best points u_1 and u_2 that maximize $\left(\sum_{j,i=1}^{i=1,2,j\leq q} X_{u_i,v_j}\right)$ and $(-X_{u_1,u_2})$.

For finding good cycles in the first step of this heuristic, we tested the same four heuristics mentioned in Section 4.2.4. Our tests showed that the GA from [80] with $q = 3$ obtained the best results in 95% of the cases. Moreover, this heuristic had the same performance as the exact method for all tests, and its CPU time was 95% lower.

The GA has the following features:

- Size of population is 100.
- Half of the initial population was constructed with the GRASP and other half was randomly selected.
- In crossover process the parents with higher fitness are favored.
- Offspring genes can accept some slight mutations.

We keep the best cycle C_w found in the final population after 100 iterations of the GA. Using this cycle, the main heuristic then checks all the combinations of $u_1, u_2 \in V \backslash C_w$ to find the one that maximizes the right-hand side of (4.14).

4.3 Cutting plane algorithm

This section presents the cutting plane algorithm (CPA) used for our computational tests. This algorithm is similar to the one described in [31]. The basic idea is to start by solving the SDP problem (RMkP), then check for violated inequalities, add them to the SDP problem, and resolve to obtain a better upper bound on the global optimal value of the max-k-cut instance.

The pseudo-code of the CPA is given in Algorithm 3. We use the following notation: i is the iterations counter, F^i is the set of inequalities in iteration i, X^i is the optimal solution of the SDP relaxation at iteration i, and $S(X^i)$ is the set of valid inequalities for (MkP) that are violated by X^i.

Initializations: $i = 0$, $F^0 = \emptyset$.
repeat
  X^i = Solve (RMkP) with F^i;
  $S(X^i)$ = set of violated inequalities for X^i ;
  if *Constraint* (4.3) *is satisfied* **then**
    STOP ; /* X^i is also optimal for (MkP) */
  end
  if $S(X^i) == \emptyset$ **then**
    STOP ; /* no violated inequalities */
  else
    $F^{i+1} = F^i \cup S(X^i)$;
    $i = i + 1$;
  end
until STOP;

Algorithm 3: Cutting plane algorithm.

The set $S(X^i)$ of inequalities is obtained by the separation routines of triangle, clique, general clique, wheel and bicycle wheel inequalities described in Section 5.2. We say that a class of inequalities is *active* if its separation routine is applied to find violations in the relaxed solution at each iteration of the CPA.

The size of the set $S(X^i)$ is controlled by the parameter NbIneq: $|S(X^i)| \leq \text{NbIneq}$ at each iteration of the CPA. After some tests with $\text{NbIneq} \in \{100, 300, 500, 1000, \infty\}$, the CPA with $\text{NbIneq} = 300$ was found to be the least expensive in terms of CPU time and CPA iterations for a variety of graphs and choices of k. This limitation in the CPA imposes that just the most violated inequalities of the set of activated families are added at each iteration.

4.4 Computational tests

The SDP relaxations of the max-k-cut were solved using the `mosek` solver [5] on a linux PC with two Intel(R) Xeon(R) 3.07 GHz processors.

Tests were performed for each value of $k \in \{4, 3, 5, 7\}$ on 147 test instances. Our instances were taken from the `Biq Mac` Library [88] or generated using `rudy` [75].

Section 4.4.1 introduces the combinations of inequalities tested, and the instances used. Section 5.4.3 explains the benchmarking methodology used for analyzing the different algorithms, and Section 5.4.4 reports the computational results. Finally, a summary of our results is given in Section 4.4.4.

4.4.1 Combinations of inequalities and test instances

This section defines the ten combinations of active inequalities developed to analyze the performance of the four classes of inequalities, and the instances used in the tests.

Combinations of inequalities

This section presents ten combinations of classes of inequalities. Each combination differ from each other by the classes that are activated or deactivated when running the CPA described in Section 4.3.

The triangle inequalities have been extensively studied in the literature and their efficiency has been confirmed by various researchers [81, 74, 4]. For this reason, we activated them in all combinations.

From the sixteen possible combinations of the remaining four classes of inequalities (clique, wheel, bicycle wheel and general clique), we used the ten combinations defined in Table 4.1

that emphasize the impact of individual classes. The first column of Table 4.1 gives the name of each combination, and the other five columns indicate with an X which classes are activated in that combination.

The ten combinations can be summarized as: one that activates none of the four classes (`Tri`), another that activates them all (`ALL`), and for each class "Inequality", two additional combinations (`Tri` + Inequality) and (`ALL` − Inequality).

Moreover, the ten combinations can be split into two groups: The first group, **G1**, is formed by the combinations that activate at most two classes of inequalities, and the second group, **G2**, corresponds to the combinations that deactivate at most one class.

Table 4.1 Ten combinations for study of valid inequalities.

Name	**Class of instances**				
	Triangle	Clique	General clique	Wheel	Bicycle wheel
`Tri`	X				
`T+Cl`	X	X			
`T+GC`	X		X		
`T+Wh`	X			X	
`T+BW`	X				X
`ALL`	X	X	X	X	X
`A-Cl`	X		X	X	X
`A-GC`	X	X		X	X
`A-Wh`	X	X	X		X
`A-BW`	X	X	X	X	

Instances

Of a total of 147 instances, 76 are dense (edge density greater than or equal to 0.5), and the remaining 71 are sparse.

- *Random*: Twelve complete graphs with the edge weights randomly selected and generated by `rudy` [75]. The dimension varies from 50 to 100 and the density between 0.5 and 1.
- *spinglass_2g (3g)*: Eleven instances ($3g$: four instances) of a toroidal 2D (3D)grid for a spin glass model with gaussian interactions. The grid has size $(n_{row}) \times (n_{columns})$. They are sparse with density between 0.05 and 0.3.
- *spinglass_2pm (3pm)*: Two instances ($3pm$: six instances) of toroidal 2D(3D)-grid for a spin glass model with +/-J interactions. The grid has size $(n_{row}) \times (n_{columns}) \times (n_{layers})$. The percentage of negative weights is 50%, dimension n vary from 36 to 64 and density between 0.05 and 0.3.

- *pm1s_n.i*: Ten weighted instances with edge weights drawn uniformly from $\{-1, 0, 1\}$, density 0.1 and dimension 80.
- *pm1d_n.i*: Ten weighted graphs with edge weights drawn uniformly from $\{-1, 0, 1\}$ and density 0.99. For densities $n \in \{80, 100\}$.
- *gkai_a(b or c)*: For each type (a,b or c) five instances with dimensions from 30 to 100, densities from 0.0625 to 0.5 (b: density 1 and for c: densities from 0.1 to 0.8). Diagonal coefficients from $[-100, 100]$ (b: [$-$63,0] and c:$[-100, 100]$), off-diagonal coefficients from [$-$100,100] (b:[0,100] and c: [$-$50,50]).
- *bqp_n-i* : For each dimension n, ten weighted graphs with dimension $n \in \{50, 100\}$ and density 0.1.
- *g05_n.i*: Ten unweighted graphs with edge probability 0.5 and dimension $n \in \{60, 80, 100\}$.
- *w01_100.i*: Ten graphs with integer edge weights chosen from $[-10, 10]$, density 0.1 and dimension 100.

4.4.2 Comparison methodology

This section explains the tools used to analyze our results. First, we present the performance profiles proposed in [17], then the data profiles as used in [63], followed by the performance versus time graphs, and the relevance histograms.

We define a comparison in terms of a set $\mathcal{P}$ of problems (the 147 instances) and a set $\mathcal{S}$ of optimization algorithms (the ten combinations).

Performance profiles

As in [17, 63], the performance profiles are defined in terms of final objective solution $z_{p,s} > 0$ obtained for each instance $p \in \mathcal{P}$ and combination $s \in \mathcal{S}$. In our case (relaxation of a maximization), larger values of $z_{p,s}$ indicate worse performance. For any pair (p, s) of problem p and combination s, the performance ratio is defined as:

$$r_{p,s} = 100 \left(\frac{z_{p,s} - \text{Best}_\text{p}}{\text{Best}_\text{p}} \right). \tag{4.15}$$

where $\text{Best}_\text{p} = \min\{z_{p,s} \forall p \in \mathcal{P} : s \in \mathcal{S}\}$.

The performance profiles of a combination $s \in \mathcal{S}$ is defined as the fraction of problems for which the performance ratio is at most α, that is,

$$\rho_s(\alpha) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : r_{p,s} \leq \alpha\}. \tag{4.16}$$

Thus, for a given α one knows the percentage of instances $p \in \mathcal{P}$ (y axis) that are solved for each combination $s \in \mathcal{S}$ ($\rho_s(\alpha)$ in the x axis).

Data profiles

As pointed out in [63], data profiles are useful to users with a specific computational time limit who need to choose a combination that is likely to reach a given performance. Hence, it shows the temporal evolution of combinations to a specific performance ratio ($r_{\max}$).

The data profiles are defined in terms of CPU time $t_{p,s}$ obtained for each $p \in \mathcal{P}$ and $s \in \mathcal{S}$. So, for a given time β we define the data profiles of a combination $s \in \mathcal{S}$ by:

$$d_s(\beta) = \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} : t_{p,s} \leq \beta \text{ and } r_{p,s} \leq r_{\max}\}. \tag{4.17}$$

Thus, for given $r_{\max}$ and time β, one sees the percentage of instances that can be solved for each combination $s \in \mathcal{S}$.

Note that in the computational results we do not analyze the data profiles for dense and sparse graphs separately because the obtained results did not contribute additional insights.

Performance versus time graphs

This benchmark shows the average performance ratio ($\Delta r_{p,s} \forall p \in \mathcal{P} : s \in \mathcal{S}$) in the x-axis, and the average time of execution ($\Delta t_{p,s} \forall p \in \mathcal{P} : s \in \mathcal{S}$) in the y-axis.

A combination ω dominates σ if $\Delta r_{p,\omega} < r_{p,\sigma}$ and $\Delta t_{p,\omega} < \Delta t_{p,\sigma}$. Thus, we can analyze domination between our combinations.

Relevance histograms

We use relevance histograms to quantify the impact of each inequality on the quality of the SDP bound. Let $\mathcal{I} = \{Cl, Wh, BW, GC\}$ be the set of four classes of inequalities, and $\Delta r_{p,s}, \forall p \in \mathcal{P}$ be the average performance ratio of a combination $s \in \mathcal{S}$. We calculate the relevance μ_i of the

inequality $i \in \mathcal{I}$ as:

$$\mu_i = \frac{(\Delta r_{p,Tri} - \Delta r_{p,T+i}) + (\Delta r_{p,A-i} - \Delta r_{p,ALL})}{2 \times (\Delta r_{p,Tri} - \Delta r_{p,ALL})}. \tag{4.18}$$

Equation (4.18) is normalized by the largest gap, which is the gap between combinations `Tri` and `ALL`. Thus, if an inequality $i \in \mathcal{I}$ has relevance 1 it means that `ALL` $-i$ = `Tri` and `Tri` $+i$ = `ALL`.

4.4.3 Computational results

This section presents and analyzes the results of our tests. Section 4.4.3 demonstrate the benchmarks of all types of graphs. Sections 4.4.3 and 4.4.3 show the benchmarks of sparse and dense instances, respectively.

Results on the full set of instances

In this part of the analysis we present and discuss the four benchmarks for all 147 instances (sparse and dense) together.

Performance profiles:

Figure 4.3 shows the performance profiles for the ten combinations. To facilitate the analysis, the performance ratio is limited to 0.8.

According to Figure 4.3.a) (for $k = 3$), the combinations with the best performance are `ALL`, `A-GC` and `A-Cl`, and the combinations of Group **G2** almost always dominate the ones of Group **G1**. Moreover, results suggest that the bicycle wheel is an important inequality since `T+BW` dominates all the other combinations of **G1** and `A-BW` has the worst performance of the **G2**. For the same reasons we can notice that wheel inequality is also very relevant to CPA for $k = 3$.

The results presented in Figures 4.3.b), c) and d) (for $k = 4, 5$ and 7, respectively) are very similar. It can be noticed that all combinations that activate the wheel inequality dominate the others. Moreover, we observe that all the combinations that have wheel and general clique inequalities can solve more than 70% of the instances with performance ratio less than 0.05. We also observe that the plots of `Tri` and `T+Cl` are very similar for those partitions, suggesting that the clique inequalities are ineffective for $k \in \{4, 5, 7\}$.

Data profiles:

Figure 4.4 shows the data profiles for a maximum performance ratio (r_{max}) equal to 0.0. Each graph shows the results for one of $k \in \{3, 4, 5, 7\}$. Moreover, the time in this analysis is limited to

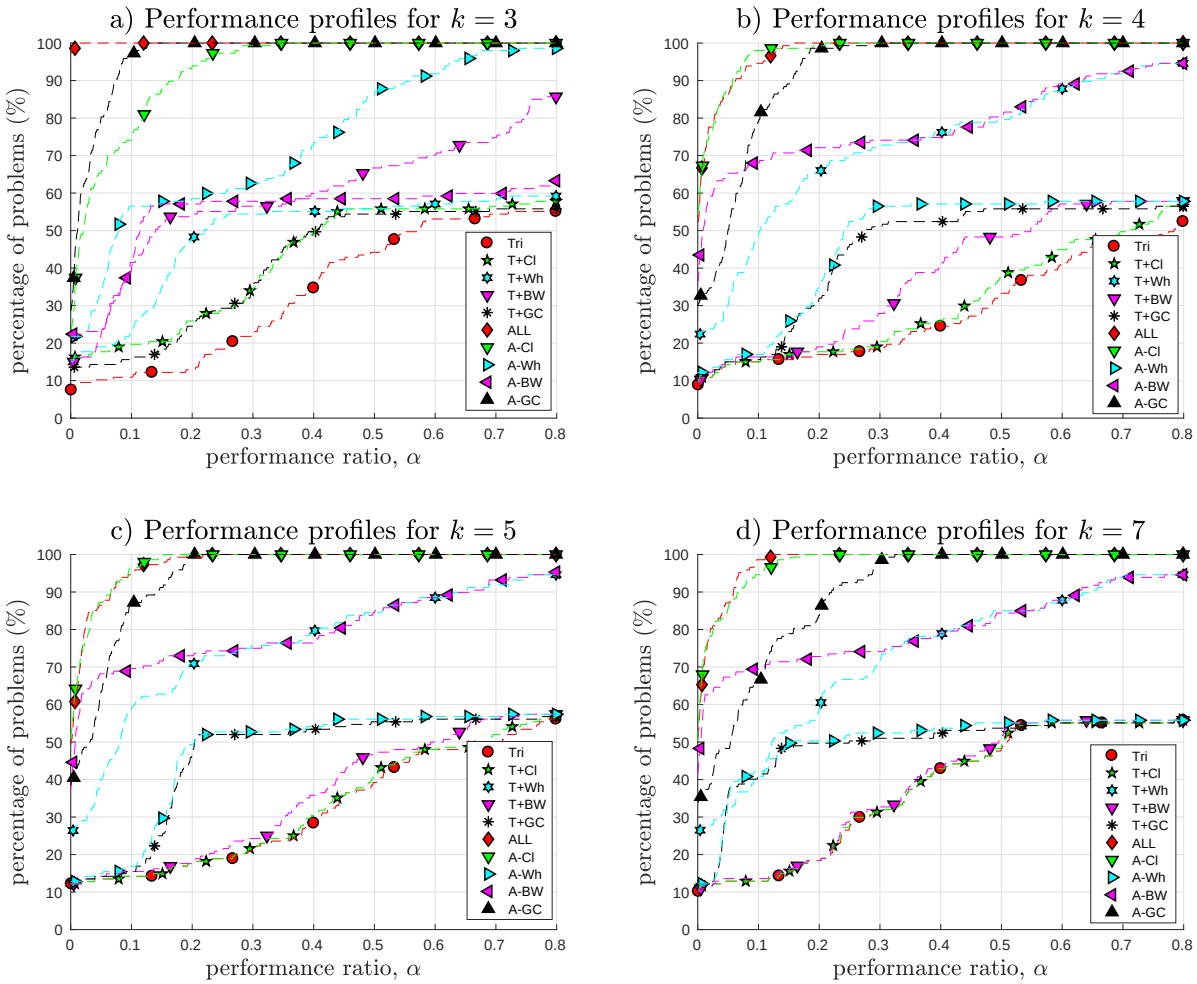

Figure 4.3 Performance profiles for all instances. Ideal point is at (0.0, 100).

5×10^4 seconds (approximately 14 hours).

In Figure 4.4.a), it can be observed that `ALL` followed by `A-GC` are the best combinations, and we see that combinations of Group **G2** perform better than those of **G1**.

The graphs in Figure 4.4.b) c) and d) can be separated in combinations with and without wheel inequality, and it can be noticed that all the algorithms that activate wheel inequalities have the better performance, e.g. `T+Wh` dominates `A-Wh` for $k \in \{4, 5, 7\}$.

Quality versus time:

Figure 4.5 presents the average performance versus CPU time for the ten combinations of the cutting plane algorithm.

In Figure 4.5.a) it may be seen that `A-Cl` and `T+GC` are dominated by `ALL` and `T+Cl`, respectively.

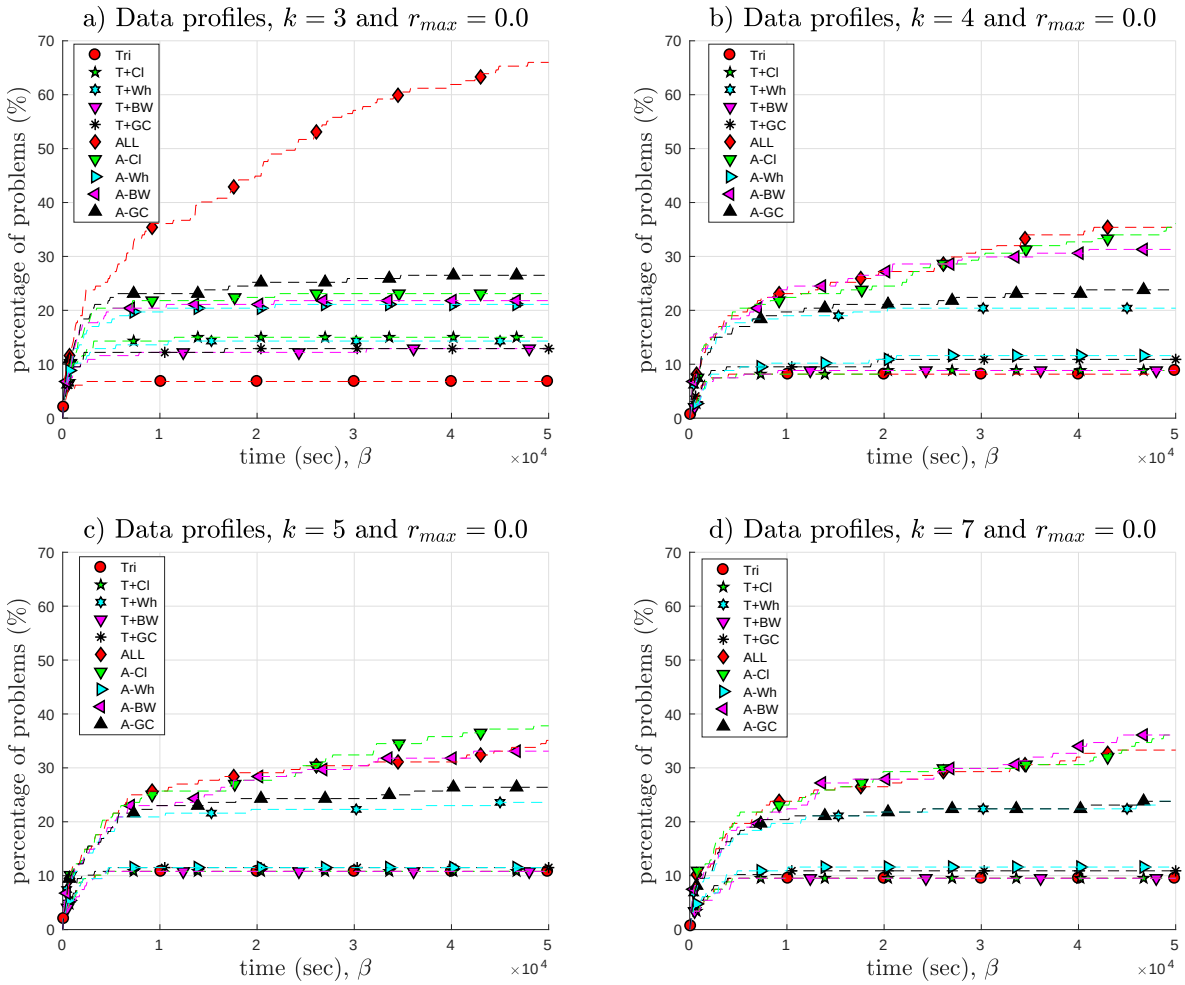

Figure 4.4 Data profiles for all instances with $r_{max} = 0.0$. Ideal point is at (0.0, 100).

Moreover, for $k = 3$, `T+Cl`, `T+GC` and `Tri` are the fastest combinations but they have the worst performance. It is also noticeable that combinations with bicycle wheel show better performance but require more CPU time.

It can be observed that the results for $k \in \{4, 5, 7\}$ (Figure 4.5.b) c) and d)) are very similar. In those graphs `A-Wh` is always dominated by `T+Wh`. We also notice that all combination that activate wheel have on average a performance ratio less than 0.2.

Finally, results in Figure 4.5 suggest that the CPU time of all the combinations is increased with the number of partitions.

Relevance of inequalities:

Figure 4.6 presents the relevance histograms for partitions $k \in \{3, 4, 5, 7\}$ of four inequalities:

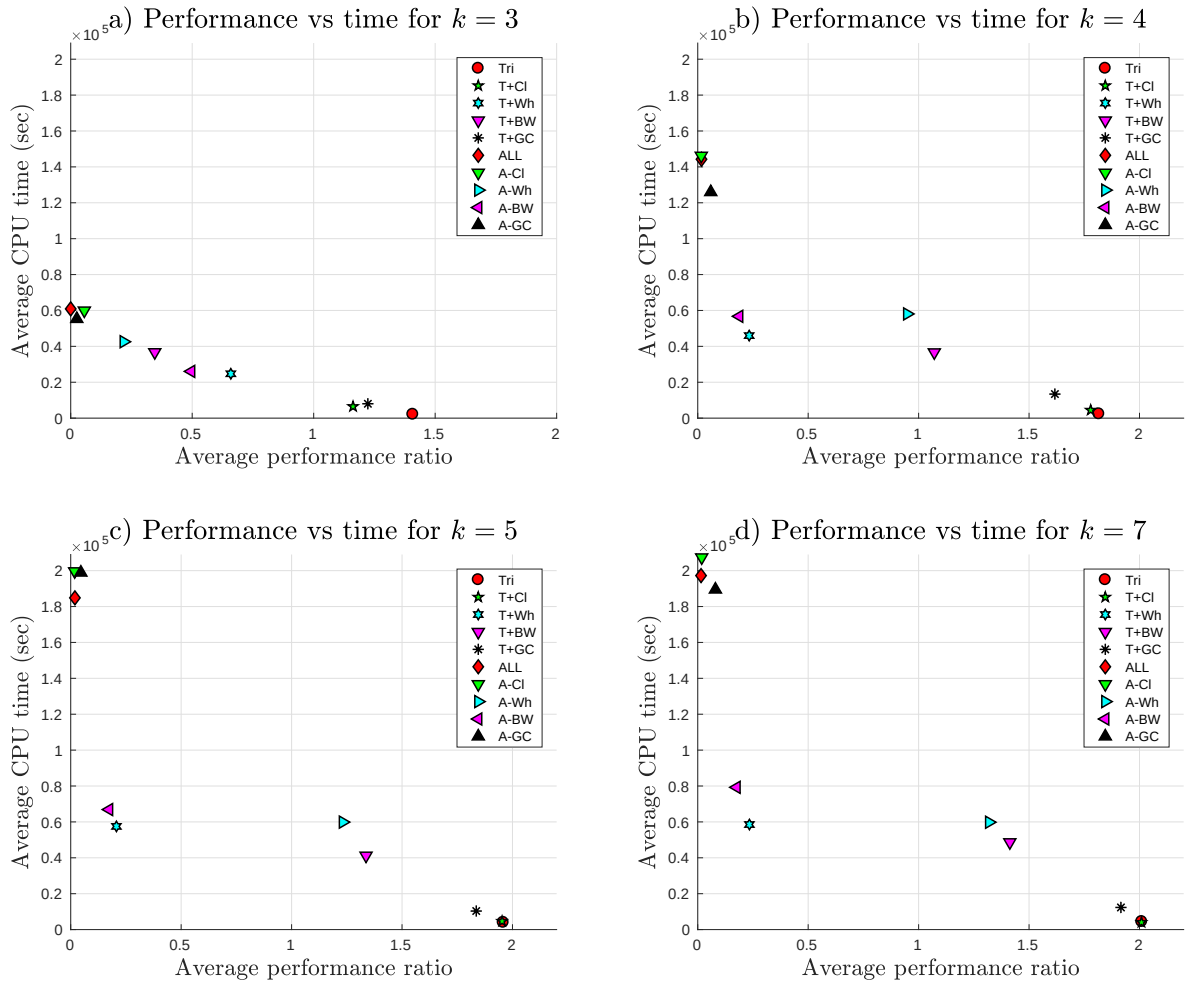

Figure 4.5 Performance versus CPU time for all instances. Ideal point is at (0.0, 0.0).

Clique, Wheel, Bicycle wheel (B. Wheel) and General clique (Gen. Clique). In Figure 4.6.a), we see that the bicycle wheel is the most relevant inequality followed by wheel. However, for $k \in \{4, 5, 7\}$ the wheel inequality is clearly the most significant. Figure 4.6 suggests that the relevance of the wheel inequality increases with k.

Results for sparse graphs

This section analyzes the results of the ten combinations on the sparse instances (density below 50%). We present the performance profiles, performance versus time and the relevance histograms for the ten combinations on these 70 instances.

Performance profiles:

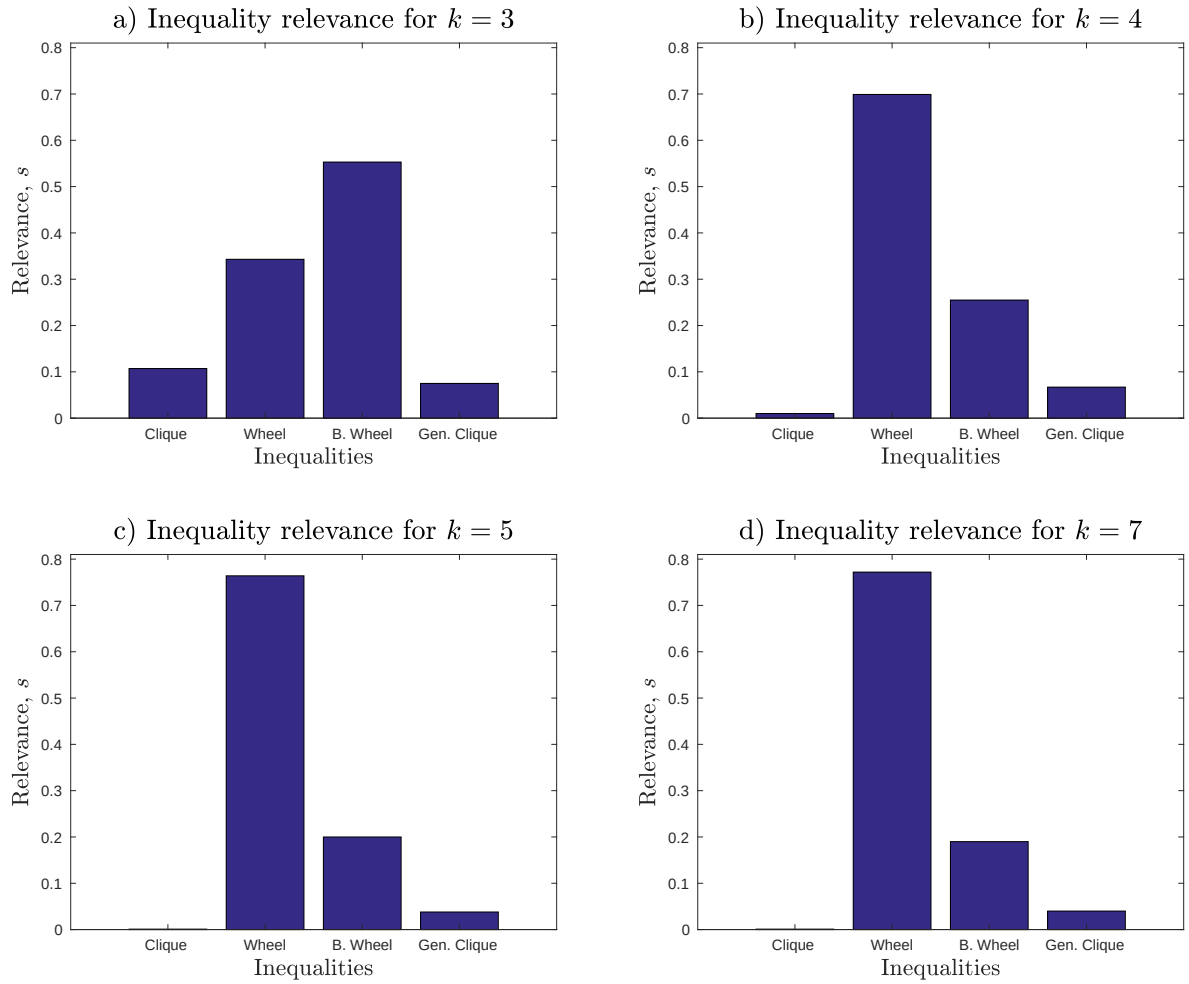

Figure 4.6 Relevance of the inequalities for all instances.

Figure 4.7 shows the performance profiles of the combinations for sparse instances.

In Figure 4.7.a) ($k = 3$), it can be observed that the combinations with the best results are `ALL`, `A-GC` and `A-Cl`, followed by `A-Wh`. For $k = 3$, it is also noticeable that bicycle wheel is the most important inequality, since all combinations that deactivate it have the worst performance. For the same reasons we can include wheel inequalities in second place of importance for $k = 3$.

Figures 4.7.b), c) and d) (for $k = 4, 5$ and 7, respectively) are very similar. It can be seen that all algorithms that the wheel inequality produce the best results. A closer look on those results reveals a strong relation between `T+Wh` (the best result for Group **G1**) and `A-BW`, suggesting that wheel is the most significant inequalities to the max-k-cut followed by bicycle wheel.

Quality versus time:

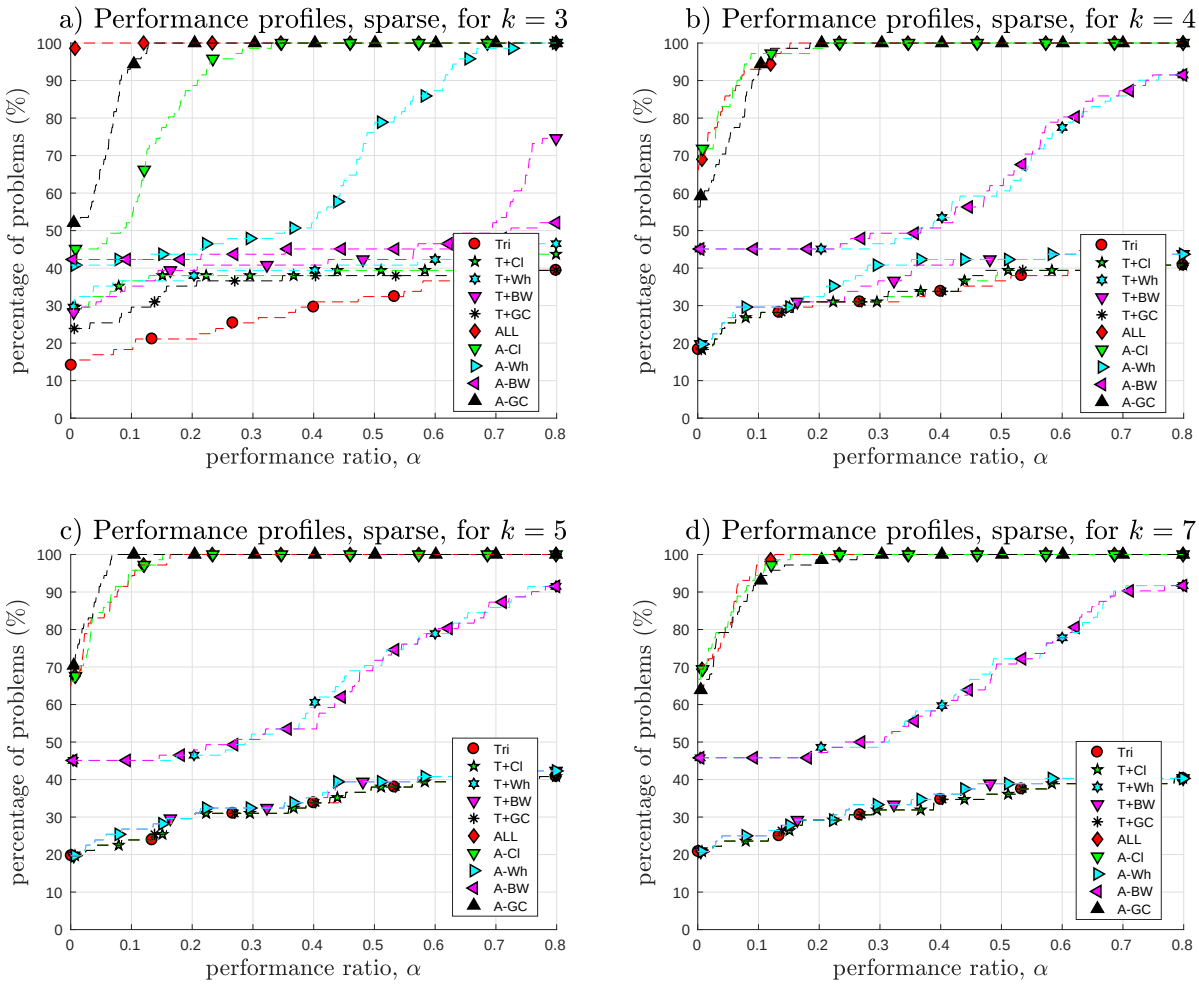

Figure 4.7 Performance profiles for sparse instances. Ideal point is at (0.0, 100).

Figure 4.8 presents the average performance versus CPU time of the combinations for sparse instances.

In Figure 4.8.a) the combinations `T+Cl`, `T+GC` and `Tri` are the fastest, but they have the worst performance. The best performance can be credited to `A-Wh` or `T+Wh` since those results are the closest to the ideal point. It is apparent that combinations with the bicycle wheel have the best performance but the highest CPU time.

It can be noticed that the results for $k \in \{4, 5, 7\}$ (Figure 4.8.b) c) and d)) are very similar. Those graphs suggest that wheel is the most relevant class of inequalities because all the combinations where it is activated have the average performance ratio lower than 0.3. Moreover, the combinations that activate wheel and bicycle wheel inequalities at the same time (`A-GC`, `A-Cl` and `ALL`) have the best average performance, but the worst CPU time. They present an average time larger than

2×10^5 seconds.

Finally, we observe that the CPU time increases with the number of partitions. For example, for $k = 7$ the combinations spend 70% more CPU time solving the cutting plane algorithm than for $k = 4$.

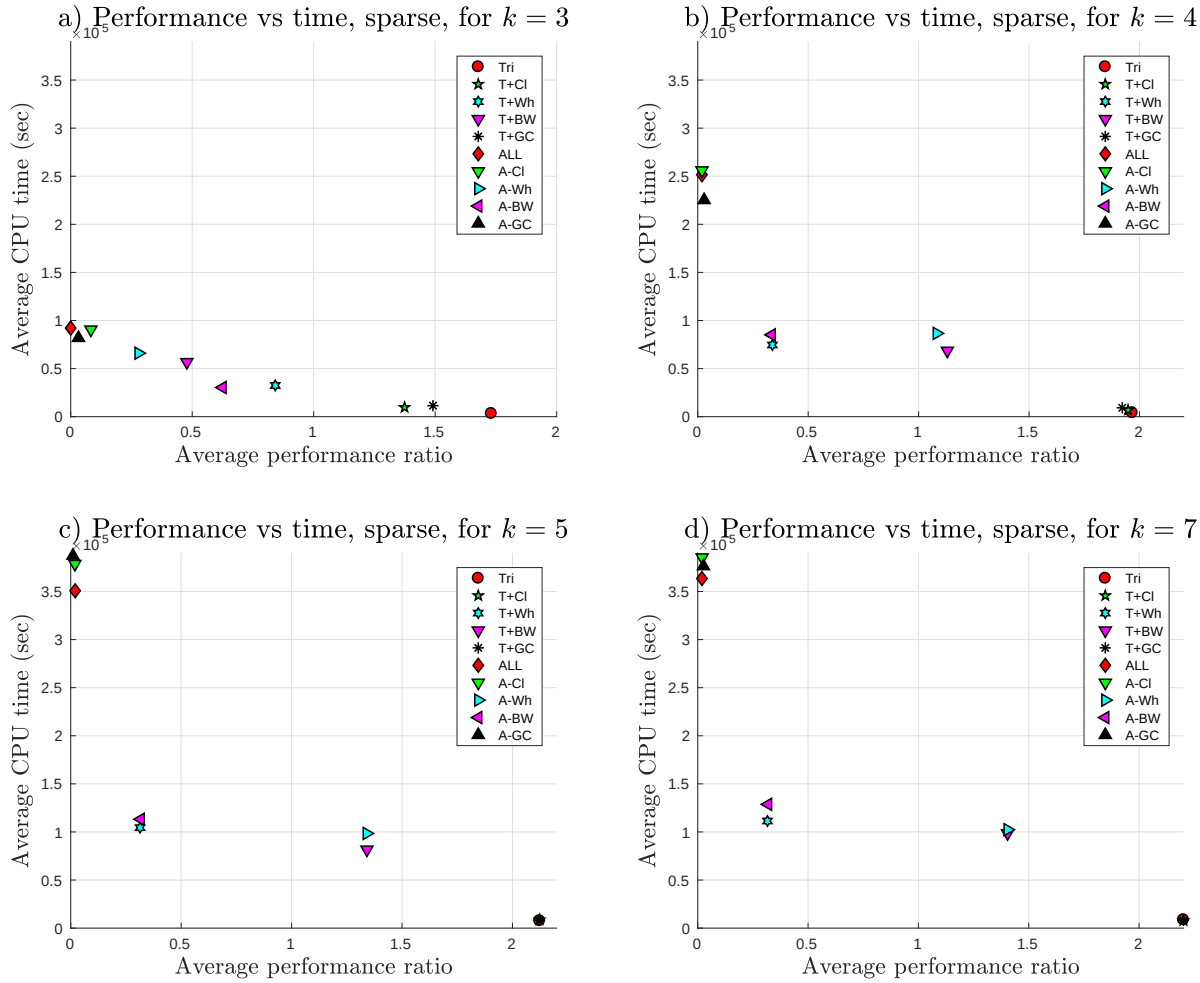

Figure 4.8 Performance versus CPU time for sparse instances. Ideal point is at (0.0, 0.0).

Relevance of inequalities:

Figure 4.9 demonstrates the relevance of the four inequalities for sparse instances. For $k = 3$, results shown in Figure 4.9.a), it can be seen that the bicycle wheel is the most relevant inequality followed by wheel.

For $k \in \{4, 5, 7\}$ it can be noticed that clique and general clique inequalities are ineffective, and that the wheel inequality is the strongest class by far.

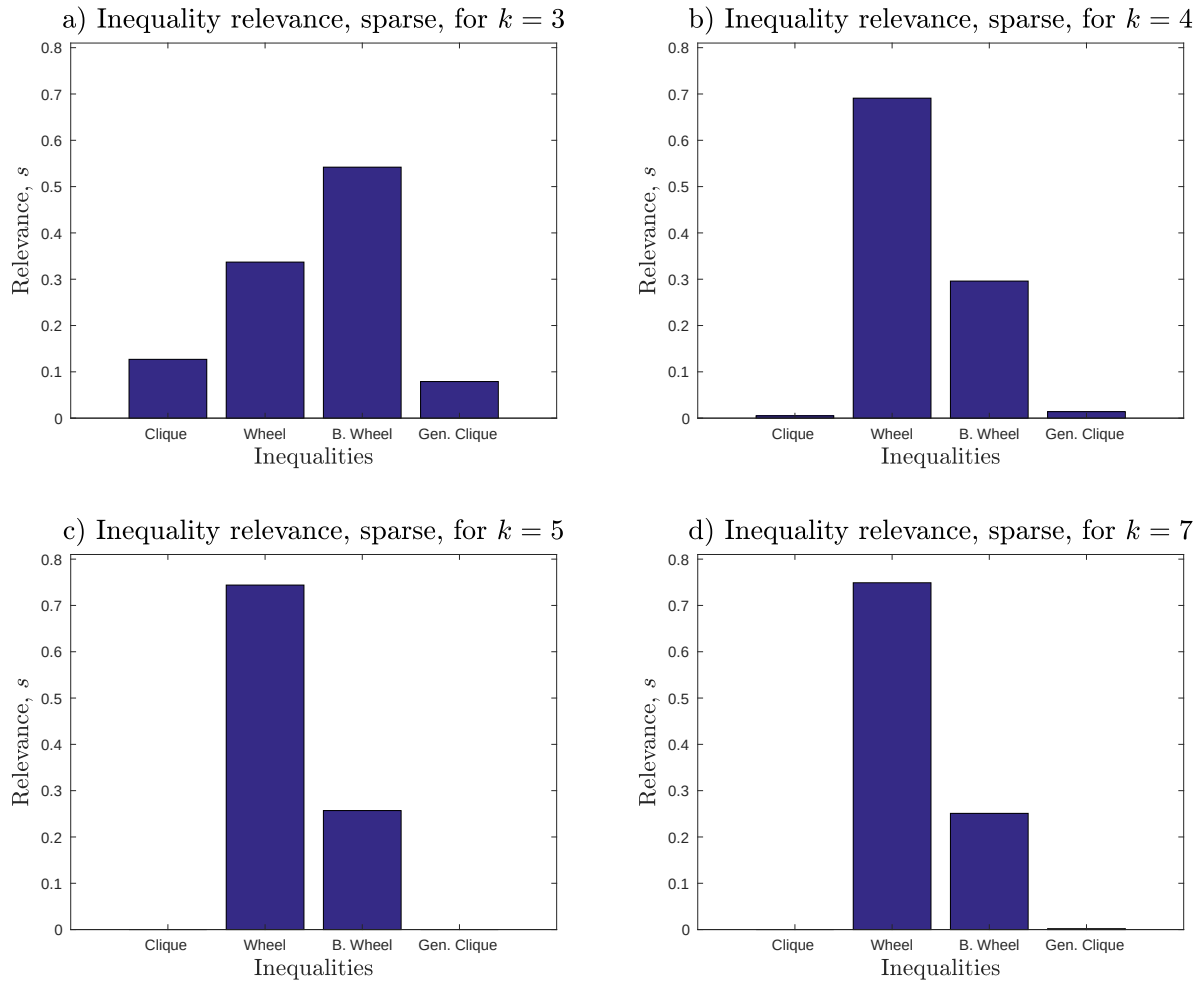

Figure 4.9 Relevance of the inequalities for sparse instances.

Results for dense graphs

This section analyzes the results of the ten combinations on the dense instances (density greater than or equal to 50%).

Performance profiles:

Similarly to the previous performances profiles, it can be observed in Figure 4.10.a) ($k = 3$) that the combinations with the best results are `ALL`, `A-GC` and `A-Cl`, followed by `A-Wh`.

The results in Figure 4.10.a) also suggest that bicycle wheel is the most significant inequality since combinations that activate it have the best performance of **G1** and **G2**. A similar analysis reveals that wheel is also relevant for this partition.

Figures 4.10.b) c) and d) (for $k = 4, 5, 7$) suggest that wheel and general clique are, respectively,

the most relevant, since the activation/deactivation of those inequalities have a huge impact in the results.

It is also apparent that for $k \in \{4, 5, 7\}$ the results of `ALL`, `A-BW` and `A-Cl` are similar and the same occurs to `Tri`, `T+Cl` and `T+BW`. It means that the activation or deactivation of clique and bicycle wheel does not change the performance of the CPA on these instances.

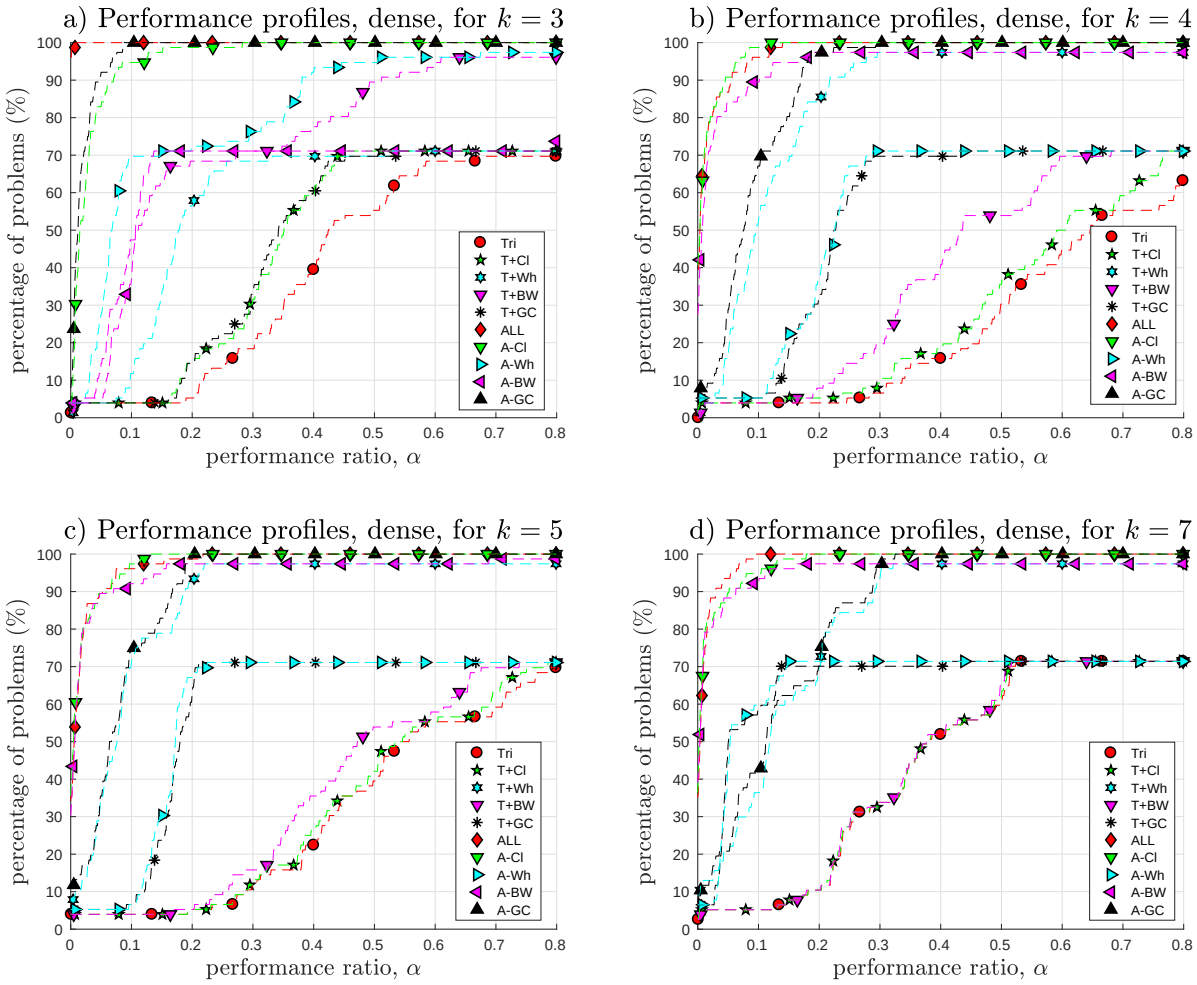

Figure 4.10 Performance profiles for dense instances. Ideal point is at (0.0, 100).

Quality versus time:

In Figure 4.11.a) the combinations `T+Cl`, `T+GC` and `Tri` are the fastest. However, they have the worst average performance (higher than 0.8). It can be seen that `A-Cl`, `A-BW`, and `T+GC` are dominated by other combinations and it is also noticeable that algorithms with bicycle wheel have the best average performance.

The results for $k \in \{4, 5, 7\}$ (Figure 4.11.b) c) and d)) are very similar. In those graphs we notice

the relevance of the wheel inequality, since all the algorithms using this inequality have average performance lower than 0.2. Furthermore, it can be concluded that `T+Wh` achieves the best results, because it is the closest to the ideal point.

Finally, we observe that all the combinations spend in average 25% more CPU time for $k = 4$ than for the other partitions.

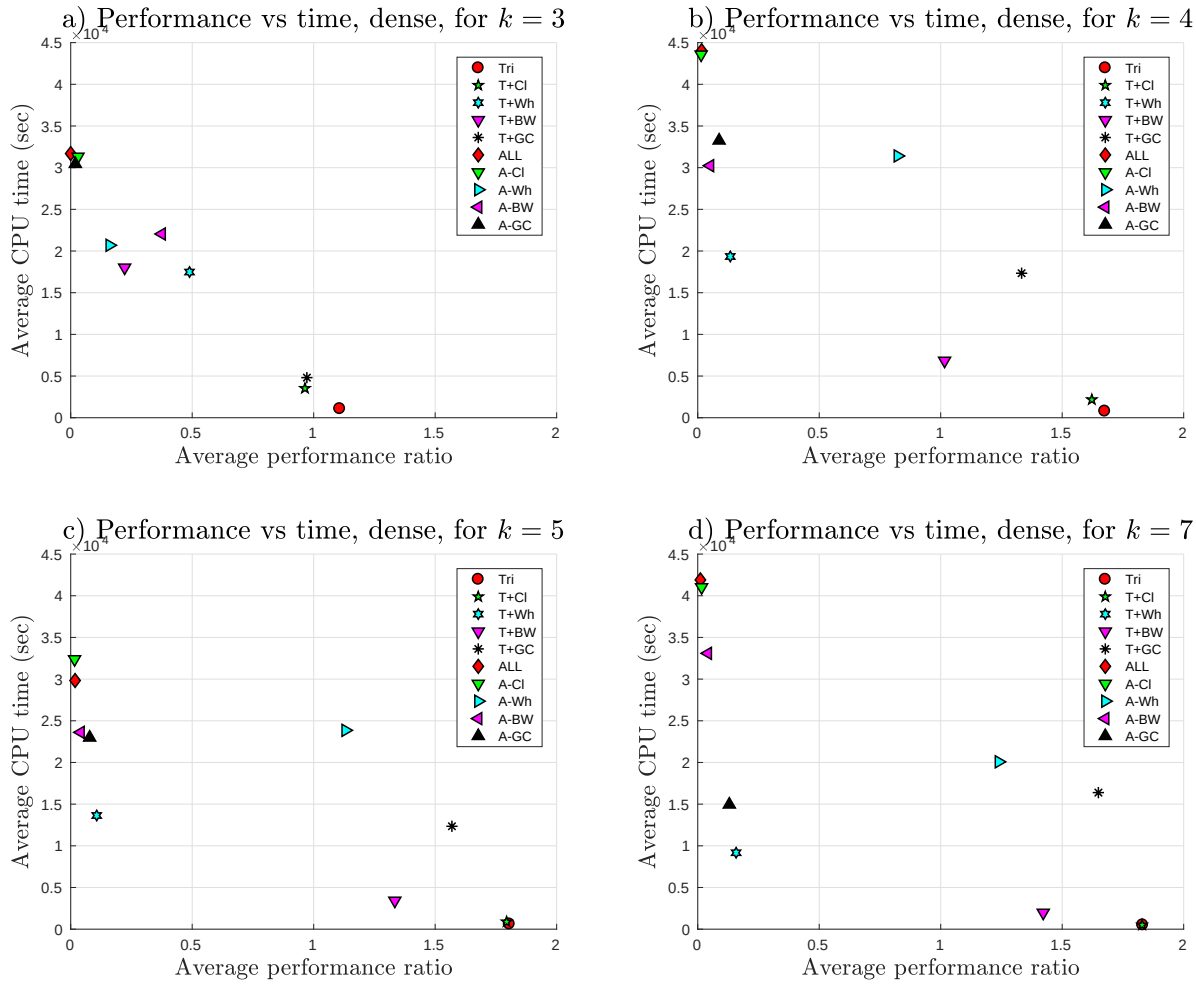

Figure 4.11 Performance versus CPU time for dense instances. Ideal point is at (0.0, 0.0).

Relevance of inequalities:

Figure 4.12 presents the four relevance histograms for dense instances. In Figure 4.12.a) it can be noticed that the bicycle wheel is the most relevant class of inequality followed by wheel and clique.

For $k \in \{4, 5, 7\}$ the results suggests that wheel inequalities are the most relevant. It is also notable that the clique and general clique inequalities are ineffective when the triangle inequalities are

activated.

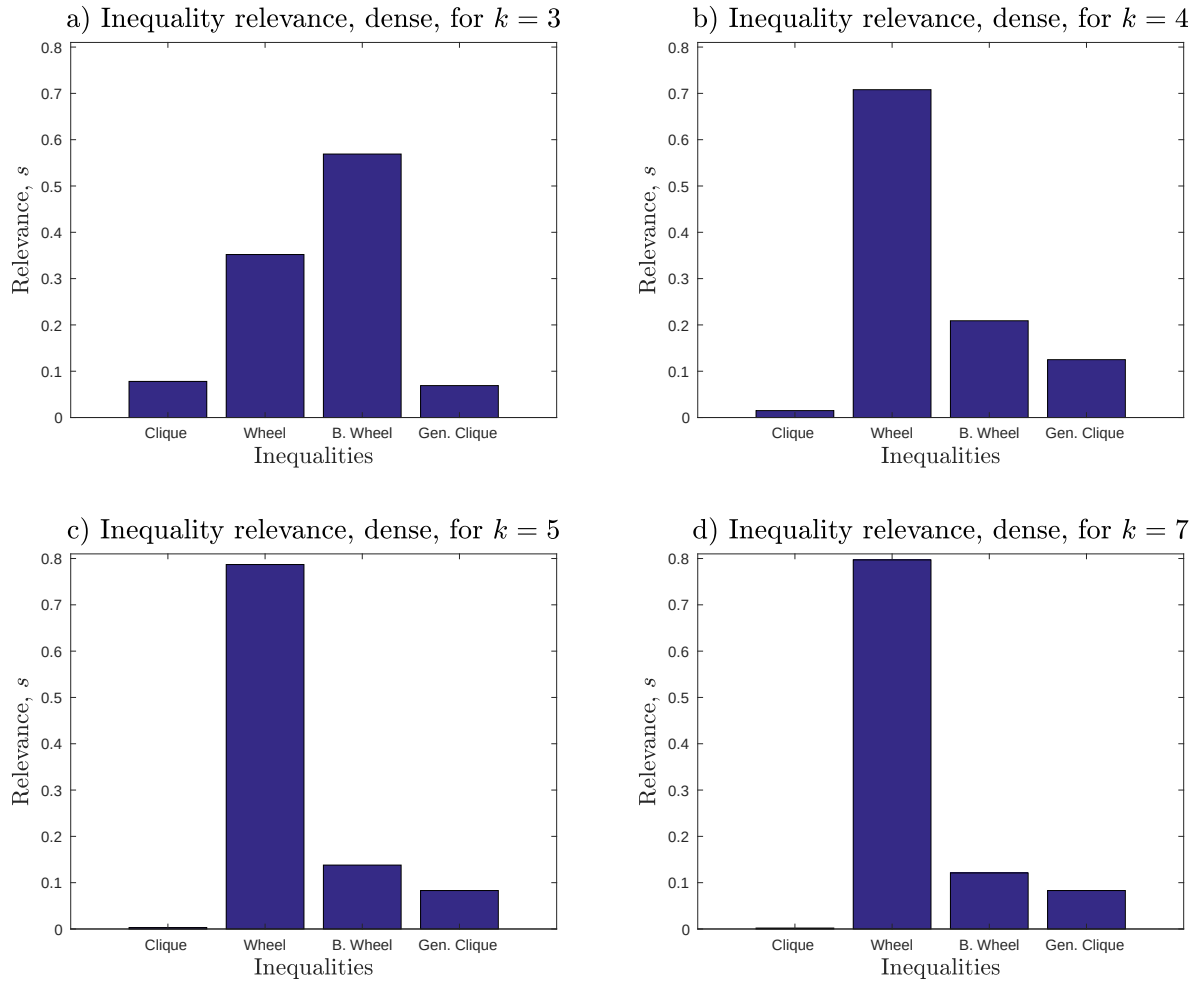

Figure 4.12 Relevance of the inequalities for dense instances.

4.4.4 Summary of the computational tests

While it is normal that the combination `ALL` (activation of all the inequalities) has the best performance, it is also the combination that requires the most CPU time. On the other hand, `Tri` is the fastest combination but delivers the worst performance. The combination `T+Wh` achieves, on average, the best ratio between CPU time and performance for all values of k considered in this study.

Table 4.2 summarizes the results and shows the best inequalities for each benchmark (performance profile, performance versus time (QvsT) and inequality relevance) for each value of $k \in \{3, 4, 5, 7\}$, and for sparse/dense/all instances. We observe that the strongest inequalities for $k = 3$ are bicycle

wheel and wheel, and for $k \in \{4, 5, 7\}$ the wheel inequalities are the strongest.

It is also noticeable that the activation of wheel combined with bicycle wheel (or general clique for dense graphs) improves the performance for all types of partitions. However, it spends in average 72% (32%) more CPU time for the CPA to terminate. Generally, we see that an increase in k causes an increase in the CPU time, especially for the sparse instances.

From a practical point of view, wheel is the most attractive inequality because it shows good performances for all instances and for all types of partitions.

Table 4.2 Summary of the best inequalities for each benchmark.

k	**Density**	**Performance profiles**	**QvsT**	**Relevance**
3	All	Bicycle Wheel and Wheel	Bicycle Wheel	Bicycle Wheel and Wheel
	Sparse	Bicycle Wheel and Wheel	Bicycle Wheel	Bicycle Wheel and Wheel
	Dense	Bicycle Wheel and Wheel	Bicycle Wheel	Bicycle Wheel and Wheel
4,5,7	All	Wheel and General Clique	Wheel	Wheel
	Sparse	Wheel and Bicycle Wheel	Wheel	Wheel
	Dense	Wheel and General Clique	Wheel	Wheel

4.5 Discussion

In this paper we carried out an experimental study of several classes of valid inequalities to strengthen the SDP relaxation of the max-k-cut problem. Specifically we considered the following classes of cutting planes: Triangle, Clique, General clique, Wheel and Bicycle wheel. We developed a different separation algorithm for each class. For triangle inequalities we used complete enumeration, for clique, general clique, and wheel inequalities a greedy heuristics was developed, and for bicycle wheel a genetic algorithm was applied. The inequalities were then tested within a standard cutting plane algorithm for $k \in \{3, 4, 5, 7\}$, using nearly 150 instances with varying properties.

To investigate the strength of each class of inequalities we created ten combinations by alternating the activation and deactivation of the inequalities. In order to guarantee a fair comparison of the combinations, we analyzed four different benchmarks: performance profiles, data profiles, quality versus time and relevance histograms.

The results are summarized in Table 4.2. We conclude that bicycle wheel is the strongest inequality for $k = 3$ and wheel for $k \in \{4, 5, 7\}$. Moreover, the inclusion of wheel and bicycle wheel inequalities at the same time strengthens even more the SDP relaxation of the max-k-cut at the cost of 72% more CPU time, on average.

In future research we plan to study more efficient algorithms to solve the SDP relaxation with

wheel and bicycle wheel inequalities, and to build a branch-and-cut algorithm to find the integer solution of the max-k-cut problem for both sparse and dense graphs for large instances.

Acknowledgments

This research was supported by Discovery grants 312125 (M.F. Anjos) and 418250 (S. Le Digabel) from the Natural Sciences and Engineering Research Council of Canada.

CHAPTER 5 ARTICLE 2: IMPROVING THE LINEAR RELAXATION OF MAXIMUM K-CUT WITH SEMIDEFINITE-BASED CONSTRAINTS

Authors: Vilmar J. Rodrigues de Sousa, Miguel F. Anjos, Sébastien Le Digabel.
Submitted: April 2018 on *EURO Journal on Computational Optimization*

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

5.1 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.

The max-k-cut problem is equivalent to the minimum k-partition problem [31, 85], and the special case $k = 2$ that is known as the max-cut problem has attracted considerable attention; see, e.g., [6, 33, 51, 70, 74].

Many industrial applications can be formulated as the max-k-cut problem, including VLSI layout design [6], statistical physics [52], and wireless communication problems [22, 68].

The general max-k-cut is known to be $\mathcal{NP}$-hard [71]. Nonetheless, many relaxations [10, 73], heuristics [56], approximations [25, 47], and exact methods [4, 21, 60] 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 5.1.1 reviews the SDP and linear programming (LP) formulations of the max-k-cut problem. Section 5.2 presents the SDP-based inequalities. Section 5.3 describes in detail the cutting plane algorithm (CPA) used to solve the relaxations, and Section 5.4 discusses the test results. Finally, some concluding remarks are made in Section 6.7.

5.1.1 Formulations

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

Semidefinite programming formulation

The vertex formulation of the max-k-cut leads to an SDP relaxation. In the approximation method of [25] 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:

$$
\begin{aligned}
(\textit{MkC-SDP}) \qquad \max_{X} \quad & \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - X_{ij}) && && (5.1)\\
\text{s.t.} \quad & X_{ii} = 1 && \forall i \in V, && (5.2)\\
& X_{ij} \geq \frac{-1}{k-1} && \forall i, j \in V, i < j, && (5.3)\\
& X \succeq 0. && && (5.4)
\end{aligned}
$$

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 [14, 25] and exact methods [4, 31]. In particular, [25] extends the max-cut approximation

of [33] to the max-k-cut. In [4] the bundleBC algorithm is proposed to solve max-k-cut problems with 70 vertices by combining the SDP branch-and-cut method of [31] with the principles of the `Biq Mac` algorithm [74]. In [4] the authors show that their method achieves a dramatic speedup in comparison to [31], especially when $k = 3$.

Linear formulation

Chopra & Rao [10] presented an edge-only 0-1 formulation of max-k-cut. For each edge $(i, j) \in E$, the variable x_{ij} takes the value 0 when (i, j) is cut, and 1 otherwise. Hence, the edge-only linear relaxation of max-k-cut can be formulated as:

$$
\begin{aligned}
(\textit{MkC-LP}) \qquad \max_{x} \quad & \sum_{i,j \in V, i<j} w_{ij}(1 - x_{ij}) && && (5.5)\\
\text{s.t.} \quad & x_{ih} + x_{hj} - x_{ij} \leq 1 && \forall i, j, h \in V, && (5.6)\\
& \sum_{i,j \in Q, i<j} x_{ij} \geq 1 && \forall Q \subseteq V \text{ with } |Q| = k + 1, && (5.7)\\
& 0 \leq x_{ij} \leq 1 && \forall i, j \in V. && (5.8)
\end{aligned}
$$

where Constraints (5.6) and (5.7) 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., [9, 10, 60]. In [9, 10] 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 [21], 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 [22]. 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 [22, 46, 79], and a chordal extension [39, 85]. The second disadvantage is mitigated by a CPA (Section 5.3) 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 [2, 10, 22]. In [2] 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 formu-

lations are expensive and impractical for large graphs.

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 [31], 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 [19], 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 [4] 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 [54]. 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 5.3.1). However, for small values of k, the latter outperforms the former.

5.2 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.

5.2.1 Semi-infinite formulation of SDP

The SIP can be defined as an optimization problem with finitely many variables and infinitely many constraints. The survey [45] discusses the theory, algorithms, and applications of semi-infinite programming. In [49] the authors study linear semi-infinite programming (LSIP) for generic SDPs.

We note that the convex constraint $X \succeq 0$ (5.4) is equivalent to

$$\mu^T X \mu = \mu\mu^T \bullet X \geq 0 \qquad \forall \mu \in \mathbb{R}^n. \tag{5.9}$$

where $n = |V|$ and typically the Euclidean norm of μ is one. Theorem 1.1.8 of [40] proves this equivalent characterization of positive semidefinite matrices. Moreover, [40] provides more fundamental results from linear algebra and the properties of the cone of symmetric semidefinite matrices.

The matrix constraint (5.9) has an infinite number of rows. By replacing (5.4) by (5.9) in *MkC-SDP* we obtain the LSIP formulation of SDP. In the cut-and-price approach proposed in [50] the authors use the LSIP of the dual SDP formulation for the *maxcut* problem. Their results suggest that the linear approach is able to solve large-scale problems.

5.2.2 Variable transformations

To incorporate Constraint (5.9) in our linear formulation we need to transform the semidefinite variable $X_{ij} \in \left[\frac{-1}{k-1}, 1\right]$ to the related $x_{ij} \in [0,1]$ 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.

5.2.3 SDP-based inequality formulation

By applying the transformation proposed in Section 5.2.2 to Constraint (5.9) we derive the following class of inequalities for *MkC-LP*:

$$\sum_{i,j \in V, i<j} \mu_{ij} x_{ij} \geq \frac{1}{k} \sum_{i,j \in V, i<j} \mu_{ij} - \frac{k-1}{2k} \sum_{i \in V} \mu_{ii} \quad \forall \mu \in \mathbb{R}^n. \tag{5.10}$$

In [49] the authors prove that these inequalities ensure that the set of linear solutions is feasible for the SDP. In Section 5.3.1 we propose an exact separation routine to deal with the infinite number of constraints.

5.3 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 5.1. 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 others termina-

tion criteria can be used, e.g., number of iterations, computational time, and improvement at each iteration.

5.3.1 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 [76], for Constraint (5.3) in the SDP formulation, and for Constraint (5.10) proposed in this work.

Separation of combinatorial inequalities

Some valid and facet-inducing inequalities have been proposed in [10] for the *MkC-LP*. Five of these families of constraints are explored computationally in [76], 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 [76] 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 5.3.1).

Separation of bound inequalities

In [40] the author indicates that it is more efficient to start the CPA with only the diagonal Constraints (5.2) of the SDP formulation and to separate $X_{ij} \geq \frac{-1}{k-1}$ iteratively.

Figure 5.2 compares the performance of the SDP formulation with and without bound separation. It shows the percentage gap (see Section 5.4.1) versus the CPU time (s) for 10 random instances, with a density of 0.8 and dimension $|V| = 100$.

Figure 5.2 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 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.

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 5.3.3).
3. **Search for violations.** Use the separation routine to find violated inequalities at the current solution (Section 5.3.1).
4. **Add inequalities.** If there are violated inequalities then add at most $NbIneq$ (see Section 5.3.1) 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 5.3.2).
6. **Modify current iterate.** Increment i. Reduce or increase ε_T, if necessary. Return to Step 2.

Figure 5.1 Scheme of cutting plane algorithm (CPA).

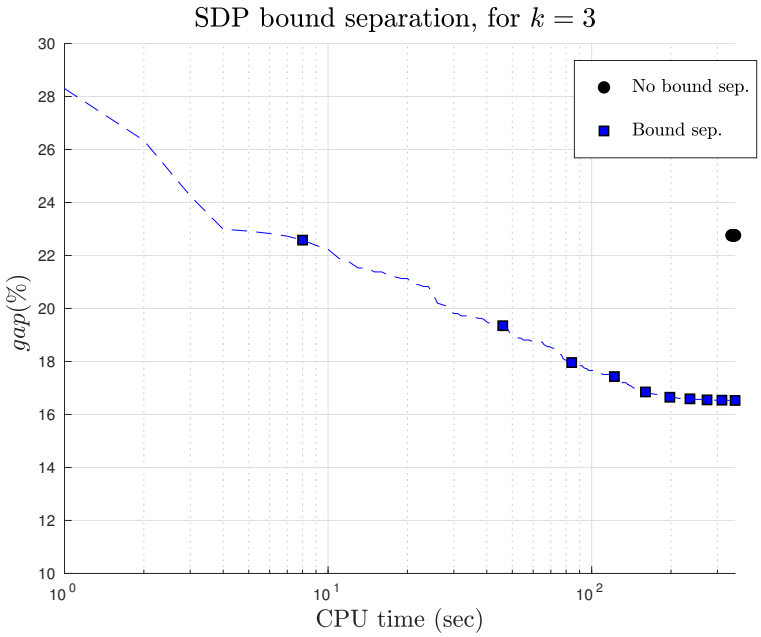

Figure 5.2 Separation of Constraint (5.3) in the SDP formulation.

Therefore, we apply bound separation for Constraint (5.3). We perform a complete enumeration of all edges $e \in E$, and only the $NbIneq$ (see Section 5.3.1) most violated inequalities are added for the next CPA iteration.

Separation of SDP-based inequalities

The family of SDP-based inequalities (5.10) has infinite combinations, which makes it impractical to add all of them in any formulation. Since the eigenvalues of a positive semidefinite matrix are non-negative [40], 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 ($\hat{X} \not\succeq 0$) then it has at least one negative eigenvalue $\lambda < 0$, and the following inequalities are violated by $\hat{x}$:

$$\sum_{i,j \in V, i<j} v_{ij} x_{ij} \geq \frac{1}{k} \sum_{i,j \in V, i<j} v_{ij} - \frac{k-1}{2k} \sum_{i \in V} v_{ii}. \tag{5.11}$$

where the columns of $\mathbf{v}$ are the eigenvectors corresponding to the values λ. The addition of (5.11) 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` [37] 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 $\mathcal{O}(9n^3)$.

Maximum number of inequalities in CPA

As shown in [76], 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 $NbIneq = 300$ for the SDP formulations, similarly to [76].

5.3.2 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 to solve the problem.

In [61] 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 ($Ite_{drop} = 3$).

5.3.3 Solving the relaxations

One of the most important decisions in the CPA is the choice of the method for solving the relaxation. We solve the SDP and LP relaxations of the max-k-cut using the interior point method (IPM) of `mosek` [5]. Our computational tests indicated that the default IPM is not efficient so, inspired by the `PDCGM` solver [35], 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 [34, 58] 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 [61] 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 [65], 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 [58] 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 5.3.1) 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 5.3 plots the data profiles (see Section 5.4.3) 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 5.3 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.

5.4 Computational tests

We solve the SDP and LP relaxations of max-k-cut using the IPM of `mosek` [5] on a Linux PC with two Intel(R) Xeon(R) 3.07 GHz processors. We performed tests for $k \in \{3, 4, \ldots, 10, 0.1|V|\}$ on 228 test problems.

5.4.1 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 [62].
- **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:

$$gap_{p,m} = \frac{UB_{p,m} - LB_p}{LB_p}. \tag{5.12}$$

- **Iteration time** ($itime_{p,m}$): The CPU time for one CPA iteration for method m and problem

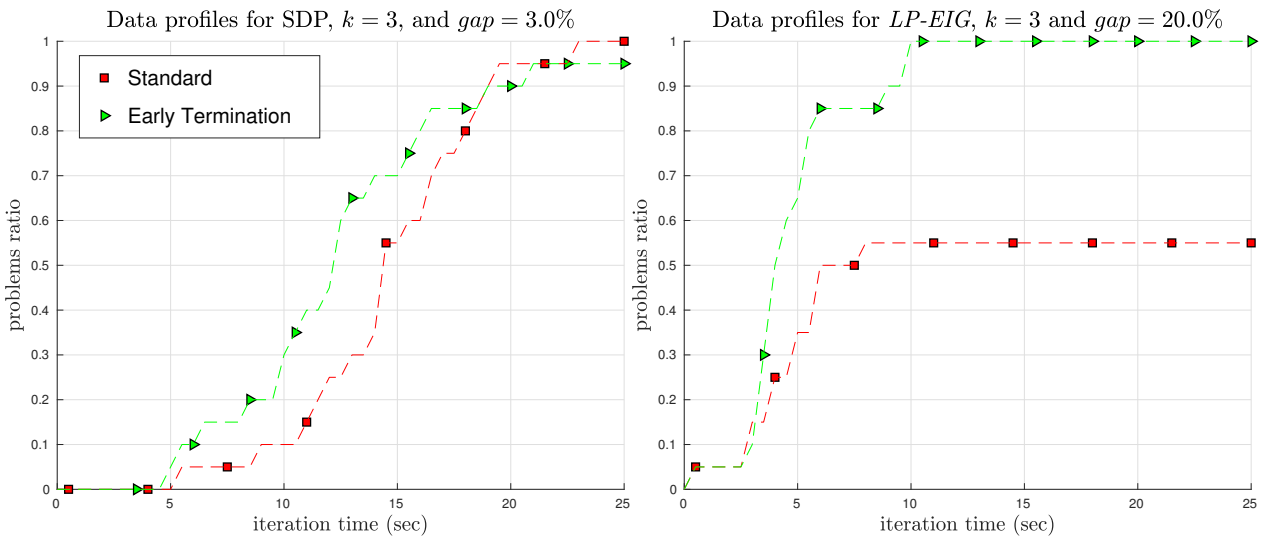

Figure 5.3 Study of early termination in interior point method (IPM).

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 5.3.1):
  - LP: Solves the LP formulation.
  - *LP-EIG*: Solves the LP formulation with the separation of SDP-based inequalities (Constraint (5.10)).
  - SDP: Solves the SDP formulation with the separation of bound inequalities (Section 5.3.1).

5.4.2 Instances

We consider 228 instances; 68 are from the `Biq Mac` library [88] and 160 were randomly generated using `rudy` [75].

- `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.

5.4.3 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 [17], and the data profiles [63]. 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 $\mathcal{K}$.

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.

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.,

$$\rho_m(\alpha) = \frac{1}{|\mathcal{P}|} \mid \{p \in \mathcal{P} \mid \mathrm{UB}_{p,m} \leq \alpha\} \mid . \tag{5.13}$$

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

Data profiles

As observed by [76], 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

$$d_m(\beta) = \frac{1}{|\mathcal{P}|} \mid \{p \in \mathcal{P} \mid itime_{p,m} \leq \beta, gap_{p,m} \leq gap_{max}\} \mid . \tag{5.14}$$

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

5.4.4 Computational results

This section presents and analyzes our computational results. Section 5.4.4 shows the performance tables for the `Biq Mac` instances. Section 5.4.4 presents these tables for the random instances. To compare the performance of SDP and ***LP-EIG*** we present the data profiles in Section 5.4.4 and the performance profiles in Section 5.4.4.

Performance tables: `Biq Mac` instances

Table 5.1 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 very strong but their computation are expensive: it takes approximately 1200 s to solve the IPM.

Table 5.2 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*** outper-

Table 5.1 Performance comparison for `Biq Mac` instances and $k = 3$.

name	dens.	$\|V\|$	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	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

forms 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.

Performance tables: *random* instances

Table 5.3 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 5.4 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.

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 Section 5.4.4 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 5.4, we present the data profiles for instances with positive weights, i.e., all 80 problems of the family $pRnd$ and 10 from g05. Figure 5.5 displays the results for instances with mixed

Table 5.2 Performance comparison for `Biq Mac` instances and $k = 10$.

name	dens.	$\|V\|$	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

Table 5.3 Performance comparison for random instances and $k = 3$.

weight	dens.	$\|V\|$	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

Table 5.4 Performance comparison for random instances and $k = 10$.

weight	dens.	$\|V\|$	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

weights, i.e., 80 instances from $nRnd$, 20 from be, and 10 from bqp, pm1s, and pm1d.

Positive weights. Figure 5.4 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 5.5 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.

Performance profiles

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

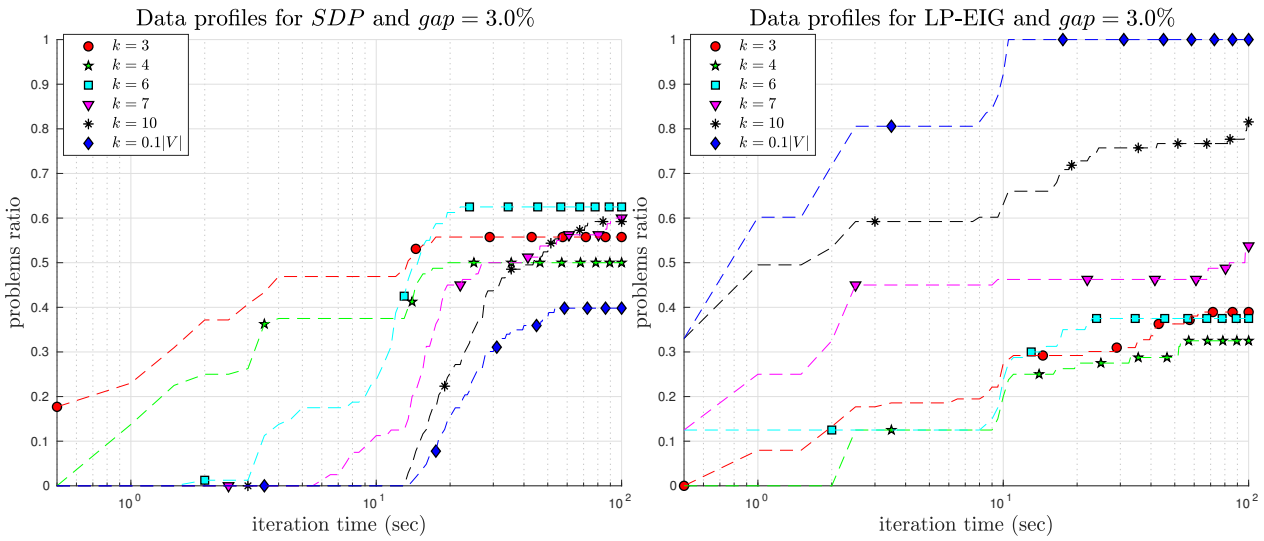

Figure 5.4 Data profiles for instances with positive weights for various values of partition size k.

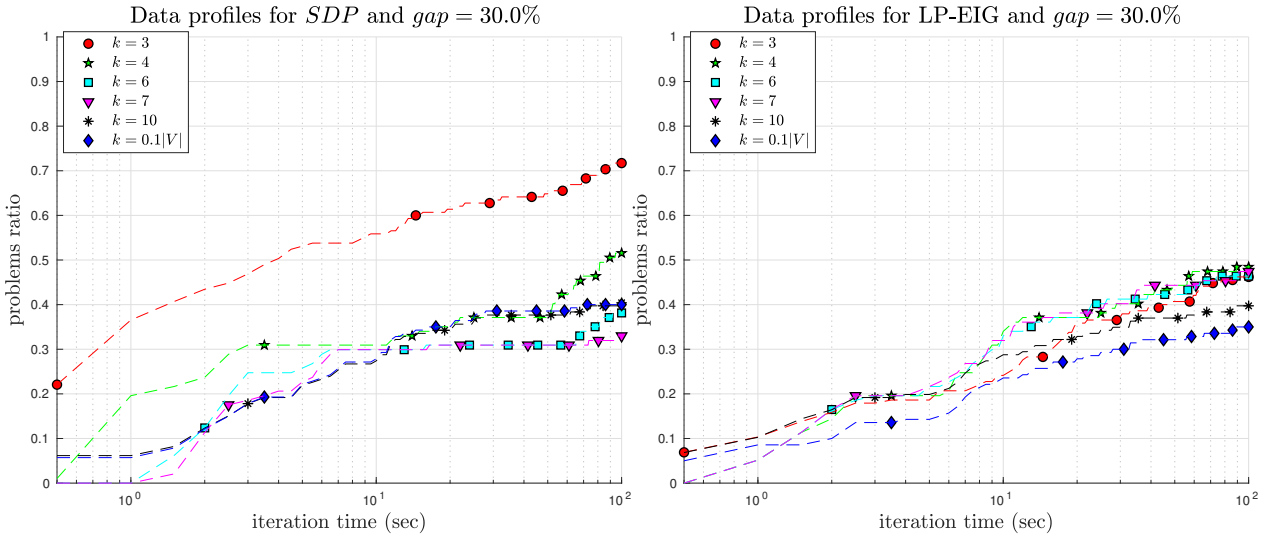

Figure 5.5 Data profiles for instances with mixed weights for various values of partition size k.

Positive weights. Figure 5.6 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 5.7 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 5.6), because no method could solve the instances with lower gaps, even when we allowed a higher value for $itime$. In Figure 5.7 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.

5.4.5 Summary of computational tests

The tables of Section 5.4.4 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 presented in Section 5.4.4 indicate that *LP-EIG* is more efficient than SDP for positive weights with $k \geq 7$ and for mixed weights with $k \geq 4$. For $k = 3$, on the other hand, the SDP consistently outperforms the linear formulations.

Table 5.5 presents a summary of our computational results, indicating the best method for each type of problem.

5.5 Discussion

We have proposed a family of SDP-based constraints (5.10) 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.

Table 5.5 Best method(s) 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*

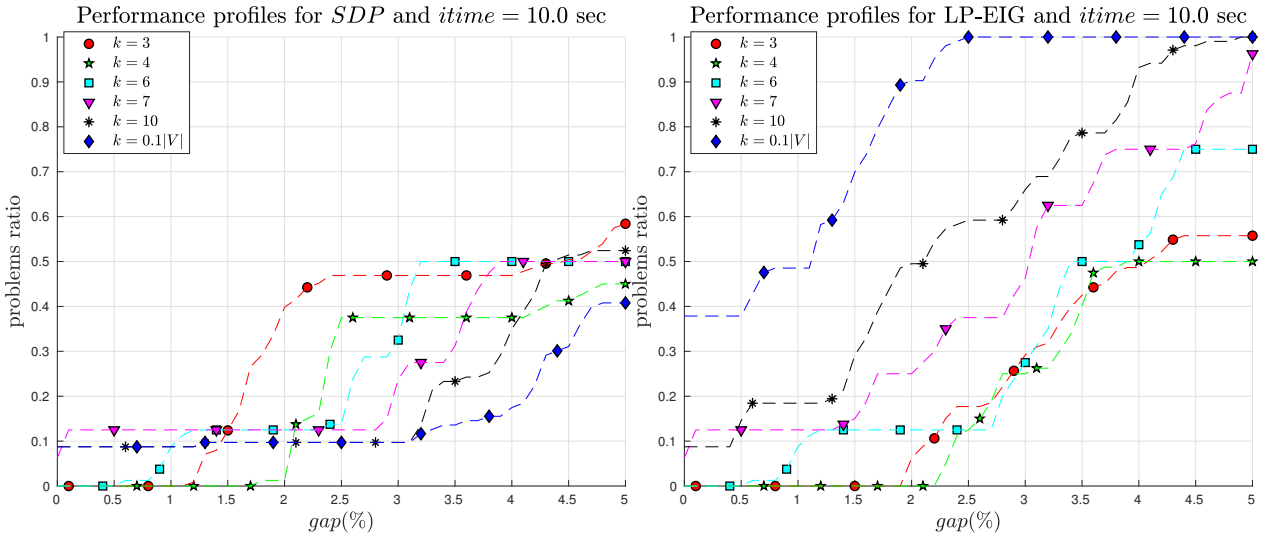

Figure 5.6 Performance profiles for instances with positive weights for various values of partition size k.

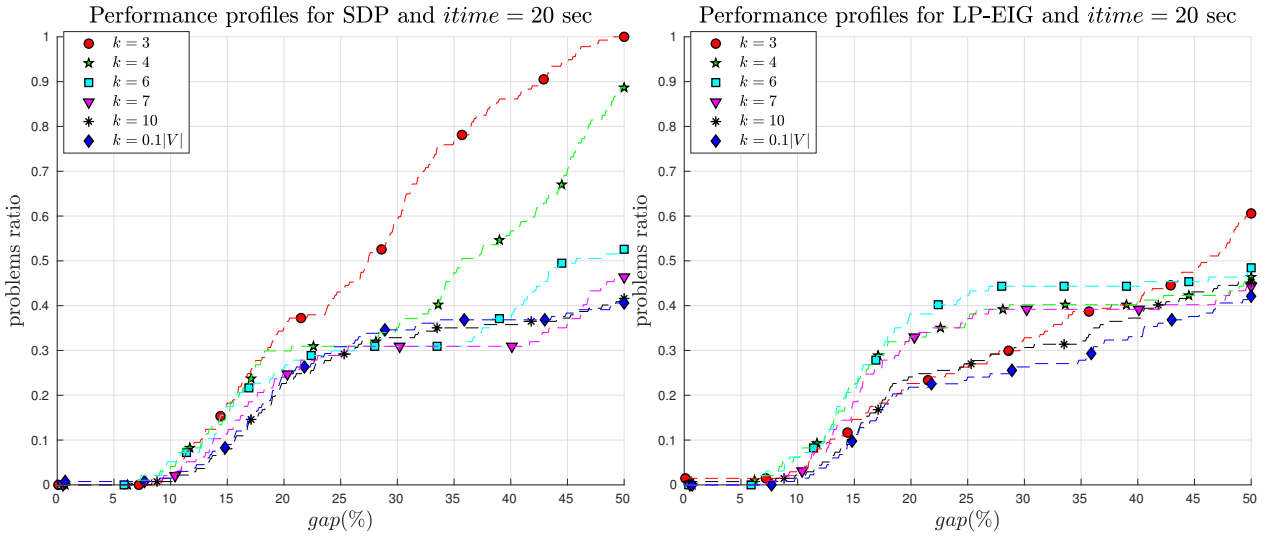

Figure 5.7 Performance profiles for instances with mixed weights for various values of partition size k.

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 5.5.

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.

CHAPTER 6 AN EXACT METHOD FOR THE MAXIMUM K-CUT PROBLEM

6.1 Introduction

The max-k-cut is a challenging graph partitioning problem that is known to be a classical problem of the class of $\mathcal{NP}$-complete problems [71]. It attracts scientific attention from the discrete community because of its variety of applications.

The objective of the max-k-cut problem is to partition the vertex set of a graph ($G = (V, E)$) into at most k subsets in a way that the sum of the weights of the edges joining vertices in different subsets is maximized.

In this chapter, we investigate the branch-and-bound algorithm to obtain global solutions for the max-k-cut problem. Therefore, we focus on reviewing all the relevant exact approaches proposed in the literature.

In [60, 59] the author applied a branch-and-cut algorithm based on the linear programming (LP) formulation of the k-way equipartition problem in the realignment of a Football league where $k = 8$ and $|V| = n = 32$. In [59], for graphs of sizes between 100 to 500 vertices, the problem is solved with a gap of less than 2.5% for $k = 4$.

A cut-and-branch algorithm (see Section 6.3.1) based on the node-and-edge linear formulation of max-k-cut is applied in [22] to solve a two-level graph partitioning problem in mobile wireless communications. In [22], the LP relaxation is tightened by general clique based inequalities. Computational results found the optimal solution for problems with $n = 100$ and $k \in \{2, 3, 4\}$ for sparse graphs.

A branch-and-bound framework based on the edge formulation of max-k-cut is studied in [85]. In [85], the authors show that in a chordal graph the linear and semidefinite formulations can be defined with only $|E|$ variables instead of $\frac{n(n-1)}{2}$. Therefore, using a chordal extension they were capable of solving to optimality some sparse graphs with $n = 200$ and $k \in \{3, 4\}$ in a few seconds.

The semidefinite programming (SDP) formulations of max-k-cut are known to provide strong bounds, especially for dense graphs and small values of k, see for example [4]. Therefore, some researchers have used this formulation to design exact methods. In [31] the authors design a branch-and-cut algorithm based on SDP (called SBC) for a variant of max-k-cut called minimum k-partition. In the SBC, the SDP relaxation was tightened with triangle and clique inequalities. Their SDP is solved using a dynamic version of the bundle method [43]. The SBC algorithm computes optimal solutions for dense graphs with 60 vertices, and for sparse graphs with 100 vertices.

The BundleBC algorithm [4] is an improvement of the SBC, where the ideas of Biq Mac solver [74] are extended to the max-k-cut problem. In [4], the separation of clique inequalities and some rules for choosing the variables are investigated. Computational results show that the use of the clique inequalities reduces the number of subproblems analyzed in the branch-and-bound tree and that the BundleBC solver is faster than SBC.

Our main contribution is to conceive an exact method to efficiently solve instances of the max-k-cut problem. We investigate several components of the branch-and-bound method, for example, the procedures for branching and for finding good feasible solutions and four heuristic methods to find feasible solution. Moreover, this work focuses on studying a branch-and-bound method that uses the strengthened edge-only LP relaxation of the max-k-cut problem proposed in [77] and the constrained SDP relaxation presented in [76].

This chapter is organized as follows. Section 6.2 presents the generic scheme of the branch-and-bound method. Section 6.3 presents the LP and SDP formulation to obtain upper bounds and four heuristic methods to provide feasible solutions (lower bounds). Section 6.4 studies strategies to select variables and nodes in the branch-and-bound tree. A computational study is provided in Section 6.6. Finally, a discussion of results and conclusion are presented in Section 6.7.

6.2 A generic branch-and-bound framework

Various combinatorial problems are solved with the branch-and-bound algorithm [64]. Fundamentally, this method uses a tree search strategy that implicitly enumerates all the solutions.

Initially, the branch-and-bound algorithm consists of only the *root node*, that corresponds to the original relaxation of max-k-cut. At each iteration i of the method, a node $\mathcal{P}_i$, not yet explored, is selected (*node selection*) and we check if this node can be fathomed (pruned), i.e., if it is integer feasible, or infeasible, or if it has an upper bound not better than a known feasible solution (also called an incumbent). Otherwise, it will generate children nodes (*branching*) that satisfy all the constraints of $\mathcal{P}_i$ and others that are imposed during the branch. In a branch-and-bound algorithm, the upper and lower bounds are improved until optimality can be proven by generating an optimality certificate.

The difference between the branch-and-bound and branch-and-cut algorithms is that at each iteration of the latter it is executed a cutting plane algorithm (CPA) to solve the relaxed formulation of a node to obtain an *upper bound*.

Although the branch-and-bound framework is simple and its description can be found in any work of enumerative methods for $\mathcal{NP}$-complete problems, e.g., [74], it is important to describe and investigate some of its components to develop an efficient solver.

The following is a brief summary of some procedures of the branch-and-bound that are studied in detail in next sections.

1. *Upper bound.* The upper bound (z_{ub}) is obtained by a solution of the relaxed max-k-cut problems. It is important to have a method that gives a good trade-off between the time to obtain relaxed solution and the quality of its bound. The SDP and LP formulations are considered, but we focus on the latter, especially the strengthened LP formulation introduced in [77].

2. *Feasible solution or lower bound.* Fast feasible solutions (z_{lb}) that have a value close to the optimal can speed up the branch-and-bound method. Section 6.3.2 presents four different heuristics to obtain feasible solutions:

   - an iterative clustering heuristic (ICH) [31] ,
   - a multiple operator heuristic (MOH) [56],
   - a variable neighborhood search (VNS) [38, 62], and
   - a greedy randomized adaptive search procedure (GRASP) [23].

3. *Splitting problem.* In [26], the authors pointed out that an important decision that impacts the effectiveness of the branch-and-bound and branch-and-cut methods is the strategy used to partition the feasible region. We investigate the branching for a vertex or for an edge of the graph. Hence, two types of splitting are studied:

   - *Dichotomic*, where the branching variables correspond to an edge $(i, j) \in E$ of the graph where each node is split into two children: in the first, the edge (i, j) is *cut*, and in the second, it is *non-cut*, and
   - *Polychotomic* or k-chotomic branching. In this strategy, a vertex $i \in V$ is set to a different partition in each subproblem. Max-k-cut problems can have at most k partitions. Hence, each node can generate k children.

4. *Branching rule.* Commonly, in the branch-and-bound algorithm, a variable that has a fractional value is selected to be branched. A relaxed solution can have several non-integral variables and some of them may impact most the next bounds. Thus, it is appropriate to use some techniques for choosing good variables to be branched. Section 6.4.2 investigates five different techniques (rules).

5. *Node selection.* At each branch-and-bound iteration, a node is selected from a set of *active* (unexplored) nodes using a selection strategy. This procedure is also known as the exploration tree strategy. The following strategies are studied in Section 6.4.3:

- the best-first search strategy (BeFS),
- the depth-first search (DFS), and
- the breadth-first search (BrFS).

The generic branch-and-bound for max-k-cut can be summarized in the scheme described in Algorithm 4.

input : Graph $G = (V, E)$
output : An optimal solution x^*
initialize:
$\mathbf{P}_0 =$ Initial problem ;
$x^* =$ `Lower-bound(`$\mathbf{P}_0$`)` ; `/* Feasible Solution */`
$z^*_{lb} =$ `Evaluate-solution(`x^*`)`;
L $= \{(\mathbf{P}_0, \infty)\}$ (problem List) ;
while L $\neq \emptyset$ **do**
  Select: $\mathbf{P} \in$ L ; `/* select node */`
  Remove: L $=$ L$\backslash\{\mathbf{P}\}$;
  Compute:
    $x_k =$ `Lower-bound(`$\mathbf{P}$`)` ;
    $z_{lk} =$ `Evaluate-solution(`x_k`)`;
    $z_{ub} =$ `Upper-bound(`$\mathbf{P}$`)` ; `/* Upper bound */`
  if $z_{lk} > z^*_{lb}$ **then**
    Update: $z^*_{lb} = z_{lk}$ and $x^* = x_k$;
    Remove all the problems $(\mathbf{P}, z_{ub})$ from L with $z_{ub} \leq z^*_{lb}$;
  end
  if $z_{ub} > z_{lb}*$ **then**
    Select variable for branch in $\mathbf{P}$; `/* Branching rule */`
    Generate subProblems from $\mathbf{P}$; `/* Splitting Problem */`
    foreach *subproblem* $\mathbf{P}_i$ **do**
      L $=$ L $\cup (\mathbf{P}_i, z_{ub})$
    end
  end
end

Algorithm 4: Generic branch-and-bound algorithm for max-k-cut.

In Algorithm 4, the initial problem ($\mathbf{P}_0$) is a relaxed formulation of the max-k-cut. The CPA presented in [77], also discussed in Section 6.3.1, is used to strengthen this first relaxation in the root node.

The `Lower-bound`() function returns a feasible heuristic solution of a problem (see Section 6.3.2) that can be computed at each iteration or occasionally. In order to reduce the total running time of

the branch-and-bound framework, the `Lower-bound()` function is executed at each 15^{th} iteration or when the node is a leaf node, i.e., a node where all the edges and vertices are assigned.

Results are shown and discussed simultaneously with the description of each component of the branch-and-bound algorithm and the procedures that impact the most are presented first. Therefore, the bound procedure is investigated in the next section.

6.3 Bounding procedures in the branch-and-bound framework

This section presents the methods used in our branch-and-bound framework for calculating the upper and lower bounds of the max-k-cut. Section 6.3.1 presents two formulations used to obtain the upper bound. Section 6.3.2 investigates four heuristic methods used to generate feasible solutions.

6.3.1 Computing upper bounds

This section presents and discusses the SDP and LP formulations of the max-k-cut. It also presents the CPA used to incorporate valid inequalities to strengthen these formulations.

Formulations

The SDP and LP formulations have been extensively studied in the literature for graph partitioning problems (see [9, 21]).

The SDP formulation. The vertex based formulation of max-k-cut induces an SDP relaxation [25]:

$$
\begin{aligned}
(SDP) \qquad \max_{X} \quad & \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - X_{ij}) && && (6.1)\\
\text{s.t.} \quad & X_{ii} = 1 && \forall i \in V, && (6.2)\\
& X_{ij} \geq \frac{-1}{k-1} && \forall i,j \in V, i<j, && (6.3)\\
& X \succeq 0. && && (6.4)
\end{aligned}
$$

where the variable $X = (X_{ij})$ for each $i, j \in V$ is equal to $\frac{-1}{k-1}$ if vertices i and j are in different partitions of the k-cut of G and $X_{ij} = 1$ otherwise. Computational results in [77] show that the bound Constraint (6.3) should be separated in the CPA.

One of the advantages of using the SDP relaxation for max-k-cut it that some "hypermetric inequalities" are implicit in the SDP formulation [19]. Therefore, it produces stronger bounds than

linear formulations.

The LP formulation. In the edge-only formulation that is presented in [10], the variable x_{ij}, for each $i, j \in V$, is equal to 0 when the edge (i, j) is cut, and 1 otherwise. The strengthened formulation, with SDP-based constraints, proposed in [77] is:

$$
\begin{aligned}
(\textit{LP-EIG}) \quad & \max_{x} \sum_{i,j \in V, i<j} w_{ij}(1 - x_{ij}) && && (6.5)\\
\text{s.t.} \quad & x_{ih} + x_{hj} - x_{ij} \leq 1 && \forall i, j, h \in V, && (6.6)\\
& \sum_{i,j \in Q, i<j} x_{ij} \geq 1 && \forall Q \subseteq V \text{ with } |Q| = k+1, && (6.7)\\
& \sum_{i,j \in V, i<j} \mu_{ij} x_{ij} \geq \frac{1}{k} \sum_{i,j \in V, i<j} \mu_{ij} - \frac{k-1}{2k} \sum_{i \in V} \mu_{ii} && \forall \mu \in \mathbb{R}^n, && (6.8)\\
& 0 \leq x_{ij} \leq 1 && \forall i, j \in V. && (6.9)
\end{aligned}
$$

where typically the Euclidean norm of μ is one. The Constraint (6.8) has an infinite number of rows, in consequence, a separation routine based on eigenvalues is introduced in [77]. The triangle (6.6) and clique (6.7) inequalities are also separated in the CPA.

One advantage of linear methods in the branch-and-bound framework is that the optimal solution x^* of an LP relaxation can easily be used for *fixing variables*. Let $e \in E$ be an edge, and z_{lb} be the value of the best known feasible solution, and d_e be the reduced cost of e, and v^* the value of relaxed solution x^*. Hence, if $x^*_e = 0$ and $v^* - d_e < z_{lb}$, then the variable x_e has value 0 at optimality, in consequence, x_e can be fixed to 0. Similarly, if $x^*_e = 1$ and $v^* + d_e < z_{lb}$ the variable x_e can be fixed to 1.

Another advantage of using the linear formulation in the branch-and-bound algorithm is that the simplex method is well suited for exploiting an advanced starting basis [57]. In the branch-and-bound tree, only a few constraints are different from one node to another. Therefore, solvers (e.g. `mosek` [5]) can use the solution of a previous node to reduce computing time of the current node.

Cutting plane algorithm and valid inequalities

An integer programming (IP) formulation can be tightened, without removing any feasible solution, by the addition of cutting planes (also called valid inequalities). A CPA is a method that, at each iteration, execute the following tasks: it solves a relaxation of an IP, searches for violated inequalities in the current solution, it adds some of them to the relaxation and restarts the process.

A branch-and-bound algorithm with a CPA was introduced in [69] for the traveling salesman problem to develop the so-called branch-and-cut method where a CPA is deployed at every subproblem in the search tree. Before comparing the performance of the branch-and-bound with the branch-and-cut method, some valid inequalities applied in the CPA are presented.

Valid Inequalities. Some valid and facet-inducing inequalities for the max-k-cut have been proposed in [10], and the relevance of triangle, clique, general clique, wheel and bicycle wheel is studied in [76], where some heuristic and exact methods are introduced to separate violated inequalities in a relaxed solution. The separation of these combinatorial constraints for *LP-EIG* and SDP is also deployed in [77] together with the separation of bound Constraint (6.3) for SDP and SDP-based Inequalities (6.8) for *LP-EIG*. We apply the same separation routines introduced in [76] and [77].

Next section compares the performance of branch-and-bound with CPA in the root node only (also called cut-and-branch) with branch-and-cut where the CPA is executed at every node of the search tree.

Implementation details: cut-and-branch versus branch-and-cut

A CPA with the early-termination method of an interior point method is used in [77]. Computational results show that this procedure improves the solutions of both SDP and *LP-EIG* methods. In [77], the CPA is stopped after 1 hour and we notice that some instances require more than 100 seconds to be solved. In order to apply the same CPA in our branch-and-bound algorithm, two others stopping criteria are incorporated: *iteration time* ($\mathrm{it\_time}$) and *iteration improvement* ($\mathrm{it\_impr}$).

The time to solve a subproblem is limited to 10 seconds ($\mathrm{it\_time} = 10s$). Therefore, if a CPA iteration costs more than $\mathrm{it\_time}$ the method is stopped and we collect the last bound obtained. The second stopping criterion is based on an empirical observation of the CPA. We noticed that the largest improvements of CPA occur in the first iteration of the method. Thus, if the CPA is stopped when the improvement is close to a small $\varepsilon \geq 0$ (stop the CPA if $\mathrm{it\_time} < \varepsilon$) so it can avoid expensive and non-important rounds of the algorithm. Let z_i be the value of the solution at iteration i. The improvement, $\mathrm{it\_impr}$, is calculated as:

$$\mathrm{it\_impr} = \frac{z_{i-1} - z_i}{z_i}$$

This work uses $\varepsilon = 10^{-4}$. This way, the bounds in our branch-and-bound will not be as strong as the one presented in [77], but it is possible to explore more nodes. In the branch-and-bound

framework, the CPA is always deployed on the root node because the LP and SDP formulations can have an exponential number of constraints.

Parameters of branch-and-bound. In order to provide a fair comparison, the following parameters are fixed for both methods:

- *feasible solution*: the best solution is set as the initial feasible solution. Hence, there is no need of using heuristic methods,
- *branching rule*: the rule R4 is used to select variable in a solution (see Section 6.4.2),
- *selecting node*: the BeFS is fixed as the separation strategy (see Section 6.4.3),
- *splitting problem*: we use the k-chotomic strategy (see Section 6.4.1), and
- *time*: the branch-and-bound is stopped after one hour.

Computational results: cut-and-branch versus branch-and-cut

Table 6.1 shows the performance of the cut-and-branch (`C&B`) and branch-and-cut (`B&C`) methods. First columns of Table 6.1 define the instances, and it is shown the time in seconds (time(s)) and the number of visited nodes ($\#nodes$) for each method.

Table 6.1 Results for cut-and-branch and branch-and-cut for $k \in \{3, 5\}$.

Instance				cut-and-branch		branch-and-cut	
name	k	$\|V\|$	dens.	time(s)	$\#nodes$	time(s)	$\#nodes$
data_random	3	30	1.0	-	-	**718.3**	3319
data_clique	3	40	1.0	111.6	1192	**15.6**	83
data_clique	3	60	1.0	878.1	657	**112.7**	65
random	3	25	0.5	563.9	628744	**42.4**	296
data_3g	3	64	0.1	**8.5**	726	22.1	244
random	3	70	0.1	-	-	**883.9**	900
data_random	5	30	1.0	-	-	-	-
data_clique	5	40	1.0	2726.7	51139	**20.6**	102
random	5	25	0.5	**1.1**	1927	73.7	8757
data_3pm	5	60	0.1	1.0	0	1.0	0
data_3g	5	64	0.1	**106.7**	9504	188.6	2350
random	5	70	0.1	**10.5**	750	240.8	600

Table 6.1 shows that in general the branch-and-cut outperforms the cut-and-branch for $k = 3$. For example, `C&B` could not solve in one hour a problem of family data_random with only 30 vertices. The `C&B` is better than the `B&C` when instances are very sparse and for $k = 5$.

6.3.2 Lower bound

In a branch-and-bound algorithm, feasible solutions (incumbent solutions) are used to prune subproblems, and it can still be returned as a solution if the method stop before optimality.

Heuristic and meta-heuristic methods are normally used to find initial feasible solutions and to improve the existing incumbent. For the max-cut ($k = 2$) problem, many heuristics have been proposed (see [89, 48]). For the max-k-cut, there are much fewer methods. This section surveys and investigate some heuristics and approximation methods.

In [33], the authors present the 0.878-approximation algorithm for the max-cut problem. Their main idea is to solve an SDP formulation and select a random hyperplane that passes through the origin, and partition the vertices $v_i \in V$ according to which side of the hyperplane they fall. This procedure is called randomized rounding. In [25] and [47] an extension of [33] is proposed to the max-k-cut problem.

In the SBC algorithm [31], the authors propose the iterative clustering heuristic (ICH) that finds feasible solutions based on the SDP solution. Computational tests show that the ICH provides better feasible solutions than those obtained by [33] (for $k = 2$) and by [25] (for $k \geq 3$). The ICH method aggregates information from an SDP solution (X^*). In summary, the ICH sums the X^*_{ij} values on the edges between three vertices, then, if the sum is greater than a constant (α), they set these three vertices in the same partition.

In [90], a multistart-type algorithm called dynamic convexized (DC) method is proposed. In the DC a local search algorithm is applied to a dynamically updated auxiliary function. Computational results demonstrate that the proposed algorithm is efficient to find feasible solutions.

In [56], the authors present a multiple operator heuristic (MOH) for the max-k-cut problem. MOH is an iterative method that applies five operators organized in three search phases. The first phase is called the descent-based improvement phase that finds a local optimum solution with two intensification oriented operators. After that, the solution is diversified in the improvement phase where the tabu search method [32] is applied with two operators. Finally, in the last phase, a random perturbation is applied to change the incumbent solution. Computational tests show that MOH provides better bounds in less time than the DC method in 90% of their tested instances.

The variable neighborhood search (VNS) [62] and the greedy randomized adaptive search procedure (GRASP) [23] are also used to find a feasible solution for max-k-cut in our branch-and-bound algorithm. VNS and GRASP are applied with success on the max-cut problem in [24].

The VNS heuristic is a iterative method that explores a distant neighborhood of the current incumbent solution and jumps from this solution to a new one if there is an improvement. In addition, a local search is applied to find a new local optimum.

The GRASP is also an iterative method that is divided in three phases: construction, local search and solution analysis. In the construction phase it generates a solution using a random greedy algorithm, then it applies a local search, and it saves the best solution found.

Implementation details of all the methods tested are described below with computational tests in several instances.

Implementation details: lower bounds

This section presents the implementation details of the four heuristics: ICH, MOH, VNS, and GRASP. Some parameters are different from the ones proposed in the literature to obtain a good trade-off between the quality and the CPU time of each method.

We consider that the solutions x and y are neighbors if the distance between these two solutions is one ($d(x, y) = 1$), i.e., if only one vertex of x is assigned to a different partition than y. A local search with simple-transfer moves, at each iteration, to a neighbor with a better solution. If any neighbor has a better solution, then we are in a local maximum. In double-transfer we consider the best neighbor with distance equal to 2. The following describes the parameters and details of implementation of each heuristic.

1. *ICH*. For the ICH, we impose the parameter $\alpha = 2.6$, and when the re-optimization is called for solving a derivate subgraph, the total time of CPA is limited to 20 seconds.

2. *MOH*. In the improvement phase (second phase), only the local search with simple-transfer (called O_3 in [56]) is considered because the fourth operator (O_4) is quite expensive. In addition, it is imposed:

   - tabu tenure is equal to 10,
   - number of iterations is 50, and
   - the perturbation strength that measure the perturbation allowed in third phase of the method is fixed in 50%.

3. *VNS*. As defined before, the VNS explores a distant neighborhood, from a local maximum x^*. Our VNS considers y as a neighbor solution of x^* if their distance is inferior to d_{max} ($1 \leq d(x^*, y) \leq d_{max}$). For VNS it is imposed:

   - $d_{max} = 12$ if number of vertex of graph is inferior than 300 ($|V| \leq 300$) and $d_{max} = 8$ otherwise, and
   - number of iterations is 10.

4. *GRASP*. The local search uses single and double-transfer moves to find the best neighbor solutions of a random solutions constructed in the construction phase of the method. Moreover, the maximum number of iterations is set to 50.

Computational results: lower bounds

The Table 6.2 shows the results of the four heuristic methods for $k \in \{3, 7\}$. Each row of this table plots the average of 5 rounds of tests. In Table 6.2, the name, k, dimension $|V|$ and density ($dens.$) of each instance is shown in the first columns. Then, for each method, the average solution (lb_{avg}) and the time of execution in seconds (time(s)) are presented.

Table 6.2 Results of four heuristic methods for finding feasible solutions.

Instance				ICH		MOH		VNS		GRASP	
name	k	$\|V\|$	dens.	lb_{avg}	time(s)	lb_{avg}	time(s)	lb_{avg}	time(s)	lb_{avg}	time(s)
bqp	3	50	0.1	**3271**	0.2	3252	0.1	3266	0.1	3244	0.1
rand	3	100	0.9	222056	0.7	241249	0.4	**243986**	0.4	240241	0.3
be	3	150	0.8	19682	2.9	**20547**	1.3	20500	0.9	20425	0.9
ising2	3	250	0.9	813412	6.6	808650	3.6	**814786**	2.8	799392	1.9
rand	3	300	0.2	52943	9.7	53963	5.4	**54467**	4.7	53656	4.2
bqp	7	50	0.1	3259	0.2	3352	0.1	**3356**	0.1	3315	0.1
rand	7	100	0.9	241500	0.3	248469	0.7	**249808**	0.5	248717	0.4
be	7	150	0.8	21086	3.2	21085	1.5	**21330**	1.1	21073	1.0
ising2	7	250	0.9	**820979**	6.6	815249	4.6	820424	4.7	809997	2.2
rand	7	300	0.2	52724	9.6	55128	6.1	**56812**	5.9	55679	4.7

Table 6.2 shows that the VSN heuristic gives, for almost all the tests, the best bound with competitive CPU time.

6.4 Selection and branching strategies

The three algorithmic components (e.g., split, branching and search strategies) play a very important role in the performance of the branch-and-bound algorithm. This section aims to present and discuss results of these components.

6.4.1 Splitting problem

The branch-and-bound algorithm converges if the size of each subproblem is smaller than the original problem and if the original problem has finite solutions. Typically, the subproblems generated

are disjoint thereby avoiding the occurrence of the same solution in different subspaces. The problem can be split into two (dichotomic branching) or more parts (polychotomic branching).

Dichotomic split. For the max-cut problem ($k = 2$) the split is usually performed by choosing a vertex $i \in V$ not yet assigned at the original node (see e.g. [74]), hence, for each subproblem generated the vertex is designated to a different partition. However, for the max-k-cut, all the exact methods proposed in the literature apply the split on an edge $\hat{x}_{ij}$, so they impose $\hat{x}_{ij} = \lfloor \hat{x}_{ij} \rfloor$ to one subproblem and $\hat{x}_{ij} = \lceil \hat{x}_{ij} \rceil$ to the other.

The dichotomic branching has the advantage of being easily implementable. However, this branching for the max-k-cut has the drawback of generating subproblems that can violate some valid inequalities. Therefore, the branch-and-bound can spend time by storing and evaluating a node that is not feasible.

k-chotomic split. In order to avoid infeasible subproblems, a vertex branching (similar to the split applied in the max-cut problem) is introduced for the max-k-cut. For each subproblem i, a selected vertex $v \in V$ is fixed in a different partition $P_i \in \{P_1, P_, \ldots, P_k\}$. Thus, a problem (node) of the branch-and-bound tree is split into k subproblems (children). This polychotomic branching is called k-chotomic split.

A drawback of applying the k-chotomic branching is that the number of unexplored nodes in the branch-and-bound tree can grows much faster than the dichotomic strategy and it can cause memory issues. Although, a combination of this strategy with depth-first search (see Section 6.4.3) can mitigate this drawback.

The next two sections computationally compare both types of splits.

Implementation details: splitting

This section discusses some implementation details for applying the k-chotomic and the dichotomic split on our branch-and-bound framework.

Fixing extra variables. For both strategies, in the LP formulation, the reduced cost of the bound solution is used to fix variables. Moreover, for the dichotomic split, when a variable is selected it is possible to fix more variables by analyzing the Triangle Inequalities (6.6). For example, if an edge x_{ih} was already fixed (in a related node), and the chosen variable in the current iterate is x_{hj}, then x_{ij} is fixed if:

$$x_{ih} = 1 \quad \& \quad x_{hj} = 1 \quad \Rightarrow \quad x_{ij} = 1, \tag{6.10}$$

$$x_{ih} = 1 \quad \& \quad x_{hj} = 0 \quad \Rightarrow \quad x_{ij} = 0, \tag{6.11}$$

$$x_{ih} = 0 \quad \& \quad x_{hj} = 1 \quad \Rightarrow \quad x_{ij} = 0. \tag{6.12}$$

For avoiding infeasible solutions in the dichotomic split, it is necessary to certificate that all clique Inequalities (6.7) are also satisfied at each subproblem. However, as it was pointed out in [76], it is computationally very expensive to enumerate all the cliques even for small instances.

Note that for k-chotomic strategy in max-k-cut problem a vertex can initially be fixed to the first partition (P_1) because all the vertex should be in a partition and all the partitions are similar. Then, in this work, the vertex $v \in V$ chosen to always be in P_1 is the one with the largest sum of incident weight edges in the input graph ($v \in \text{argmax}_{v \in V} \sum_{j=1}^{n} w_{vj}$).

Parameters of branch-and-bound. In order to provide a fair comparison, the following parameters are fixed for both strategies:

- *feasible solution*: the best solution is set as the initial feasible solution. Hence, there is no need of using heuristic methods,
- *branching rule*: the rule R4 is used to find variable (see Section 6.4.2),
- *selecting node*: the BeFS is fixed as the separation strategy (see Section 6.4.3),
- the branch-and-cut algorithm is used to solve the problems, and
- *time*: the algorithms is stopped after one hour.

Computational results: splitting

Table 6.3 shows results of the dichotomic and k-chotomic strategies. The first columns of Table 6.3 identify the instances, and for each strategy it is shown the time in seconds (time(s)) and the number of visited nodes ($\#nodes$).

Table 6.3 shows that the k-chotomic strategy has better results for almost all the problems. Moreover, from six out of all the twelve problems, the dichotomic strategy did not find the optimal solution before one hour of computation.

6.4.2 Branching Rules

As it is pointed out in [55], the variable and node selections are both critical decisions that impact the performance of branch-and-bound methods. For example, branching on a variable that does

Table 6.3 Results for dichotomic and k-chotomic strategies in the branch-and-bound framework for $k \in \{3, 5\}$.

Instance				dichotomic		k-chotomic	
name	k	$\|V\|$	dens.	time(s)	$\#nodes$	time(s)	$\#nodes$
data_random	3	30	1.0	-	-	**718.3**	3319
data_clique	3	40	1.0	20.4	92	**15.6**	83
random	3	25	0.5	**24.4**	76	42.4	296
data_3pm	3	60	0.1	-	-	**100.1**	1226
data_3g	3	64	0.1	449.1	128	**22.1**	244
random	3	70	0.1	-	-	**715.69**	750.0
data_random	5	30	1.0	-	-	-	-
data_clique	5	40	1.0	40.5	145	**20.6**	102
random	5	25	0.5	-	-	**73.7**	8757
data_3pm	5	60	0.1	0.0	0	0.0	0
data_3g	5	64	0.1	-	-	**106.7**	9504
random	5	70	0.1	75.6	8200	**10.5**	750

not give any serious improvement on any of the children nodes can lead to an extremely large and expensive search tree. The branching procedure aims to decide on which variable to branch to create children nodes.

Many researchers have investigated the branching in the branch-and-bound framework, see e.g. [1]. In particular, the *strong branching* strategy (e.g. [27, 53]) tests all the fractional candidates to find the one that gives the best improvement. Another very popular branching is the *pseudo-cost branching* [7] that uses historical information of previous changes to choose the variable. In [1], the authors present the *reliability branching* as a generalization of strong and pseudo-cost branching strategies for branch-and-bound algorithms. Their computational results show that this rule is better than the strong and pseudo-cost branching.

In [4], four branching rules proposed in [42] are investigated for the `BundleBC` method. In the first rule, the variable selected is the most decided, i.e, the edge that is the closest to 0 or to 1. The second rule is more elaborated, it selects the edge of vertices i' and j' such that:

$$i' \in \operatorname*{argmin}_{i \in V} \sum_{r \neq i, r=1}^{n} (0.5 - |x^*_{ir} - 0.5|)^2, \tag{6.13}$$

$$j' \in \operatorname*{argmin}_{j \in V, j \neq i'} \sum_{r \neq j, r=1}^{n} (0.5 - |x^*_{jr} - 0.5|)^2. \tag{6.14}$$

In the third rule, the variable chosen is the edge least decided. The fourth rule is similar to second

but for the vertex j' the argmin is replaced for argmax. Computational results show that the second rule provides the best results and the first rule gives the worse.

In the k-chotomic split strategy, we state that a vertex $v \in V$ is already fixed in a node $\mathbf{Q}$ if v was previously assigned to one of the partitions $v \in \{P_1, P_2, \ldots, P_k\}$ in a related node of the search tree. Base on previous results presented in the literature, the following rules are investigated:

- **R1.** In this rule, the variable chosen is the *most decided variable*. For the LP formulation that has solution $\hat{x}$, it will branch on the variable $\hat{x}_{ij}$ that is the closest to 0 or to 1 for all edge $(i, j) \in E$. For the k-chotomic split, the vertex j of $\hat{x}_{ij}$ must be already assigned to a partition.

- **R2.** This rule is similar to the third rule of [4], then it consists of choosing the edge that is the *least decided*. Similarly to R1, while applying this rule in the k-chotomic split, we impose that one of the vertices is fixed in a partition.

- **R3.** This rule is similar to the best rule of [4]. For the k-chotomic split, we impose the vertex $j' \in V$ to be already fixed and the selected variable $i' \in V$ comes from Argument (6.13).

- **R4.** In this rule, the selected variable is the edge with the largest weight. For the k-chotomic split, the vertex with the largest sum of incident edges is selected.

- **R5.** The last rule is an adaptation of the *reliability branching* of [1]. This rule uses the pseudo-cost score of a variable $\hat{x}_{ij}$. The pseudo-cost is estimated by summing all the improvements of previous branches when the variable was selected, and if the number of times that the variable $\hat{x}_{ij}$ was selected is inferior to a reliability constant $(\eta = 4)$, it uses the strong branching strategy that tests the improvement of a variable by simulating a branching.

The next section shows the performance of each branching rule in our method.

Computational results: branching rules

Table 6.4 shows the result of five branching rules. The first columns of the table identify the instances and for each rule, the time in seconds (time(s)) and the number of visited nodes ($\#nodes$).

In Table 6.4, the results of all the rules are quite similar. However, the rule R4 has the best values. For example, for $k = 5$ and $|V| = 25$, the rule R4 finds the optimal solution in 77 seconds while the others need at least 400 seconds.

Table 6.4 Comparative results for five branching rules in the branch-and-bound framework.

Instance			R1		R2		R3		R4		R5	
k	$\|V\|$	dens.	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$
3	30	1.0	472	2254	635	3011	625	3267	640	3319	**384**	1612
3	40	1.0	17	122	25	185	**15**	98	16	83	44	110
3	25	0.5	30	35	32	103	**30**	32	31	119	34	20
3	64	0.1	29	100	27	50	24	50	**11**	50	30	100
3	60	0.1	305	3650	274	3578	1132	15986	**100**	1226	268	3614
3	70	0.1	1844	4067	**676**	716	-	-	750	701	-	-
5	30	1.0	-	-	-	-	-	-	-	-	-	-
5	40	1.0	**16**	82	25	140	18	100	21	102	71	89
5	25	0.5	409	33880	464	38007	-	-	**77**	7209	400	32952
5	64	0.1	468	2600	293	1750	-	-	**220**	1753	511	5164
5	60	0.1	0	0	0	0	0	0	0	0	0	0
5	70	0.1	-	-	254	1100	711	6889	**193**	2000	766	4450

6.4.3 Node selection

Node selection is the classical task of deciding how to explore the branch-and-bound tree by using some strategies for selecting an unexplored node to be analyzed. Depending on the strategy one of the following issues is favored: the number of explored nodes in the search tree or the memory capacity of the computer.

The best-first, the depth-first, and the breadth-first are among the most commonly used strategies in existing optimization codes. Other variants of these strategies are studied in the literature see e.g. the surveys [55, 64].

The following three strategies to select active nodes on the branch-and-bound algorithm are studied:

- *Best-First Search (BeFS).* This strategy selects the best node not yet explored. The BeFS selects a node with the largest upper bound, so with the greatest potential improvement of the objective value. Usually, BeFS explores fewer nodes than other strategies but it maintains a larger tree in terms of memory.

- *Depth-First Search (DFS).* The DFS strategy uses the data structure of last-in, last-out to select a node. Therefore, the method goes deep in the search tree and only starts backtracking if a node is pruned. This strategy is easily implementable, the memory requirements are low and it finds feasible solutions faster than other strategies. A disadvantage is that this method is very sensitive to the branching rule of the first nodes in the branch-and-bound tree.

- *Breadth-First Search (BrFS).* The BrFS is the opposite of the DFS strategy since it is implemented with a first-in, first-out data structure. The BrFS operates well on unbalanced search

trees and finds solutions that are close to the root node. However, like BeFS, the memory requirements to store the unexplored nodes are quite high and they depend on heuristic methods for finding feasible solutions.

Computational results: node selection

Table 6.5 shows the results of the three strategies. The first columns identify the instances, and other columns show the time in seconds (time(s)) and the number of visited nodes ($\#nodes$) in the branch-and-bound tree of each strategy.

Table 6.5 Comparative results of node selection strategies: BeFS, BrFS and DFS.

Instance				BeFS		BrFS		DFS	
name	k	$\|V\|$	dens.	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$
data_random	3	30	1.0	640.1	3319	637.7	3319	**633.0**	3319
data_clique	3	40	1.0	15.6	83	14.7	83	**14.2**	83
random	3	25	0.5	29.4	94	**28.0**	94	38.3	188
data_3pm	3	60	0.1	100.1	1226	**78.9**	1205	102.96	1586
data_3g	3	64	0.1	**10.8**	50	**10.8**	50	19.3	75
random	3	70	0.1	750.0	701	546.37	779	**382.9**	980
data_random	5	30	1.0	-	-	-	-	-	-
data_clique	5	40	1.0	20.6	102	19.7	102	**19.4**	102
random	5	25	0.5	76.7	7209	**76.4**	7209	76.5	7209
data_3pm	5	60	0.1	0.0	0	0.0	0	0.0	0
data_3g	5	64	0.1	**220.18**	1753	311.1	2180	242.2	1993
random	5	70	0.1	193.4	2000	298.9	2965	**138.3**	1844

Table 6.5 shows that the results of these strategies are very similar. For the most difficult instances, where the time of solutions is superior to 300 seconds, the DFS strategy was the best, but for "easy" problems the BrFS was the best.

6.5 Computational environment and instances

Tests are performed on a Linux PC with two Intel(R) Xeon(R) 3.07 GHz processors. The `mosek` solver [5] is used for solving the SDP and LP formulations.

Our branch-and-bound is tested in some of the most difficult instances from the literature, especially from the `BundleBC` [4]. We also generate some instances as it is describe bellow:

- Random: these instances were generated using `rudy` graph generator [75]. The density varies between 10% to 100%.

- Data: we also consider instances from [4] for our numerical experiments. Some of the instances are from applications in statical physics and other were generated also using `rudy`. The name of these instances starts with the keyword "data".
- In order to test larger instances in the lower bound test (see Section 6.3.2), we also use some instances from [77]. They are bqp, ising2, be, rand.

6.6 Computational results

This section presents the computational tests. Section 6.6.1 highlights the importance of the changes in LP from [76] and [77]. In Section 6.6.2, results of the SDP and *LP-EIG* formulations in the branch-and-bound algorithm are plotted in Table 6.7.

6.6.1 Comparison of the linear formulations in the branch-and-bound framework

Most of the exact methods proposed in the literature for the linear formulation of the max-k-cut problem use triangle and clique inequalities to strength the LP bound (see e.g. [85, 60]). However, computational tests in [76] show that the use of wheel, bicycle wheel, and general clique inequalities can improve even more the bounds of max-k-cut.

The following linear methods are investigated in order to highlight the relevance of the inequalities studied in [76] and the SDP-based Inequalities (6.8) in the branch-and-bound framework:

- *LP-tc*: in this method, the linear relaxation solved in the bounding procedure of the branch-and-bound framework activates only triangle and clique inequalities.
- *LP-comb*: in this method, the following five constraints are activated: triangle, clique, wheel, bicycle wheel and general clique. Those are the combinatorial inequalities investigated in [76].
- *LP-EIG*: this method includes, in addition to the five combinatorial inequalities, the SDP-based Inequalities (6.8) in the LP relaxations.

Table 6.6 shows the time in second (time(s)) and the number of visited nodes ($\#nodes$) for the three linear formulations in a branch-and-cut framework. The first columns of this table show the name, k, dimension $|V|$ and density (dens.) of each instance. The branch-and-cut method is stopped after two hours and we report the results of instances that have a certificate of optimality, i.e,. the optimal solution is found. Otherwise, we insert the signal "-".

Table 6.6 demonstrates that, in general, the *LP-EIG* method explores fewer nodes in the branch-and-bound tree. For $k = 3$, the *LP-comb* is most of the time better than *LP-tc* and sometimes this

Table 6.6 Comparison of the linear formulations in the branch-and-bound framework.

Instance				*LP-tc*		*LP-comb*		*LP-EIG*	
name	k	$\|V\|$	dens.	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$
data_2g_7_491	3	30	0.1	9.2	420	4.1	140	**2.5**	0
data_2g_8_648	3	64	0.1	833.7	22549	191.9	2938	**181.0**	2576
data_2g_9_9211	3	81	0.1	1955.4	46725	591.9	8400	**591.4**	7200
data_3g_444_444	3	81	0.1	**196.1**	2695	3728.5	52962	-	-
data_3pm_345_345	3	81	0.1	870.3	12552	**77.9**	1165	86.7	960
data_clique_40	3	40	1.0	20.0	26	8.1	8	**2.5**	0
data_clique_50	3	50	1.0	108.8	197	**25.1**	23	48.5	35
data_clique_60	3	60	1.0	203.9	146	113.3	20	**28.7**	12
random	3	40	0.3	714.7	6665	46.4	219	**40.4**	95
random	3	30	0.5	135.5	586	**47.1**	160	66.2	35
random	3	40	0.5	-	-	893.1	168	**357.8**	25
random	3	30	1.0	807.0	807	**81.6**	136	102.3	80
random	3	40	1.0	571.9	435	455.2	152	**104.5**	83
data_2g_7_491	5	30	0.1	1.0	0	1.0	0	1.0	0
data_2g_8_648	5	64	0.1	140.0	605	4.0	60	**3.8**	20
data_2g_9_9211	5	81	0.1	-	-	-	-	**89.6**	920
data_3g_444_444	5	81	0.1	**264.1**	4302	-	-	-	-
data_3pm_345_345	5	81	0.1	69.2	741	152.2	1262	**25.4**	115
data_clique_40	5	40	1.0	-	-	-	-	**1.0**	0
data_clique_50	5	50	1.0	-	-	-	-	**2.8**	0
data_clique_60	5	60	1.0	-	-	-	-	**21.3**	0
random	5	40	0.3	271.8	3958	625.4	4733	**240.7**	1613
random	5	50	0.3	-	-	2463.1	8106	**2263.9**	856
random	5	30	0.5	67.7	385	97.4	150	**42.6**	54
random	5	40	0.5	-	-	3334.3	4486	**2012.5**	1073
random	5	30	1.0	159.5	525	142.7	476	**130.1**	233
random	5	40	1.0	698.6	520	743.1	465	**562.5**	337

method is better than *LP-EIG*. However, usually, the *LP-EIG* gives best values, for example, for four instances, only the *LP-EIG* method was capable of finding the optimal solution.

6.6.2 Comparison of *LP-EIG* versus SDP

Table 6.7 displays the results of branch-and-bound with *LP-EIG* and with the SDP formulations. It shows the time in second (time(s)) and the number of visited nodes ($\#nodes$) for both methods and for the `BundleBC` solver. The first columns of this table give the name, k, dimension $|V|$ and density (dens.) of each instance.

The branch-and-cut method of *LP-EIG* and SDP is stopped after two hours and Table 6.7 exhibit

only the results that have a certificate of optimality, i.e,. the optimal solution is found. Otherwise the symbol "-" is reported.

Table 6.7 Results of the *LP-EIG* and SDP formulations in the branch-and-bound framework.

Instance			BundleBC [4]		*SDP*		*LP-EIG*	
name	k	$\|V\|$	time(s)	$\#nodes$	time(s)	$\#nodes$	time(s)	$\#nodes$
data_random_40_k=3	3	40	**29.0**	145	47.7	31	-	-
data_3pm_344_344	3	48	102.0	21	**4.3**	2	7.0	2
data_2g_7_491	3	49	5.0	3	41.3	60	**2.5**	0
data_random_50_k=3	3	50	3963.0	10741	**3391.7**	1794	-	-
data_clique_60	3	60	**3.0**	3	32.4	0	28.7	12
data_2g_8_37	3	64	**111.0**	9	2029.0	1093	149.1	1786
data_2pm_8_888	3	64	116.0	7	63.1	21	**2.0**	20
data_3g_444_444	3	64	198.0	7	**174.3**	69	-	-
data_clique_70	3	70	**36.0**	11	161.5	8	68.5	16
data_2g_9_9211	3	81	**169.0**	5	-	-	801.0	50233
data_2pm_9_999	3	81	576.0	21	-	-	**29.3**	223
data_2g_10_1001	3	100	106.0	5	-	-	**46.8**	1357
data_random_40_k=3	5	40	7925.0	18243	**1998.7**	674	-	-
data_3pm_344_344	5	48	1618.0	205	1008.2	420	**35.2**	5788
data_2g_7_491	5	49	5724.0	3623	4.8	1	**1.0**	0
data_random_50_k=3	5	50	-	-	-	-	-	-
data_clique_60	5	60	7.0	3	95.0	4	**21.3**	0
data_2g_8_37	5	64	1979.0	261	345.4	300	**7.1**	96
data_2pm_8_888	5	64	245.0	11	5.7	2	**0.7**	0
data_3g_444_444	5	64	**979.0**	37	3689.2	595	-	-
data_clique_70	5	70	17.0	3	2036.0	54	**41.0**	0
data_2g_9_9211	5	81	523.0	13	-	-	**89.6**	920
data_2pm_9_999	5	81	2507.0	51	55.8	160	**3.4**	35
data_2g_10_1001	5	100	-	-	-	-	**94.2**	3580
data_random_40_k=3	7	40	-	-	-	-	-	-
data_3pm_344_344	7	48	1240.0	191	1126.8	438	**25.1**	3524
data_2g_7_491	7	49	-	-	3.6	0	**1.4**	1
data_random_50_k=3	7	50	-	-	-	-	-	-
data_clique_60	7	60	79.0	29	-	-	**61.4**	1120
data_2g_8_37	7	64	30370.0	6703	978.5	200	**11.2**	911
data_2pm_8_888	7	64	13.0	309	55.1	31	**0.7**	0
data_3g_444_444	7	64	**868.0**	29	947.4	77	-	-
data_clique_70	7	70	152.0	31	-	-	**20.6**	0
data_2g_9_9211	7	81	2418.0	65	-	-	**363.9**	19600
data_2pm_9_999	7	81	963.0	23	217.7	41	**2.4**	100
data_2g_10_1001	7	100	-	-	-	-	**508.8**	6200

Table 6.7 demonstrates that, in general, the SDP visits fewer subproblems than *LP-EIG*. For dense instances, when $k = 3$ the SDP outperforms *LP-EIG*. The *LP-EIG* outperforms SDP for the other

cases, especially for sparse problems. In 67% of the instances tested the SDP or *LP-EIG* are better than `BundleBC` especially for $k \geq 5$. Moreover, the *LP-EIG* has uncovered the solution of 3 problems.

In general, the *LP-EIG* outperforms the other methods for sparse instances and for large values of k.

6.6.3 Summary of the computational tests

The following are a summary of all the results presented in this chapter :

1. *cut-and-branch versus branch-and-cut.* In Section 6.3.1 shows that the branch-and-cut method outperforms the cut-and-branch method in 6 out of 11 tests. However, the cut-and-branch is better for sparse instances and $k = 5$.

2. *Lower bound.* Computational tests in Section 6.3.2 demonstrates that the VNS obtains the best results.

3. *Splitting problem.* Section 6.4.1 shows a huge difference between the results of the dichotomic and the k-chotomic strategies. The k-chotomic is the best strategy in almost all instances. Moreover, the dichotomic strategy could not solve five instances that k-chotomic solved in less than 30 minutes.

4. *Branching rules.* There is not a lot of difference between the results of the five rules tested.

5. *Node selection.* Our tests are not conclusive because all the three strategies have similar results. However, the DFS has been the best strategy for the most difficult instances.

6. *LP comparison.* Our tests show the relevance of combinatorial inequalities {triangle, clique, general clique, wheel and bicycle wheel } and especially the relevance of the proposed SDP-based Inequality 6.8 on max-k-cut.

7. *LP versus SDP.* Our last tests compared the performance of *LP-EIG* with SDP. *LP-EIG* outperforms the SDP based branch-and-bound in most of the problems, especially for sparse and for $k \geq 5$.

From a practical point of view, the best arrangements are: lower-bound = VNS, splitting = k-chotomic, branching rule=R4 (Node with the largest incident weight), node selection=BrFS or DFS.

6.7 Discussion

This work investigates the branch-and-bound method to solve the max-k-cut problem where we computationally study all its features to design an efficient method.

In the upper bound procedure, we study the SDP and LP relaxation and we also investigated the inclusion of the CPA in the branch-and-bound framework. For the lower bound, we studied four heuristic methods: ICH, MOH, VNS, and GRASP.

We also investigate the selection and branching strategies. For the split strategy, we consider two options: the dichotomic split where each selected node derivate two other subproblems, and the k-chotomic strategy where each node generates k subproblems. Five rules for choosing the next variable to be branched is studied in Section 6.4.2. Finally, we study three strategies for the node selection: BeFS, DFS, and BrFS.

We conclude from our computational tests that the proposed inequalities in [76, 77] are quite relevant in the branch-and-bound scheme for the max-k-cut problem, especially SDP-based inequalities. Moreover, our computational tests show that for many instances our method is faster than the ones proposed in the literature.

CHAPTER 7 GENERAL DISCUSSION

This chapter discusses the results of the three following projects of our research: the computational study of valid inequalities presented in Chapter 4, the improvement of the linear relaxation studied in Chapter 5, and the branch-and-bound investigated in the previous chapter.

The first objective of this thesis was to study some classes of valid inequalities that are facet-defining for the max-k-cut problem. We designed and tested several separation routines to find violations in an iteration of the cutting plane method. For the separation routines, we noticed that some heuristic methods are more efficient than their associated exact methods proposed in the literature. For example, for the bicycle wheel inequalities, the exact method spends almost 50% more CPU time to find violations that are as strong as the ones found by our proposed heuristic. Computational tests presented in [76] showed that the wheel and bicycle wheel inequalities are very relevant for strengthening max-k-cut relaxations.

In our second article [77], the main objective was to introduce a formulation of max-k-cut that is as fast as the linear formulation and as strong as the SDP relaxation. Hence, we investigated the semi-infinite formulation of SDP that provided a strong class of constraints to the LP relaxation. This family of constraints was called SDP-based inequalities and our computational results showed that the new linear relaxation outperforms SDP for problems that have a large number of partitions ($k \geq 7$). Moreover, the LP relaxation with SDP-based inequalities has shown to be convenient for exact methods because it provides strong and inexpensive bounds.

In Chapter 6, the concepts presented in [76] and [77] are applied in an exact method to find global solutions of the max-k-cut problem. In order to conceive an efficient solver, we investigated five components of the branch-and-bound algorithm. We observed that some procedures have a huge impact on the performance of the branch-and-bound method, for example, the k-tomic strategy has solved 75% of the problems in less CPU time than the dichotomic strategy for $k \in \{3, 5\}$. Computational results showed that the investigation carried out on [76] is very relevant to the branch-and-bound method, and with the new linear relaxation proposed in [77], we were able to uncover many optimal solutions.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

This last chapter summarizes the advancements presented in this thesis, the limits and constraints of application of our methods and the main recommendations for future research.

8.1 Advancement of knowledge

In summary, in this thesis we made the following advances:

- We investigated the triangle, clique, wheel, bicycle wheel and general clique inequalities. Moreover, we introduce some efficient separation routines for each family of inequalities.
- We introduce, in our second paper [77], the SDP-based inequalities that have improved the linear relaxation.
- For the exact method, we investigated carefully the branch-and-bound method.
- We proposed the use of the VNS [62] and the GRASP [23] heuristic to find feasible solutions of the max-k-cut. Computational results have shown that these methods, mainly the VNS, provide better solutions than the others heuristics proposed in the literature.

8.2 Limits and constraints

The following are the main limitations and drawbacks of the concepts presented in this thesis:

- The SDP-based inequalities proposed in [77] are very large and dense, in the sense that they have approximately $|E|$ variables with non-zero coefficients. Therefore, the addition of a few of these inequalities can make the solution of the linear relaxation very expensive. For example, in some cases, the LP with SDP-based inequality can be more expensive than the SDP formulation.
- Based on empirical tests, we fixed many parameters for the cutting plane algorithm and for the branch-and-bound method. Whereas some problems can behave differently, we do not change the parameters for each specific problem, and we do not consider the fact that some parameters may adapt with the evolutions of the methods.
- In order to obtain upper bound we solve our relaxations using `mosek` solver [82]. This makes our method very dependent on `mosek` performance.

- Although the proposed exact method is able to uncover the optimal solution for some problems up to 200 nodes we still cannot solve larger and dense instances.

8.3 Recommendations

The following are recommendations for future research concerning methods investigated in this thesis:

- *More compact SDP-based inequalities.* In order to overcome the issue of large SDP-based inequalities, we may use the dual solution of the SDP relaxation instead of the primal, as it is pointed out in [50]. Because the dual formulation has fewer variables than the primal, therefore it provides a more compact inequality.
- *Optimization of parameters.* Applying machine learning techniques [55] for the branch-and-bound and the CPA may improve even more our results by optimizing some of their parameters.
- *Hybrid branch-and-bound.* A branch-and-bound method that uses both SDP and LP relaxations can be useful because it may exploit the best qualities of each relaxation in the same branch-and-bound tree.
- *Extension.* With a few modifications, the proposed linear relaxation can be extended to solve general SDP problems. Hereby, introducing a mixed integer programming tool for a generic SDP problem.

REFERENCES OF BIBLIOGRAPHY

[1] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42 – 54, 2005.

[2] 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.

[3] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[4] 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.

[5] Mosek ApS. mosek. http://www.mosek.com, 2015.

[6] 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.

[7] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[8] J.-D. Cho, S. Raje, and M. Sarrafzadeh. Fast approximation algorithms on maxcut, k-coloring, and k-color ordering for vlsi applications. *IEEE Transactions on Computers*, 47(11):1253–1266, 1998.

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

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

[11] A. Coja-Oghlan, C. Moore, and V. Sanwalani. Max k-cut and approximating the chromatic number of random graphs. *Random Structures & Algorithms*, 28(3):289–322, 2006.

[12] W.-M. Dai and E.S. Kuh. Simultaneous floor planning and global routing for hierarchical building-block layout. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(5):828–837, 1987.

[13] G. B. Dantzig. *Linear programming and extensions*. 1963.

[14] 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.

[15] M. Deza, M. Grötschel, and M. Laurent. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, 17(4):981–1000, 1992.

[16] M. Deza and M. Laurent. *Geometry of cuts and metrics*. Springer Publishing Company, Incorporated, 1st edition, 2009.

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

[18] H.A. Eiselt and C.-L. Sandblom. *Linear programming and its applications*. Springer-Verlag Berlin Heidelberg, Berlin, Germany, 1st edition, 2007.

[19] A. Eisenblätter. *The semidefinite relaxation of the k-partition polytope is strong*, volume 2337 of *Lecture Notes in Computer Science*, pages 273–290. Springer Berlin Heidelberg, 2002.

[20] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736.

[21] J. Fairbrother and A. N. Letchford. Projection results for the k-partition problem. *Discrete Optimization*, 2017.

[22] 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.

[23] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[24] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 17(6):1033–1058, 2002.

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

[26] G. Gamrath. Improving strong branching by propagation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 347–354, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[27] G. Gamrath. Improving strong branching by domain propagation. *EURO Journal on Computational Optimization*, 2(3):99–122, 2014.

[28] A. R. García, I. S. González, C. H. Goya, and P. C. Gil. IBSC system for victims management in emergency scenarios. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security, IoTBDS 2017, Porto, Portugal, April 24-26, 2017*, pages 276–283, 2017.

[29] D. Gaur, R. Krishnamurti, and R. Kohli. The capacitated max k-cut problem. *Mathematical Programming*, 115(1):65–72, 2008.

[30] A. M. H. Gerards. Testing the odd bicycle wheel inequalities for the bipartite subgraph polytope. *Mathematics of Operations Research*, 10(2):359–360, 1985.

[31] 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.

[32] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[33] 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.

[34] J. Gondzio. Interior point methods 25 years later. 218:587–601, 2012.

[35] J. Gondzio, P. González-Brevis, and P. Munari. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation*, 2016.

[36] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, 1989.

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

[38] P. Hansen and N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

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

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

[41] C. Helmberg. Semidefinite programming. *European Journal of Operational Research*, 137(3):461–482, 2002.

[42] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(2), 1998.

[43] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[44] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26(12):1519–1534, 2000.

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

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

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

[48] Y.-H. Kim, Y. Yoon, and Z. W. Geem. A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm and Evolutionary Computation*, 2018.

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

[50] 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.

[51] 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.

[52] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. *Computing exact ground states of hard ising spin glass problems by branch-and-cut*, pages 47–69. Wiley-VCH Verlag GmbH & Co. KGaA, 2005.

[53] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

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

[55] A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP*, 25(2):207–236, 2017.

[56] 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.

[57] I. Maros and G. Mitra. Strategies for creating advanced bases for large-scale linear programming problems. *INFORMS Journal on Computing*, 10(2):248–260, 1998.

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

[59] J. E. Mitchell. Branch-and-cut for the k-way equipartition problem, 2001.

[60] J. E. Mitchell. Realignment in the national football league: Did they do it right? *Naval Research Logistics*, 50(7):683–701, 2003.

[61] J. E. Mitchell, P. M. Pardalos, and M. G. C. Resende. *Interior point methods for combinatorial optimization*, pages 189–297. Springer US, Boston, MA, 1999.

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

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

[64] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79 – 102, 2016.

[65] 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.

[66] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. Society for Industrial and Applied Mathematics, 1994.

[67] V. Nikiforov. Max k-cut and the smallest eigenvalue. *Linear Algebra and its Applications*, 504:462 – 467, 2016.

[68] 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.

[69] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[70] 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. Springer, New York, 2011.

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

[72] D. Recalde, D. Severín, R. Torres, and P. Vaca. An exact approach for the balanced k-way partitioning problem with weight constraints and its application to sports team realignment. *Journal of Combinatorial Optimization*, Feb 2018.

[73] F. Rendl. Semidefinite relaxations for partitioning, assignment and ordering problems. *Annals of Operations Research*, 240(1):119–140, 2016.

[74] 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.

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

[76] V. J. Rodrigues de Sousa, M. F. Anjos, and S. Le Digabel. Computational study of valid inequalities for the maximum k-cut problem. *Annals of Operations Research*, 265(1):5–27, 2018.

[77] V. J. Rodrigues de Sousa, M. F. Anjos, and S. Le Digabel. Improving the linear relaxation of maximum k-cut with semidefinite-based constraints. Technical Report G-2018-26, Gerad, 2018.

[78] J. K. Scholvin. Approximating the longest path problem with heuristics: A survey. Master's thesis, University of Illinois at Chicago, 1999.

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

[80] M. H. Seyed, H. T. Sai, and M. Omid. A genetic algorithm for optimization of integrated scheduling of cranes, vehicles, and storage platforms at automated container terminals. *Journal of Computational and Applied Mathematics*, 270:545–556, 2014. Fourth International Conference on Finite Element Methods in Engineering and Sciences (FEMTEC 2013).

[81] R. Sotirov. An efficient semidefinite programming relaxation for the graph partition problem. *INFORMS Journal on Computing*, 26(1):16–30, 2014.

[82] Development Core Team. *MOSEK modeling manual*. MOSEK ApS, 2013.

[83] E. R. van Dam and R. Sotirov. Semidefinite programming and eigenvalue bounds for the graph partition problem. *Mathematical Programming*, 151(2):379–404, 2015.

[84] E.R. van Dam and R. Sotirov. New bounds for the max-k-cut and chromatic number of a graph. *Linear Algebra and its Applications*, 488:216–234, 2016.

[85] G. Wang and H. Hijazi. Exploiting sparsity for the min k-partition problem. *ArXiv e-prints*, 2017.

[86] Z. Wang, H. Tian, K. Yang, and Z. Liu. Frequency resource allocation strategy with qos support in hybrid cellular and device-to-device networks. *International Journal of Communication Systems*, 28(7):1201–1218, 2014.

[87] T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex minlp problems. *Computers & Chemical Engineering*, 19:131 – 136, 1995.

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

[89] Q. Wu, Y. W., and Z. Lu. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, 34:827 – 837, 2015.

[90] W. Zhu, G. Lin, and M. M. Ali. Max-k-cut by the discrete dynamic convexized method. *INFORMS Journal on Computing*, 25(1):27–40, 2013.