# Computational study of a branching algorithm for the maximum $k$-cut problem

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

January 30, 2020

**Abstract.** This work considers the graph partitioning problem known as maximum $k$-cut. It focuses on investigating features of a branch-and-bound method to efficiently obtain global solutions. An exhaustive experimental study is carried out for two main components of a branch-and-bound algorithm: computing bounds and branching strategies. In particular, we propose the use of a variable neighborhood search heuristic to compute good feasible solutions, the $k$-chotomic strategy to split the problem, and a branching rule based on edge weight to select variables. Moreover, this work analyses the linear relaxation strengthened by semidefinite-based constraints, the cutting plane algorithm, and some node selection strategies. Computational results show that the method using the best procedures of the branch-and-bound outperforms the state-of-the-art and uncover the solution of several instances, especially for problems with $k \geq 5$.
**Keywords.** Maximum $k$-cut, branch-and-cut, graph partitioning, semidefinite programming, eigenvalue constraint.

**AMS subject classifications.** 90C57, 90C27

## 1 Introduction

The maximum $k$-cut (max-$k$-cut) problem is a graph partitioning problem in an undirected graph $G = (V, E)$ with weight on the edges. This problem aims to maximize the sum of weights that are cut, i.e, edges that have endpoints (vertices) in different partitions. Additionally, the vertex set $V$ can be partitioned into at most $k$ subsets.

The max-$k$-cut is a challenging combinatorial problem that is known to be in the class of $\mathcal{NP}$-complete problems [43] and it attracts scientific attention from the discrete community, e.g., [6, 22, 30, 44, 41, 14].

The objective of this work is a computational study of the main components of the branch-and-bound method to efficiently solve the max-$k$-cut problem to optimality. To compute upper bounds,

[*]Vilmar.de.sousa@gerad.ca GERAD and Département de Mathématiques et Génie Industriel, Polytechnique Montréal,

[†]www.miguelanjos.com School of Mathematics, University of Edinburgh

[‡]www.gerad.ca/Sebastien.Le.Digabel GERAD and Département de Mathématiques et Génie Industriel, Polytechnique Montréal,

we compare the performance of methods using semidefinite, linear relaxation and the impact of adding SDP-based inequalities [46]. For lower bounds, four methods are compared and two new heuristics to find good feasible solutions for the max-$k$-cut are proposed. For branching strategies, several ways of splitting, branching, and selection are proposed. To the best of our knowledge, no research has investigated all the components of the branch-and-bound for the max-$k$-cut problem.

This work is organized as follows. Section 2 reviews some techniques applied to obtain strong relaxations and to solve the problem. Section 3 introduces the general scheme of the branch-and-bound method and Section 4 presents the computational environment and the set of instances used in this work. Sections 5 and 6 show and discuss simultaneously the results of experimental tests with implementation details of each component of the branch-and-bound algorithm. Section 5 studies the three main aspects of the bounding procedure: relaxations, the cutting plane algorithm (CPA), and methods to find feasible solutions (lower bounds). Strategies to select variables and different techniques to explore the branch-and-bound tree are investigated in Section 6. A comparison study with the state-of-the-art is provided in Section 7. Finally, a discussion of results and conclusion are presented in Section 8.

# 2 Literature review

This section presents the most popular and recent methods proposed in the literature to solve the max-$k$-cut problem. It recalls some of the most important formulations of the max-$k$-cut and then it presents some solution approaches designed to find global solutions.

## 2.1 Formulations

This section presents three integer formulations of the max-$k$-cut. The first formulation considers only the edges of the graph as variables, the second takes into account the edges and the vertices, and the third formulation includes only vertex variables.

### 2.1.1 Edge-only formulation

A 0-1 edge formulation is proposed in [9], where the integer variable $x_{ij}$ for each vertex $i, j \in V$ 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

$$\max_{x} \sum_{i,j \in V, i<j} w_{ij}(1 - x_{ij}) \tag{1}$$

$$x_{ih} + x_{hj} - x_{ij} \leq 1 \qquad \forall i, j, h \in V, \tag{2}$$

$$\sum_{i,j \in Q, i<j} x_{ij} \geq 1 \qquad \forall Q \subseteq V \text{ with } |Q| = k+1, \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V. \tag{4}$$

where Constraint (2) and (3) are called triangle and clique inequalities, respectively. Triangle Inequalities (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. Constraints (3) impose that at least one edge in a clique with $k + 1$ vertices cannot be cut. For the edge-only formulation, the graph must be at least chordal otherwise dummy edges (of zero weight) must be added (see [50]).

### 2.1.2 Node-and-edge formulation

In [9], 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 $p \in \{1, \dots, k\}$. The binary variables are

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

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

$$\max_{x,y} \sum_{(i,j)\in E, i<j} w_{ij}(1 - x_{ij}) \tag{5}$$

$$\sum_{p=1}^{k} y_{vp} = 1 \qquad \forall v \in V\ , \tag{6}$$

$$x_{ij} \geq y_{ip} + y_{jp} - 1 \qquad \forall((i,j) \in E, p = 1, \dots, k), \tag{7}$$

$$x_{ij} \leq y_{ip} - y_{jp} + 1 \qquad \forall((i,j) \in E, p = 1, \dots, k), \tag{8}$$

$$x_{ij} \leq -y_{ip} + y_{jp} + 1 \qquad \forall((i,j) \in E, p = 1, \dots, k), \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall(i,j) \in E, \tag{10}$$

$$y_{vp} \in \{0, 1\} \qquad \forall(v \in V, p \in \{1, \dots, k\}). \tag{11}$$

Constraints (6) imposes that all vertices must be in one partition and Constraints (7) to (9) ensure that edges are cut when their vertices are in different partitions. Constraints (8) and (9) may be removed when all edges weight of an instance are non-negative.

### 2.1.3 Node-only formulation

The third formulation is based on the vertices of the graph only. In [17], the authors mention that just allowing the variables to be $x_i \in \{1, \dots, k\}$ for each vertex $i \in V$ does not generate useful integer formulations. However, if the variable $x$ is one of the $k$ vectors $a_1, a_2, \dots, a_k$ in domain $\mathbb{R}^{k-1}$ such that the dot product is:

$$a_i^T a_j = \frac{-1}{k-1}$$

this product provides a convenient quadratic formulation. In Lemma 4 of [17], it is proved that the value $-1/(k-1)$ gives the best angle separation for $k$ vectors. Then, for $x_i \in \{a_1, \dots, a_k\}$:

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

Therefore, the following quadratic formulation of the max-$k$-cut is obtained:

$$\max_{x} \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - x_i \cdot x_j) \tag{12}$$

$$x_i \in \{a_1, a_2, \ldots, a_k\} \qquad \forall i \in V. \tag{13}$$

## 2.2 Solutions approaches: Branch-and-bound components

The branch-and-bound algorithm is widely applied in the literature to obtain global solutions of the max-$k$-cut problem and of several other combinatorial problems, for example, see survey [38].

Fundamentally, the branch-and-bound method is a tree search strategy that implicitly enumerates all the solutions. The algorithm starts from a node (*root node*) containing a relaxation of the max-$k$-cut problem. At each iteration $i$ of the method, a node $\mathbf{Q}_i$, not yet explored, is selected (*node selection*) and it checks 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 solution). Otherwise, it will generate children nodes (*branching*) with a more restricted relaxation of the problem. In a branch-and-bound algorithm, the upper and lower bounds are improved until optimality can be proved by a certificate.

### 2.2.1 Upper bound

This section presents the most popular and recent methods proposed in the literature to solve the max-$k$-cut problem. It recalls some linear programming (LP) and a semidefinite programming (SDP) relaxations, and the most important valid inequalities.

**Linear relaxation:** An advantage of applying a linear relaxation of the max-$k$-cut problem in a branch-and-bound framework is that the optimal solution $x^*$ of an LP can easily be used for *fixing variables*. Let $e \in E$ be an index of , $z_{lb}$ is the value of the best known feasible solution, $d_e$ is 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 $x_e$ is zero 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 LP is that the simplex method is well suited for exploiting an advanced starting basis [34]. Therefore, solvers (e.g. `mosek` [5]) can use the solution of previous-selected nodes in the branch-and-bound tree to reduce the computing time.

The nonlinear terms in the edge-only and node-and-edge formulations are the binary Constraints (4) and (10)-(11), respectively. By relaxing these constraints, the edge-only and the node-and-edge linear formulations are obtained.

The edge-only linear formulation is obtained by relaxing Constraints (4). A disadvantage of this relaxation is the large number of inequalities: in a complete graph, it can have $3\binom{|V|}{3}$ triangle Inequalities (2) and $\binom{|V|}{k+1}$ clique Inequalities (3). Note that, in general, for sparse and non-chordal graphs, a chordal extension [24] still adds a lot of dummy edges.

Replacing Constraints (10) and (11) by their correspondent linear relaxations defines the linear node-and-edge formulation. In [14], the authors show that this relaxation is very weak and it suffers from symmetry. This formulation can be improved by the so-called representative formulations [2] where a representative variable is added to break the symmetry. However, this extended formulations can add several variables and constraints to the model.

A branch-and-bound framework based on the edge formulation of the max-$k$-cut is studied in [50]. 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 are capable of solving to optimality some sparse graphs with $n = 200$ and $k \in \{3, 4\}$ in a few seconds.

**Semidefinite relaxation:** A second way of solving the max-$k$-cut is using its SDP relaxation. An advantage of the SDP relaxation is that it provides strong bounds, especially because some hypermetric inequalities are implicit [12] in the SDP formulation. However, it comes with expensive processing time, see the recent analysis of [47, 3].

The SDP relaxation is obtained from the quadratic node-only formulation (12)-(13) by replacing Constraints (13) with

$$x_i \in B_n \quad \forall i \in V \tag{14}$$

where $B_n$ is the unit sphere in $n$ dimensions. A constraint $x_i \cdot x_j \geq \frac{-1}{k-1}$ for all $i, j \in V$ ought to be added to avoid solutions where $x_i \cdot x_j = -1$. Thereby, replacing the inner product $x_i \cdot x_j$ for $X_{ij}$ in the positive semidefinite matrix ($X \succeq 0$) gives the SDP formulation proposed in [17]:

$$\max_X \frac{(k-1)}{k} \sum_{i,j \in V, i<j} w_{ij}(1 - X_{ij}) \tag{15}$$

$$X_{ii} = 1 \qquad \forall i \in V, \tag{16}$$

$$X_{ij} \geq \frac{-1}{k-1} \qquad \forall i, j \in V, i < j, \tag{17}$$

$$X \succeq 0 \tag{18}$$

Although this formulation has only a few constraints, the $\frac{n(n-1)}{2}$ Constraints (17) can impact a lot the computing time. Therefore, in [25], the author indicates that it is more efficient to start only with Constraint (16) and to separate $X_{ij} \geq \frac{-1}{k-1}$ successively in a cutting plane algorithm. Many other alternative approaches are proposed in the literature, see for example [13, 10].

In [49], the authors use the eigenvalues of the weighted adjacency matrix of the graph to propose several strong bounds for max-$k$-cut. The bounds of [49] are further improved by the use of the smallest eigenvalue in [40] and later with the whole spectrum of the adjacent matrix in [4].

**Cutting plane:** A branch-and-bound algorithm with a CPA was introduced in [42] 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 to improve the relaxed solution.

In general, relaxations can be tightened, without removing any feasible solution, by the addition of valid cutting planes. In [8], the authors show that the following classes of inequalities (known as the combinatorial inequalities) defines facets for the max-$k$-cut when $k \geq 3$:

- The general clique inequalities have the form:

$$\sum_{i,j \in Q} x_{ij} \geq z \qquad \forall Q \subseteq V, |Q| = p \tag{19}$$

where $z = \frac{1}{2}t(t-1)(k-q) + \frac{1}{2}t(t+1)q$ and $p := tk + q$ such that $t$, $q$ are positive integer numbers.

- The wheel inequalities have the form:

$$\sum_{(ij) \in O} x_{ij} - \sum_{(ij) \in C} x_{ij} \leq \left\lfloor \frac{|C|}{2} \right\rfloor \tag{20}$$

for all $O$, $C$ forming a wheel in $G$ and where $C$ is an odd cycle with $|C| \geq 3$ and $O$ is the set of edges linking the hub vertex to the vertices inside the cycle.

- The bicycle wheel inequalities have the form

$$\sum_{(ij) \in O} x_{ij} - \sum_{(ij) \in C} x_{ij} - x_{u_1 u_2} \leq 2 \left\lfloor \frac{|C|}{2} \right\rfloor \tag{21}$$

for all $O$, $C$, $(u_1 u_2)$ forming a bicycle wheel in $G$ with $|C|$ odd and $|C| \geq 3$.

These combinatorial inequalities can be applied in both the edge-only and node-and-edge formulations because they consider only the edges variables. In [13], the authors obtained, via the projection of these "edge-only" inequalities, new families of valid inequalities for the node-and-edge relaxation. However, the authors are uncertain of the practical use of these inequalities.

In [47], the authors proposed an SDP-based inequality that is represented as

$$\sum_{i,j \in V, i<j} \mu_i \mu_j x_{ij} \geq \frac{1}{k} \sum_{i,j \in V, i<j} \mu_i \mu_j - \frac{k-1}{2k} \sum_{i \in V} \mu_i \mu_i \qquad \forall \mu \in \mathbb{R}^n \tag{22}$$

where typically the Euclidean norm of $\mu$ is one. Constraint (22) has an infinite number of rows then a separation routine based on eigenvalues is also introduced in [47]. Computational results show that SDP-based inequalities are very relevant for the problem with $k \geq 7$.

In [9], the authors present a large class of hypermetric inequalities for the max-$k$-cut. This class of constraints generalizes some of the previous inequalities such as the Triangle (2) and Clique (3) inequalities. For brevity and because of the complexity of separation of these inequalities is unknown the details are skipped.

In [36, 35], the author applied a branch-and-cut algorithm based on the LP formulation of the $k$-way equipartition problem in the realignment of a Football league where $k \in \{4, 8\}$ and $|V| = 32$. In [35], for graphs of sizes between $100$ to $500$ vertices, the problem is solved with a gap of less than $2.5\%$ for $k = 4$.

The branch-and-cut algorithm based on the node-and-edge linear formulation of max-$k$-cut is applied in [14, 27]. For both references, the LP relaxation is tightened by general clique and triangle inequalities. In [14] the authors apply the branch-and-bound to solve a two-level graph partitioning problem in mobile wireless communications and they solve problems with $n = 100$

and $k \in \{2, 3, 4\}$ for sparse graphs. In [27], the authors study the connected max-$k$-cut problem for applications in gas an power networks.

A modified version of combinatorial constraints ((2), (3), (19)-(21)) is also applied to reinforce the SDP relaxation of the max-$k$-cut problem.

In [48], the authors compare Triangle (2) and Clique (3) inequalities in the SDP max-$k$-cut. Their experimental tests show that triangle inequalities are more relevant than the clique. In [46], authors study some of the combinatorial inequalities. Their results show that triangle, wheel and bicycle wheel are the most important classes of inequalities.

In [20], the authors design a branch-and-cut algorithm based on the SDP formulation (called SBC) of the minimum $k$-partition problem. The SBC method includes triangle and clique inequalities and it uses a dynamic version of the bundle method [26] to solve the SDP formulation. The SBC algorithm is able to compute optimal solutions for dense graphs with $60$ vertices and for sparse graphs with $100$ vertices.

The state-of-the-art `BundleBC` algorithm [3] is an improvement of the SBC, where the ideas of the `Biq Mac` solver [44] are extended to the max-$k$-cut problem. In [3], the separation of clique inequalities and some rules for choosing the variables are investigated. Computational results show that the use of clique inequalities reduces the number of subproblems analyzed in the branch-and-bound tree and that the `BundleBC` solver is faster than the SBC. The performance of `BundleBC` is compared with our proposed method in Section 7.

#### 2.2.2 Lower bound

In the branch-and-bound algorithm, feasible solutions (incumbent solutions) are used to prune subproblems and the least feasible solution is also returned as a solution if the method stops before optimality.

Heuristic and metaheuristic are usually used to find initial feasible solutions and to improve the existing incumbent. For the max-cut problem ($k = 2$), many heuristics have been proposed (see, for example, [52, 29]). For max-$k$-cut, there are very few options.

In [22], the authors present the $0.878$-approximation algorithm for the max-cut problem. The 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. In [17, 28, 39], the authors propose extensions of the $0.878$-approximation to the max-$k$-cut problem.

In the SBC algorithm [20], the authors propose an 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 [22] (for $k = 2$) and by [17] (for $k \geq 3$). In summary, the ICH sums the value of the relaxed solution of edges between three vertices, then, if the sum is greater than a constant term, the three vertices are set in the same partition.

In [53], a multistart 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 [33], the authors present a multiple operator heuristic (MOH) for the max-$k$-cut problem. MOH is an iterative method that applies five operators organized into three search phases. The first phase is called the descent-based improvement that finds a local optimum solution with two intensification-oriented operators. In second phase, the solution is diversified in the improvement stage where the tabu search method [21] is applied with two operators. Finally, in the last step, 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) [37] heuristic is an iterative method that explores a distant neighborhood of the current incumbent solution and jumps from a current solution to a new one if there is an improvement. In addition, a local search is applied to find a new local optimum.

The greedy randomized adaptive search procedure (GRASP) [15] is also an iterative method that is divided into three stages: 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 found solution.

The VNS and GRAPS are applied with success for the max-cut problem in [16].

#### 2.2.3 Branching Rules

Many researchers have investigated several variable selection strategies in the branch-and-bound framework, see for example [1, 32]. A strategy that has attracted considerable interest is the *strong branching* (see [19, 31]). This method 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 the strong and the pseudo-cost branching strategies. Their computational results show that this rule is better than the strong and pseudo-cost branching.

In [3], the authors investigate four branching rules for the `BundleBC` solver. In the first rule, the variable selected is the most decided, i.e, the edge that is the closest to $0$ or $1$. The second rule is more elaborated, it selects the edge of vertices $i'$ and $j'$ such that:

$$i' \in \underset{i\in V}{\operatorname{argmin}} \sum_{r\neq i, r=1}^{n} (0.5 - |\hat{x}_{ir} - 0.5|)^2, \tag{23}$$

$$j' \in \underset{j\in V, j\neq i'}{\operatorname{argmin}} \sum_{r\neq j, r=1}^{n} (0.5 - |\hat{x}_{jr} - 0.5|)^2 \tag{24}$$

where $\hat{x}$ is the optimal (upper bound) solution of the current node. In the third rule of [3], the chosen variable is the edge that is least decided. The fourth rule is similar to the second but for the vertex $j'$, the $\operatorname{argmin}$ is replaced by $\operatorname{argmax}$. Computational results show that the second rule provides the best results and the first rule gives the worse.

#### 2.2.4 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 [32, 38]

# 3 Proposed branch-and-bound framework

This section sketches the branch-and-bound algorithm used to solve the max-$k$-cut problem to optimality. Although the framework is well known and its description can be found in any work of enumerative methods for $\mathcal{NP}$-complete problems, e.g., [44], it is important to describe and investigate some of its components to develop an efficient solver.

## 3.1 Outline

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

1. *Upper bound.* The upper bound ($z_{ub}$) is obtained by a solution of the relaxed max-$k$-cut problem. It is important to apply a method that gives a good trade-off between the time of execution and the quality of bounds. Section 5 investigates the SDP and LP relaxations. The inclusion of a cutting plane algorithm on the branch-and-bound tree is also evaluated.

2. *Feasible solutions as lower bounds.* Fast feasible solutions ($z_{lb}$) that have a value close to optimality can speed up the branch-and-bound method by pruning desirable branches. Section 5.3 analyzes the following four heuristic methods:

   - Iterative clustering heuristic (ICH) [20],
   - Multiple operator heuristic (MOH) [33],
   - Variable neighborhood search (VNS) [23, 37], and
   - Greedy randomized adaptive search procedure (GRASP) [15].

3. *Splitting problem.* In [18], the authors pointed out that an important decision that impacts the effectiveness of the branch-and-bound method is the strategy used to partition the feasible region. For the max-$k$-cut problem, the branching can be applied to a vertex or an edge of the graph. Hence, the following two types of splitting are studied in Section 6.1:

   - *Dichotomic*, where the branching variables correspond to an edge $(i, j) \in E$ of the graph where each node is split into two children nodes: in the first, the edge $(i, j)$ is *cut*, and in the second, it is *not-cut* i.e, vertices $i$ and $j$ are imposed to be in the same partition,
   - *polychotomic* or $k$-chotomic branching. In this strategy, a vertex $i \in V$ is set to a different partition in each child node. Max-$k$-cut problems can have at most $k$ partitions, hence, each node can generate $k$ children.

4. *Branching rule.* Commonly a variable that has a fractional value is selected to be branched. A relaxed solution may have several non-integral variables. Thus, it is appropriate to use some techniques for choosing the right variables to be branched on Section 6.2 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 6.3:

- Best-first search strategy (BeFS),
- Depth-first search (DFS), and
- Breadth-first search (BrFS).

## 3.2 The branch-and-bound scheme

The proposed generic branch-and-bound can be summarized in the scheme described in Algorithm 1 below. The initial relaxation ($\mathbf{P}_0$) is a relaxed formulation of the max-$k$-cut problem. Each node in the list `L` is known as an active node of the branch-and-bound. The `Lower-bound(`$\mathbf{P}$`)` function returns a feasible heuristic solution ($x$) and its lower bound value ($z_{lb}$). The lower bound can be computed for each branch or occasionally. Likewise, the function `Upper-bound(`$\mathbf{P}$`)` returns the value of the upper bound ($z_{ub}$) of the relaxed formulation $\mathbf{P}$ and its relaxed solution ($\hat{x}$). Moreover, a CPA can be applied to strengthen the upper bound of the relaxation.

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

**Algorithm 1:** Generic branch-and-bound algorithm for max-$k$-cut.

# 4 Computational settings

The results of experimental tests are shown and discussed simultaneously with the description of each component of the branch-and-bound algorithm. Therefore, this section aims to describe the computational environment, the instances used and the comparison of methodologies.

**Computational environment** Tests are performed on a Linux PC with two Intel(R) Xeon(R) $3.07$ GHz processors. The `mosek` $8.1$ solver [5] is used for solving the SDP and LP relaxations.

## 4.1 Comparison of results

To analyze the procedures of the branch-and-bound algorithm, a modified *performance profiles* [11] which shows on the $y$-axis the percentage of problems that are solved for a given time ($x$-axis). Hence, they show the temporal progression of methods. Moreover, a *performance table* is used to measure the quality of different methods to generate feasible solutions.

## 4.2 Set of instances

The procedures of the branch-and-bound method are tested in some of the most difficult instances from the literature, especially from the `BundleBC` [3] and `Biq Mac` [51] libraries. Some instances were also generated using the `rudy` graph generator [45]. The instances are divided into:

**SET A** This set considers two subclasses of instances with a total of $65$ instances. The first class consists of 40 instances from [3]. Some of these instances are complete graphs and others are stems from an application in physics to determine the energy-minimum states of so-called Potts glasses instances. The second class was generated by `rudy` such that the instances have a varying number of vertices ($|V|$) ranging from 20 to 50 and such that edges are chosen randomly in such a way that graphs have edge densities of $25\%$ and $50\%$.

**SET B** Some of the largest instances from [51] are considered in this set. It consists of the following instances:

- bqp: ten weighted graphs with dimension $100$, density $0.1$, and edge weights are chosen in $\{-100, 0, 100\}$.
- g05: ten unweighted graphs with edge probability $0.5$ and dimension $100$.
- pm1d: ten weighted graphs with edge weights in $\{-1, 0, 1\}$, density $0.99$, and dimension $100$.
- ising2: six one-dimensional Ising chain instances with dimension $|V| \in \{200, 250, 300\}$.
- ising3: three one-dimensional Ising chain instances with dimension $|V| \in \{200, 250, 300\}$.
- be: ten Billionnet and Elloumi instances with density $d = 0.8$, with edge weights in $\{-50, 50\}$ and dimension equal to $200$.

# 5 Bounding procedures in the branch-and-bound framework

This section describes some procedures to calculate upper bounds and four heuristic methods are investigated to obtain lower bounds for the max-$k$-cut problem.

## 5.1 Computing upper bounds

This section presents the computational results of the branch-and-bound algorithm for edge-only, node-and-edge and node-only relaxations.

### 5.1.1 Upper bound methods

For the LP relaxations, the relevance of the SDP-based Constraints is also investigate since it has been shown in [47] that these constraints are strong but expensive. Therefore, this section considers the next five methods:

- *SDP*: this method applies the SDP relaxation reinforced with combinatorial inequalities ((2), (3), (19)-(21)).
- *Edge*: this method uses the edge-only relaxation (see Section 2.2.1) reinforced with combinatorial inequalities.
- *Edge-EIG*: this method is *Edge* with the addition of SDP-based Inequalities (22) proposed in [46].
- *NoEd*: this method uses the node-and-edge relaxation of the max-$k$-cut presented in Section 2.2.1. This formulation is also reinforced with combinatorial inequalities.
- *NoEd-EIG*: this method is *NoEd* with the addition of SDP-based Inequalities (22).

### 5.1.2 Computational results of formulations

Figure 1 shows the results of the five methods for instances of SET A and for $k \in \{3, 5, 7, 10\}$.

Results show that the SDP is the most efficient method when $k = 3$ and that the *Edge-EIG* is the best one for larger values of partitions. The strength of SDP for small $k$ can be explained by the fact that only a few bound constraints (17) are violated.

For $k \geq 5$, the *Edge-EIG* is better than other methods. For $k \in \{7, 10\}$, after $200$ seconds the *Edge-EIG* solves more than $70\%$ of the instances while other methods take at least $1000$ seconds to solve the same amount of instances. Moreover, results indicates that SDP-based inequalities (22) are very relevant for the LP formulations since *Edge-EIG* and *NoEd-EIG* obtained better solutions than *Edge* and *NoEd*, respectively.

In Figure 1, the performance of *Edge* and *NoEd* are very similar, but *Edge-EIG* is much better than *NoEd-EIG*, this is because of different SDP-based Inequalities (22) are activated in the CPA for each formulation.

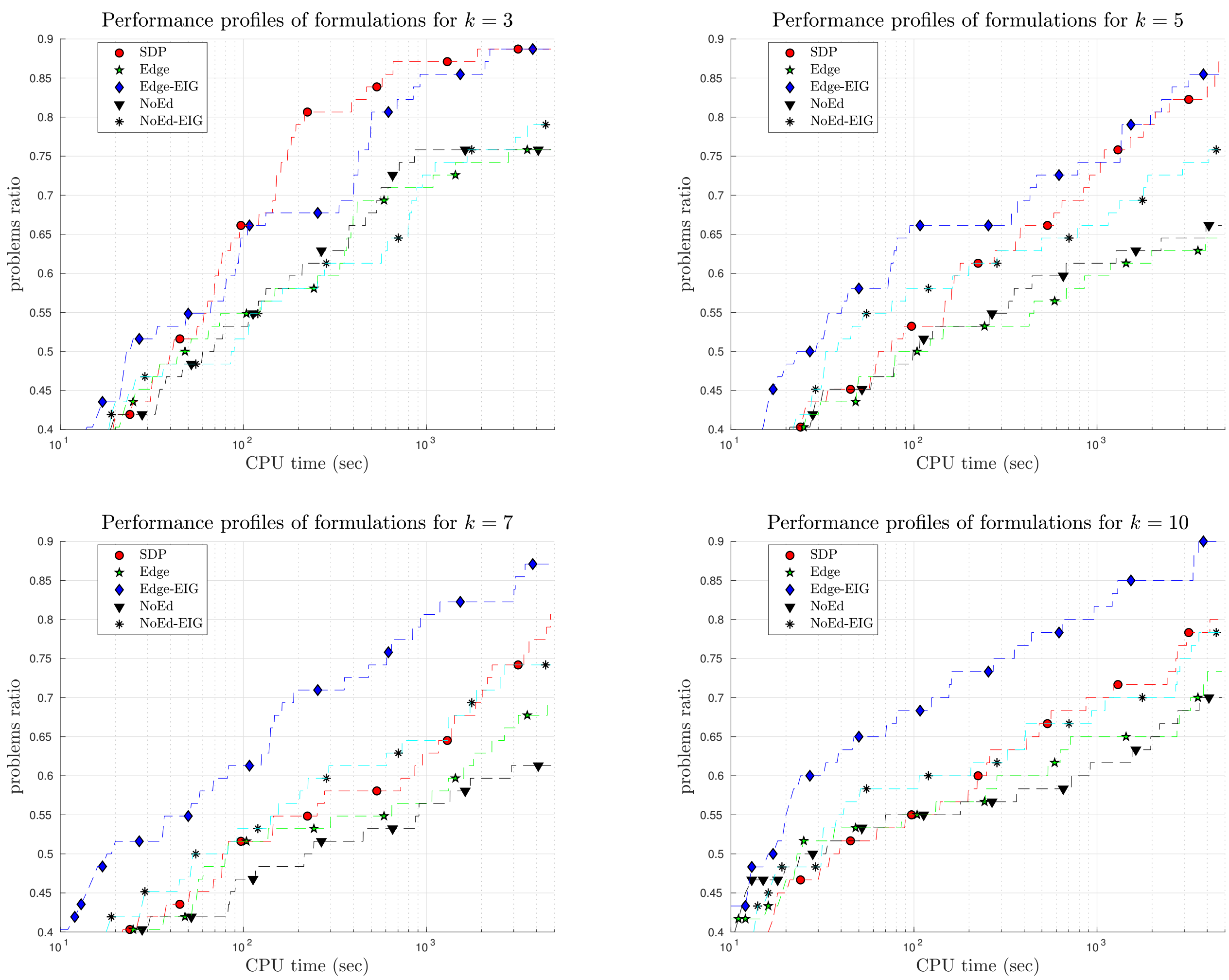


Figure 1: Performance profiles of different formulations of the max-$k$-cut problem, for the instances of SET A and for $k \in \{3, 5, 7, 10\}$.

## 5.2 Cutting plane algorithm (CPA)

The CPA is an iterative method that solves a relaxed formulation of the problem, and then searches and adds some of the violated inequalities (or cut planes) in the relaxation. This section aims to investigate the branch-and-bound method with CPA.

This work applies the CPA proposed in [47] that implements the *early-termination* technique in the interior point method for the LP and SDP formulations of the max-$k$-cut problem using separation routines for combinatorial and SDP-based constraints (22).

In [47], the CPA is stopped after one hour and some relaxations require more than $100$ seconds to be solved. To apply the same CPA in our branch-and-bound algorithm, two other 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 ($\mathrm{it\_time} = 10\mathrm{s}$). Therefore, if a CPA iteration lasts more than $\mathrm{it\_time}$ the method is stopped and the last bound obtained. The second stopping criterion is based on the fact that the largest improvements of CPA occur in the first iterations of the CPA, for example, see computational study presented in [47]. Therefore, the proposed CPA is stopped when $\mathrm{it\_impr}$ is inferior to a small $\varepsilon = 10^{-4}$. Hence, it avoids expensive and non-important rounds of the algorithm. Let $z_i$ be the value of the upper bound solution of the current CPA iteration $i$. The improvement, $\mathrm{it\_impr}$, is calculated as:

$$\mathrm{it\_impr} = \frac{z_{i-1} - z_i}{z_i} \tag{25}$$

Since the CPA is stopped prematurely, the bounds obtained at the end of our method are not as strong as the observed in [47], but branch-and-bound can explore more nodes in the search tree.

### 5.2.1 Branch-and-cut methods

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 so the inclusion of all constraints is impractical. However, one can also include a CPA on other nodes of the branch-and-bound tree to further reinforce their solutions. Therefore, the following strategies are analyzed:

- `cut-BB`: is the branch-and-bound that applies the CPA only on the root node, also known as the cut-and-branch method.
- `BC-i`: a different method is considered for each $i \in \{2, 7, 21\}$. Those approaches are cut-and-branch methods that also execute the CPA when the level ($l$) of a node in the branch-and-bound tree is a multiple of $i$. The level of a node is how far it is from the root. Hence, the root node is at level zero ($l_{root} = 0$) and all the direct children nodes of the root node are at level 1 and their children are at level $l_i + 1 = 2$. For example, in the `BC-2` strategy, the CPA is executed at each pair-level node.
- `BC-all`: is the branch-and-cut method that executes the CPA in all the nodes of the branch-and-bound tree.

### 5.2.2 Computational results: branch-and-cut

Figure 2 shows the results of the five methods described in this section for instances of SET A and for $k \in \{3, 5, 7, 10\}$.

Results in Figure 2 show that the branch-and-bound is more efficient when the CPA is applied in all the nodes of the enumeration tree, i.e., the `BC-all` is the most efficient method especially for $k \in \{3, 5, 7\}$ followed by `BC-1`. For instance, for $k \in \{3, 5\}$, the `BC-all` and `BC-1` could solve 60% of all problems in SET A in less than 200 seconds while `BC-21` and `cut-BB` need at least 1000 seconds to solve the same amount of problems.

## 5.3 Computing lower bounds

This section presents the implementation details of the four heuristics discussed in 2.2.2: ICH, MOH, VNS, and GRASP. To obtain a good trade-off between the quality and the CPU time, some parameters are different from the literature.

Two solutions $a$ and $b$ are neighbors if the distance between these two solutions is one ($d(a, b) = 1$), i.e., if only one vertex of $a$ is assigned to a different partition than $b$. A local search with simple-transfer moves, at each iteration, to a neighbor with a better solution. If any neighbor has a better

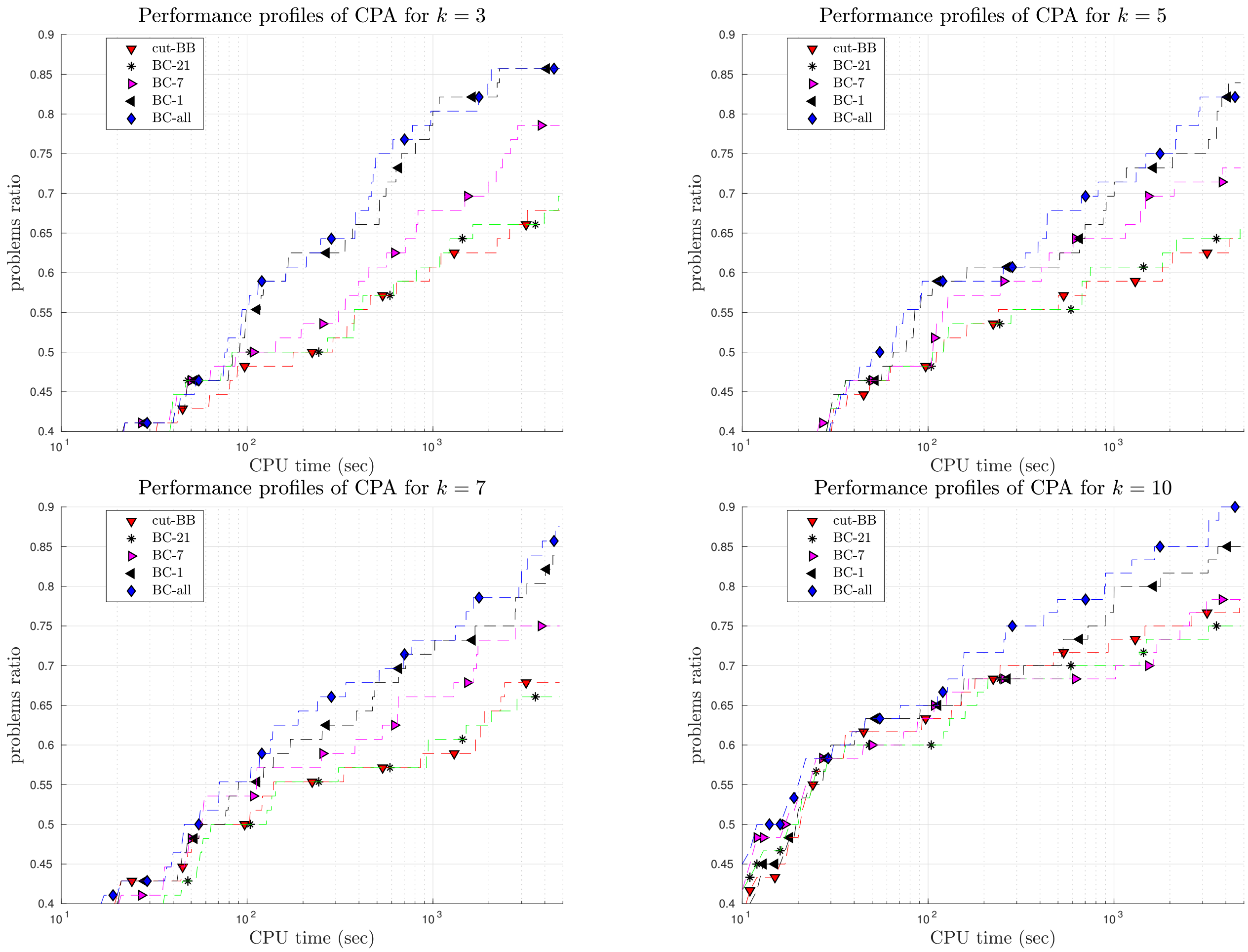


Figure 2: Performance profiles of CPA into the branch-and-bound algorithm, for the instances of SET A and for $k \in \{3, 5, 7, 10\}$.

solution, then it is a local maximum. In double-transfer, the best neighbor with a distance equal to 2 is considered. The following describes the parameters and details of the implementation of each heuristic.

1. *ICH*. For the ICH, the constant term that decides if three nodes should be in same partition is fixed to $1.7$ (as proposed in [20])), and when the re-optimization is called for solving a derivate subgraph, the total time of CPA is limited to $1.0$ seconds.

2. *MOH*. In the improvement phase (second stage), only the local search with simple-transfer (called $O_3$ in [33]) is considered the fourth operator ($O_4$) is quite expensive. In addition, it is imposed:

    - tabu tenure is equal to $10$,
    - the perturbation strength that measures the perturbation allowed in the third phase of the method is fixed in $50\%$,
    - maximum number of iterations is $500$, and

- maximum time is $2.0$ seconds.

3. *VNS*. As defined before, the VNS explores a distant neighborhood, from a local maximum $a^*$. In this work, the VNS heuristic considers a solution $b$ as a neighbor solution of $a^*$ if their distance is inferior to $d_{max}$ ($1 \leq d(a^*, b) \leq d_{max}$). 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,
   - the maximum number of iterations is $1000$, and
   - the maximum time is $2.0$ seconds.

4. *GRASP* applies a local search that uses single and double-transfer moves to find the best neighbor solutions of a random feasible solution build in the construction phase. Moreover, the maximum number of iterations is set to $5000$ and the maximum time of execution is $2.0$ seconds.

Moreover, in order to reduce the total running time of the branch-and-bound framework, the computation of a feasible solution is executed at each $15^{th}$ (experimentally, a good choice) iteration of the branch-and-bound method.

### 5.3.1 Computational results: lower bounds

Table 1 shows the results of the four heuristic methods for $k \in \{3, 5, 7, 10\}$. In Table 1, the name and the number of partitions $k$ of each set of instances are shown in the first columns. Then, for each method, the average gap ($gap_{avg}$) and the time of execution in seconds (time(s)) are presented.

The *gap* is calculated by the best known feasible solution ($lb_{best}$) of the instances tested. Therefore, the percentage gap of an instance $i$ of method $m$ is obtained by:

$$gap(\%)_{i,m} = \frac{lb_{best} - lb_{i,m}}{lb_{best}} * 100 \tag{26}$$

where $lb_{i,m}$ is the value of the solution found for the problem $i$ and by method $m$. Table 1 shows, at each line, the average result of ten rounds of executions, and it highlights the results with best performances.

Table 1 shows that on average the gap of all four methods is lower than 2.3% and that the VSN heuristic gives, for almost all the tests, the best bound with competitive CPU time. The proposed VNS heuristic has on average a gap of $0.5\%$ for the tested instances.

# 6 Branching strategies

This section presents the following three procedures of the branch-and-bound: split, branching and selection. These three algorithmic components play a very important role in the performance of the branch-and-bound algorithm.

Table 1: Results of four heuristic methods for finding feasible solutions.

| Instance | | ICH | | MOH | | VNS | | GRASP | |
|---|---|---|---|---|---|---|---|---|---|
| name | $k$ | $gap(\%)$ | time(s) | $gap(\%)$ | time(s) | $gap(\%)$ | time(s) | $gap(\%)$ | time(s) |
| SET A | 3 | 0.67 | 8.1 | **0.24** | 2.0 | 0.33 | 1.2 | 0.54 | 1.7 |
| SET B | 3 | 1.27 | 25.1 | 1.08 | 2.0 | **0.80** | 2.1 | 1.36 | 2.0 |
| SET A | 5 | 0.96 | 13.2 | 0.53 | 1.9 | **0.21** | 1.3 | 0.77 | 1.7 |
| SET B | 5 | 0.99 | 31.8 | 1.32 | 2.0 | **0.89** | 2.1 | 1.64 | 2.0 |
| SET A | 7 | 0.55 | 7.4 | 0.85 | 1.9 | **0.33** | 1.4 | 0.91 | 1.7 |
| SET B | 7 | 1.04 | 30.5 | 1.77 | 2.0 | **0.83** | 2.1 | 1.80 | 2.0 |
| SET A | 10 | 0.59 | 7.30 | 0.94 | 1.9 | **0.22** | 1.3 | 1.30 | 1.6 |
| SET B | 10 | 1.12 | 30.5 | 1.84 | 2.0 | **0.42** | 2.1 | 2.30 | 2.0 |

## 6.1 Splitting problem

The branch-and-bound algorithm converges if the solution space of each subproblem is contained in the solution space of the original problem and if the original problem has finite solutions. Typically, the subproblems generated at each branch (or node) 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. [44]), 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 $\hat{x}_{ij} = \lfloor \hat{x}_{ij} \rfloor = 0.0$ to one subproblem and $\hat{x}_{ij} = \lceil \hat{x}_{ij} \rceil = 1.0$ to the other.

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

**$k$-chotomic split.** To avoid infeasible subproblems, this work proposes a vertex branching that is similar to the split applied in the max-cut problem. In this strategy, 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. However, a combination of this strategy with depth-first search (see Section 6.3) can mitigate this drawback.

### 6.1.1 Implementation details: splitting

This section discusses some implementation details for applying the $k$-chotomic and the dichotomic split on the 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 Triangle Inequalities (2) and Clique inequalities (3). For Triangle (2), 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:

$$x_{ih} = 1 \quad \& \quad x_{hj} = 1 \quad \Rightarrow \quad x_{ij} = 1, \tag{27}$$

$$x_{ih} = 1 \quad \& \quad x_{hj} = 0 \quad \Rightarrow \quad x_{ij} = 0, \tag{28}$$

$$x_{ih} = 0 \quad \& \quad x_{hj} = 1 \quad \Rightarrow \quad x_{ij} = 0. \tag{29}$$

To avoid any infeasible solutions in the dichotomic split, it is necessary to certify that all clique Inequalities (3) are also satisfied at each subproblem. However, as it was pointed out in [46], it is computationally very expensive to enumerate all the cliques even for small instances.

For the $k$-chotomic strategy in the max-$k$-cut problem, any vertex of the graph 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 $v \in V$ with the largest sum of incident weight edges in the input graph $(v \in \text{argmax}_{v\in V} \sum_{j=1}^{n} w_{vj})$ is set at $P_1$.

### 6.1.2 Computational results: splitting

Figure 3 shows the results of dichotomic and $k$-chotomic splits described in this section for instances of SET A and for $k \in \{3, 5, 7, 10\}$.

For $k = 3$, the results show that the $k$-chotomic split is more efficient than dichotomic, for example, the $k$-chotomic strategy can solve $85\%$ of problems in $1000$ seconds, but the dichotomic can solve less than $75\%$ for the same time. For $k \in \{5, 7\}$, the dichotomic have slightly better performance for instances that can be solved before 300 seconds but for the most expensive instances the $k$-chotomic can solve $5\%$ more instances than dichotomic. For $k = 10$, both methods have similar performances.

## 6.2 Branching Rules

The variable and node selections are both critical decisions that impact the performance of the branch-and-bound methods. Branching on a variable that does not give any serious improvement on any of the children nodes can lead to an extremely large and expensive search tree.

### 6.2.1 Implementation details: branching

In the $k$-chotomic split strategy, a vertex $v \in V$ is already fixed in a node $\mathbf{Q}_i$ in the branch-and-bound tree 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. The following rules are investigated in this work:

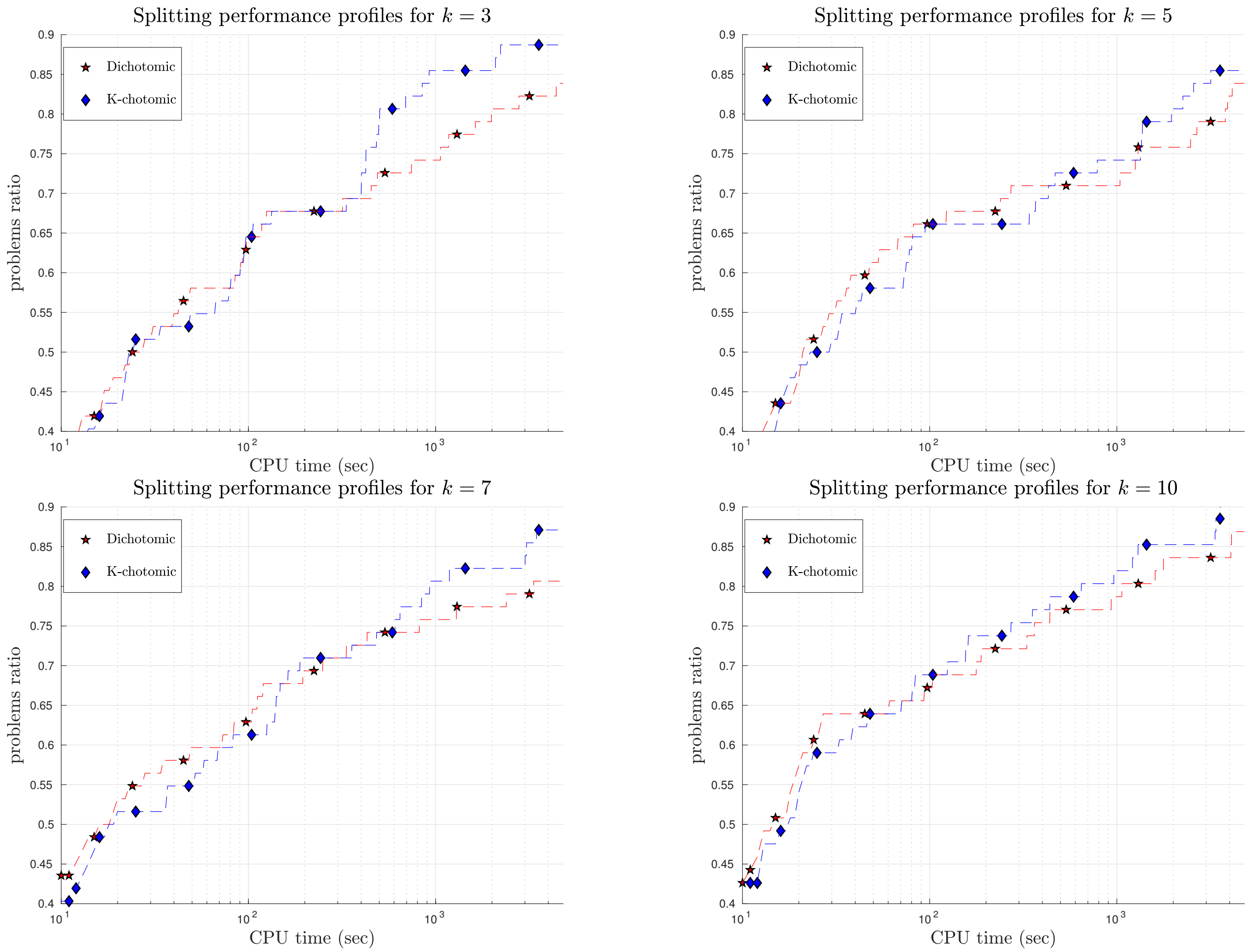


Figure 3: Performance profiles of dichotomic and $k$-chotomic strategies, for the instances of SET A and for $k \in \{3, 5, 7, 10\}$.

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

- `R2`. The second rule is similar to the third rule proposed in [3]. Then, it consists of choosing the edge that is the *least decided*. Similarly to R1, while applying this rule in the $k$-chotomic split, one of the vertices has to be already assigned to a partition.

- `R3`. This rule is similar to the best rule of [3]. For the $k$-chotomic split, the vertex $j' \in V$ has to be already fixed and the selected variable $i' \in V$ comes from Argument (23).

- `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 fifth 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.

### 6.2.2 Computational results: branching rules

Figure 4 shows the results of five rules to select variable to be branched for instances of group SET A and for $k \in \{3, 5, 7, 10\}$.

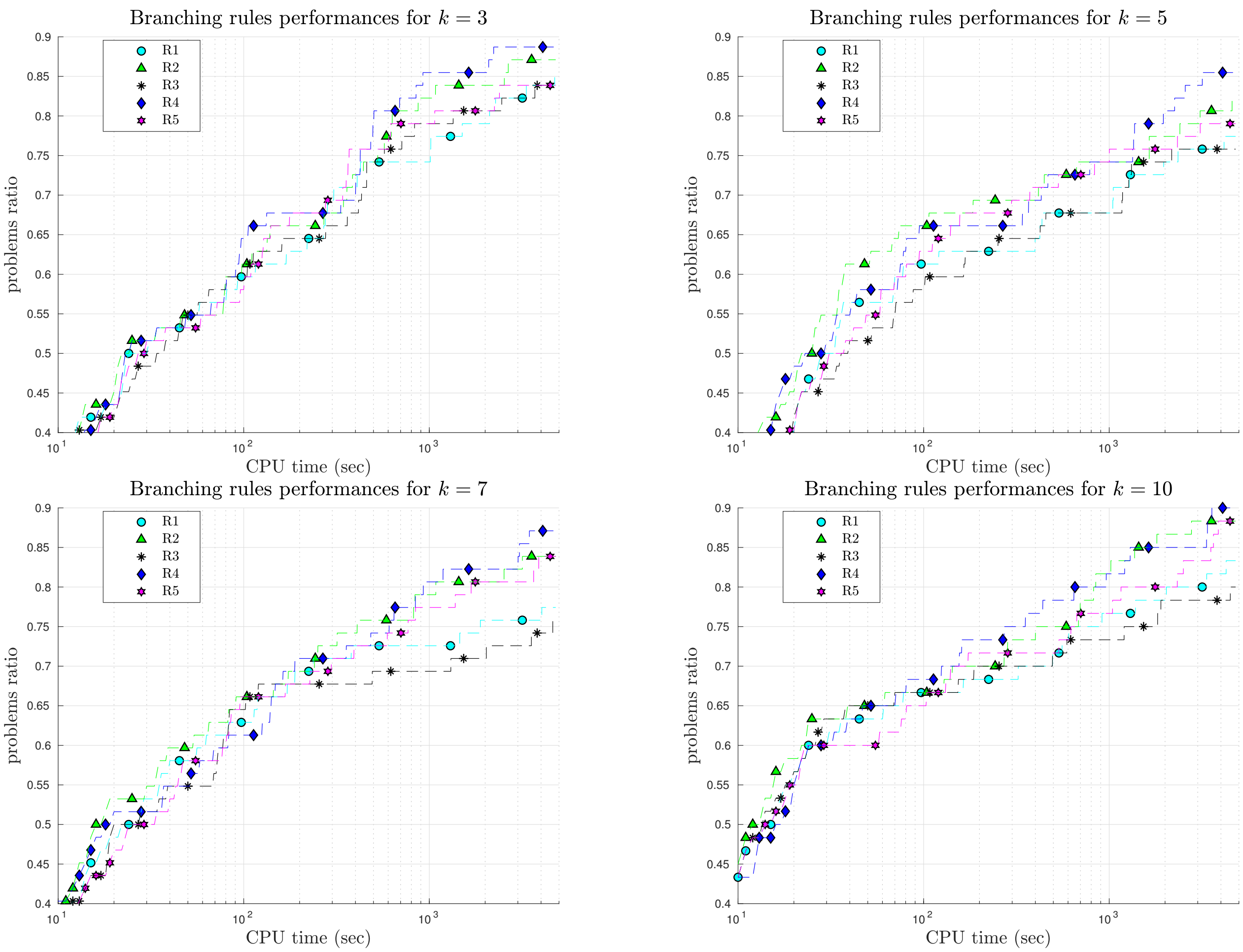


Figure 4: Performance profiles of five branching rules, for the instances of SET A and for $k \in \{3, 5, 7, 10\}$.

For $k = 3$, the rule `R4` has the best performance, for example, `R4` solves $87\%$ of the instances while others can solve at most $83\%$ in 1 hour. For $k \in 5, 7, 10$, the `R2` and `R4` have the best results, especially for the most difficult instances.

## 6.3 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 following three strategies to select active nodes on the branch-and-bound algorithm are considered:

- *Best-First Search (*`BeFS`*).* This strategy selects the best active node in the branch-and-bound tree. 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.

- *Breadth-First Search (*`BrFS`*).* The `BrFS` 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 good feasible solutions.

- *Depth-First Search (*`DFS`*).* The DFS strategy is the opposite of `BrFS`, so it uses the data structure of last-in, first-out to select a node. Therefore, the method goes deep into 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.

### 6.3.1 Computational results: node selection

Figure 5 shows the results of three strategies to select an explored node on the branch-and-bound tree for instances of group SET A and for $k \in \{3, 5, 7, 10\}$.

In Figure 5, the results of the three selection strategies are very similar but the `DFS` strategy obtains slightly better performance, especially for $k = 7$.

# 7 Comparison with state-of-the-art

This section shows the results of the branch-and-bound for the *Edge-EIG* and *SDP* formulations and it compares their performance with the `BundleBC` solver. The components of our branch-and-bound are fixed to the following parameters:

- *Type of branch-and-bound*: the branch-and-cut algorithm that applies the CPA in all the nodes (`BC-all`) is used to find the optimal solutions,
- *feasible solution*: the *VNS* heuristic is applied to find good feasible solutions,
- *branching rule*: the rule `R4` is the method to select variables,
- *node selection*: the `BeFS` is the strategy used to select unexplored nodes on the branch-and-bound tree,
- *splitting problem*: the $k$-chotomic strategy to split the problem is set.

For the `BundleBC`, the triangle (2) and clique (3) inequalities are activated. All the methods are set to stop after $1.5$ hours, they use only one CPU thread and the tests are executed on the same machines. Figure 6 shows the performance profiles of *SDP*, *Edge-EIG* and `BundleBC` for $k \in \{3, 5, 7, 10\}$.

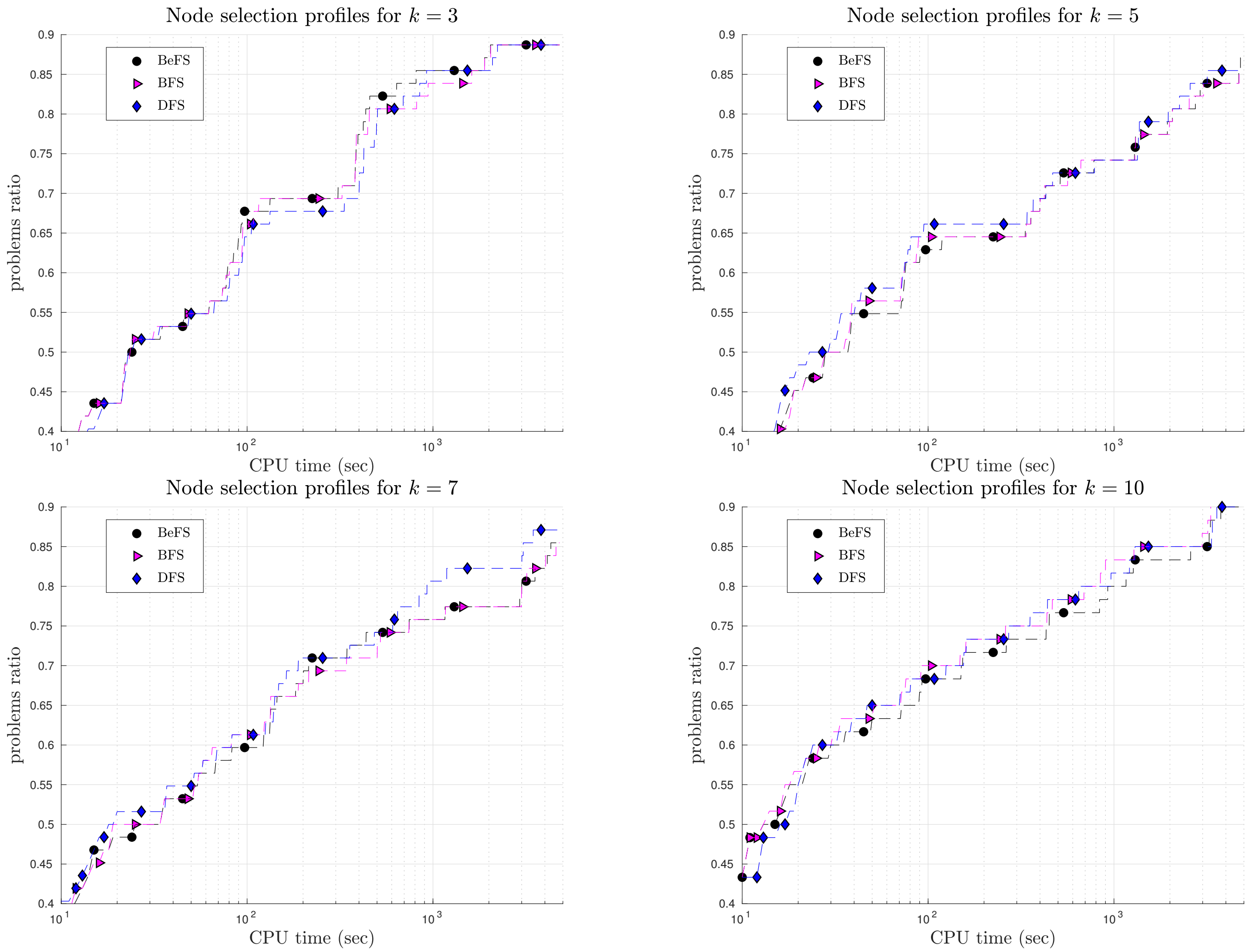


Figure 5: Performance profiles of three strategies of node selection: `BeFS`, `BrFS`, and `DFS`; for $k \in \{3, 5, 7, 10\}$ and for the instances of SET A.

The results of Figure 6 show that `BundleBC` is the most efficient method for $k = 3$. However, for $k \geq 5$ the *Edge-EIG* method, which is the LP edge-only formulation reinforced with combinatorial ((2), (3), (19)-(21)) and SDP-based inequalities (22), obtains the best results, principally for $k \geq 7$. For example, for $k = 7$ the *Edge-EIG* can solve $60\%$ of the problems in less than $100$ seconds while *SDP* and `BundleBC` take almost $1000$ seconds to solve the same ratio of problems.

# 8 Discussion

To design an efficient method, this work investigates some of the most important features of the branch-and-bound method to solve the max-$k$-cut problem to optimality.

In the upper bound procedure, the SDP and LP relaxations are investigated together with the inclusion of the CPA in the branch-and-bound tree. For the lower bound, four heuristic methods: ICH, MOH, VNS, and GRASP were proposed. 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

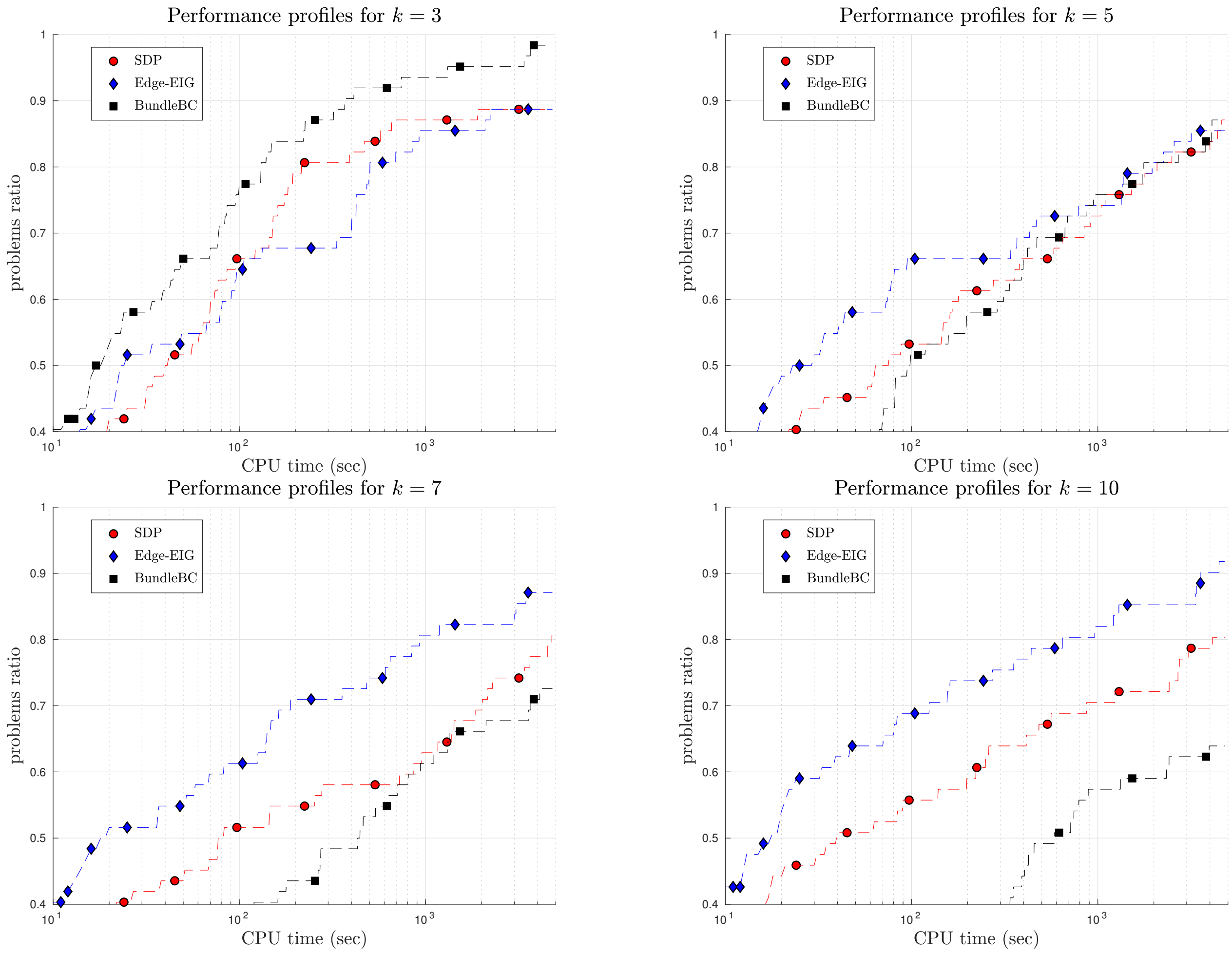


Figure 6: Performance profiles of *SDP*, *Edge-EIG*, and `BundleBC` for $k \in \{3, 5, 7, 10\}$ and for the instances of SET A.

be branched is studied. Moreover, three strategies for the node selection: `BeFS`, `DFS`, and `BrFS` were studied.

From our computational tests, we conclude that the SDP-based inequalities proposed in [47] are relevant in the branch-and-bound scheme. The results presented in this work can be summarized as

1. *Formulations.* Results of Section 5.1 show that the *Edge-EIG* and *SDP* formulations are the most efficient methods. While semidefinite based methods (*SDP* and `BundleBC`) obtains the best results for $k = 3$ the *Edge-EIG* outperforms for $k \geq 5$.

2. *The branch-and-bound with CPA.* Section 5.2 studies five different strategies of applying the CPA in the branch-and-bound method. Results show that the strategy that includes the CPA in all the nodes of the branch-and-bound tree is the most efficient.

3. *Feasible solutions.* Computational tests in Section 5.3 demonstrates that the VNS heuristic obtains the best performances.

4. *Splitting problem.* Section 6.1 studies the performance of branch-and-bound with dichotomic

and $k$-chotomic strategies. The $k$-chotomic has the best results, especially for the most difficult instances.

5. *Branching rules.* The fourth rule (`R4`) that selects variables with the largest weights provides the best results.

6. *Node selection.* The performance of the three strategies studied is quite similar. However, `DFS` has been the best strategy for the most difficult instances.

7. *Comparison with* `BundleBC`. Our tests compared the performance of *Edge-EIG* and SDP with `BundleBC` solver. The `BundleBC` method is most efficient for small values of partitions and *Edge-EIG* outperforms all the methods when $k \geq 5$.

Future research involves the design of learning methods to select the right combination of components of branch-and-bound for any instance and to guide the selection of violated inequalities in the CPA, especially for SDP-based constraints.

# References


[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] 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.

[4] M. F. Anjos and J. Neto. A class of spectral bounds for max k-cut. *Discrete Applied Mathematics*, 2019.

[5] Mosek ApS. mosek. http://www.mosek.com, 2019.

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

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

[10] E. de Klerk, D.V. Pasechnik, and J.P. Warners. On approximate graph colouring and max-$k$-cut algorithms based on the $\theta$-function. *Journal of Combinatorial Optimization*, 8(3):267–294, 2004.

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

[12] 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.

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

[14] 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.

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

[16] 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.

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

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

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

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

[27] C. Hojny, I Joormann, H L"uthen, and M. Schmidt. Mixed-integer programming techniques for the connected max-k-cut problem. Technical report, Optimization Online, 2019.

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

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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

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

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

[38] 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.

[39] Alantha Newman. Complex semidefinite programming and max-k-cut. *CoRR*, abs/1812.10770, 2018.

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

[41] 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.

[42] 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.

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

[44] 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.

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

[46] 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.

[47] V. J. Rodrigues de Sousa, M. F. Anjos, and S. Le Digabel. Improving the linear relaxation of maximum k-cut with semidefinite-based constraints. *EURO Journal on Computational Optimization*, 7(2):123–151, 2019.

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

[49] 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.

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

[51] A. Wiegele. Biq mac library - binary quadratic and max cut library. http://biqmac.uni-klu.ac.at/biqmaclib.html, 2019.

[52] 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.

[53] 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.