The Vertex k-cut Problem

Denis Cornaz[1], Fabio Furini[1], Mathieu Lacroix[2], Enrico Malaguti[3], A. Ridha Mahjoub[1], Sébastien Martin[4]

[1] *PSL, Université Paris Dauphine, CNRS, LAMSADE UMR 7243 75775 Paris Cedex 16, France.*
denis.cornaz@dauphine.fr
fabio.furini@dauphine.fr
mahjoub@lamsade.dauphine.fr

[2] *Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, 99, Avenue J.-B. Clément 93430 Villetaneuse, France.* mathieu.lacroix@lipn.univ-paris13.fr

[3] *DEI, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.*
enrico.malaguti@unibo.it

[4] *LCOMS, Université de Lorraine, Ile du Saulcy, 57045 Metz, France*
sebastien.martin@univ-lorraine.fr

Last update: September 21, 2017

Abstract

Given an undirected graph $G = (V, E)$, a vertex k-cut of G is a vertex subset of V the removing of which disconnects the graph in at least k connected components. Given a graph G and an integer $k \geq 2$, the vertex k-cut problem consists in finding a vertex k-cut of G of minimum cardinality. We first prove that the problem is NP-hard for any fixed $k \geq 3$. We then present a compact formulation, and an extended formulation from which we derive a column generation and a branching scheme. Extensive computational results prove the effectiveness of the proposed methods.

keywords: Vertex Cut, Mixed-Integer Programming Models, Branch and Price, Exact Algorithms.

1. Introduction

A *vertex cut* of a graph $G = (V, E)$ is a strict subset of vertices $V_0 \subset V$ such that the graph obtained from G by removing V_0 has at least two (non-empty, connected and pair-wise disconnected) components. If the number of components is at least k, the vertex cut V_0 is called a *vertex k-cut*.

Given G and an integer $k \geq 2$, the *vertex k-cut problem* is to find, if it exists, a vertex k-cut of minimum cardinality. Berger, Grigoviev and Zwaan [7] showed that the problem is NP-hard (k being part of the input) but polynomial-time solvable for graphs of bounded treewidth. Ben Ameur and Didi Biha [5] proved that, for $k = 2$, it is polynomial-time solvable as it amounts to computing $|V|^2$ maximum flows. A fixed-parameter algorithm for the vertex k-cut problem, considering the parameter k, whould be an algorithm solving the problem with a running time of the form $f(k) \times poly(|V|)$ where $f(k)$ is any function and $poly(|V|)$ is a polynomial in $|V|$. Marx [21] showed that such an algorithm is unlikely to exist as he proved the $W[1]$-hardness of the problem. However, the complexity for fixed k was an open question. The first contribution of this paper improves on Marx's results by showing that the vertex k-cut problem is NP-hard for any *fixed* $k \geq 3$.

Our second contribution is to investigate the hardness of the problem in practise, we report computational experiments on the problem using Integer Linear Programming (ILP) tools to solve it on DIMACS instances. Despite its basic setting, the vertex k-cut problem has received limited attention according to ILP approaches.

However, several ILP models for variants of the vertex k-cut problem have been studied, see e.g., [6, 9, 17, 2, 15, 8, 11, 12]. Variants where a set of edges is removed to partition the graph, instead of a vertex set, have been also widely studied, see [10, 13, 14, 18, 19, 24] for an overview. Let us detail the literature on the vertex variants.

The *k-separator problem* is a variant where cardinality bounds are required. It consists in finding a vertex cut whose removal gives a graph where the size of each connected component is less than or equal to k. In [6], the authors analyze the complexity on several classes of graphs. They also propose approximation algorithms, a formulation and a polyhedral study. Another variant of this problem exists where the cardinality constraints are not on the size of the connected components but on vertex sets. More precisely, the problem consists in finding a vertex cut V_0 such that $V \setminus V_0$ can be partitioned into two sets of cardinality less than or equal to k, and no edge is incident to both sets. Remark that each set may contain several connected components. This problem is NP-hard even for planar graphs [17] or maximum degree 3 graphs [9]. A first polyhedral study on this problem is done in [2] from which a Branch-and-Cut algorithm is derived [15]. In [8] the authors introduce valid inequalities based on a lower bound given by the number of disjoint paths between all pairs of vertices. For these inequalities, the authors analyze their facial structures and add these inequalities in a Branch-and-Cut algorithm. In the *q-balanced vertex k-separator problem*, the bound is not on the size of the sets but on their differences. More formally, one seeks for a vertex k-cut V_0 such that $V \setminus V_0$ can be partitioned into k pairwise disconnected sets $V_1, \ldots, V_k$ and $|V_i| - |V_j|$ is at most q for all $i \neq j \in \{1, \ldots, k\}$. Different integer linear programming formulations are given in [11]. The *multi-terminal vertex k-cut problem* consists, given a set $T \subset V$ of k terminals, in finding a vertex k-cut V_0 of G containing no terminal such that each connected component of $G[V \setminus V_0]$ contains at most one terminal. In [21] Marx shows also the W[1]-hardness of this problem. A path-based formulation is given in [12] for solving this problem and several inequalities are proposed. A polyhedral analysis is also performed and an efficient Branch-and-Cut algorithm is developed.

The paper is organized as follows. Section 2 is devoted to the NP-hardness proof of the vertex k-cut problem for any fixed $k \geq 3$. In Section 3, we reformulate this problem as a stable set problem with additional constraints. We deduce a compact integer linear program based on this reformulation. In Section 4, we present a formulation with an exponential number of variables and a polynomial number of constraints. We also give a column generation scheme to solve the linear relaxation. We prove the effectiveness of this approach by showing that the subproblem is polynomial-time solvable, first by using submodular function minimization, and second by using flow techniques. Section 5 reports the experimental results we obtain by solving the two formulations, the first one by a general-purpose ILP solver and the second by a Branch-and-Price algorithm. The rest of this section is devoted to notation and assumption.

Notation. Throughout, K denotes the set of integers $\{1, ..., k\}$ and $G = (V, E)$ is a simple undirected graph with $|V| = n$ and $|E| = m$. The complement of S is denoted $\overline{S} = V \setminus S$, and the complement of G is denoted $\overline{G} = (V, \overline{E})$, so $\overline{E} = \{uv : uv \notin E\}$. We say that u and v are *neighbours* if there is an edge $uv \in E$. A subset $W \subseteq V$ of vertices is a *clique of* G, if any two vertices of W are neighbours, and it is a *stable set* of G if it is a clique in $\overline{G}$. The cardinality of the largest stable set of G is denoted by $\alpha(G)$. A subset $W \subseteq V$ of vertices is a *vertex k-multiclique* of G, if there is a k-partition $\pi = \{W_1, ..., W_k\}$ of W such that any two vertices in different sets of π are adjacent in G, with $W_i \neq \emptyset$ for all $i \in \{1, ..., k\}$. For each $W \subseteq V$, we indicate by $\delta(W)$ the subset of edges incident with exactly one vertex in W (i.e., all edges uv with $u \in W,\ v \in V \setminus W$), and with $E(W)$ the subset of edges incident with two vertices in W (i.e., all edges uv with $u, w \in W$). Finally, we indicate by $\delta(v) \subseteq E$ the subset of edges incident with v.

Assumption. In the rest of the paper, we will assume that $\alpha(G) \geq k$. This is clearly a necessary and sufficient condition for G to have a vertex k-cut. We assume that G is connected. We will also use implicitely the basic property that a vertex k-cut V_0 is a set of vertices such that $V \setminus V_0$ can be partitioned into k non-empty subsets $V_1, ..., V_k$ that are pairwise disconnected, i.e., there is no edge between two subsets V_i and V_j for all $i \neq j \in \{1, \ldots, k\}$.

2. Complexity

In this section, the NP-hardness of the vertex k-cut problem for any fixed $k \geq 3$ is proved.

We start by observing that the problem is equivalent to the *vertex k-multiclique problem* which consists, given an undirected graph $G = (V, E)$ and $k \geq 2$ in determining a vertex k-multiclique of maximum cardinality.

Proposition 1 *A vertex subset V_0 of a graph $G = (V, E)$ is a vertex k-cut if and only if $W = V \setminus V_0$ is a vertex k-multiclique in the complement graph $\overline{G}$.*

We now state our complexity result.

Theorem 1 *For any fixed $k \geq 3$, the vertex k-cut problem is NP-hard.*

Proof. In order to prove the theorem, it suffices to prove that the vertex 3-cut problem is NP-hard. Indeed, G has a vertex k-cut of size s if and only if $\tilde{G}$ has a vertex $(k+1)$-cut of size s, where $\tilde{G}$ is obtained by adding an isolated vertex to G. The basic idea of the proof is to reduce an instance of the NP-hard maximum stable set problem in tripartite graphs [22] into an instance of the vertex 3-cut problem.

By Proposition 1, it suffices to prove that the vertex 3-multiclique problem is NP-hard. We actually prove that this problem is already NP-hard in the class of tripartite graphs. To this end, we will use a reduction from the maximum stable set problem in tripartite graphs, which is NP-hard by Lemma 6 in [22]. Let $G = (V_1 \cup V_2 \cup V_3, E)$ be a tripartite instance of the maximum stable set problem. Since every isolated vertex belongs to all maximal stable sets, it is still NP-hard to solve tripartite instances with additional isolated vertices, hence, without loss of generality, we can suppose that V_i contains an isolated vertex v_i for each $i \in \{1, 2, 3\}$. We define the instance $\tilde{G} = (V_1 \cup V_2 \cup V_3, \tilde{E})$ of the 3-multiclique problem where $\tilde{E} = \{uv : u \in V_i, v \in V_j, i \neq j, uv \notin E\}$. (In Figure 2, the white vertices represent a maximal stable set of G (left graph). The same set corresponds to a maximal 3-multiclique on $\tilde{G}$ (right graph).)

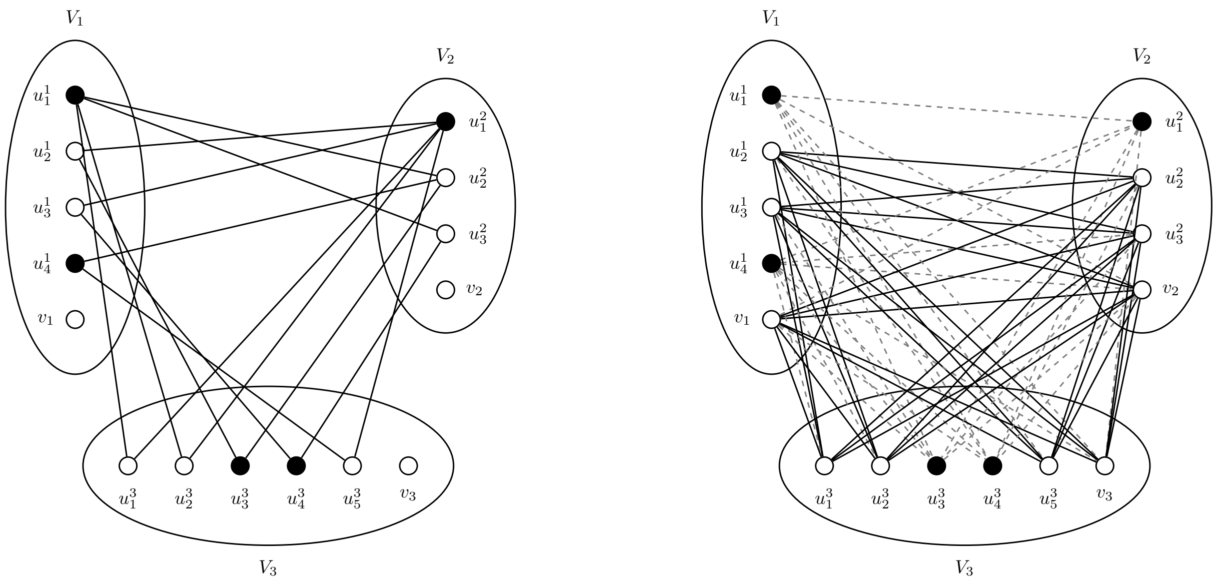


Figure 1: Reduction from the maximum stable set problem in tripartite graphs to the 3-multiclique problem.

We claim that a vertex subset S is a stable set of G containing $\{v_1, v_2, v_3\}$ if and only if S is a vertex 3-multiclique of $\tilde{G}$ containing $\{v_1, v_2, v_3\}$. Indeed, by construction, two vertices $u \in V_i \cap S$ and $v \in V_j \cap S$ where $i \neq j$ are adjacent in $\tilde{G}$. Thus S is a vertex 3-multiclique in $\tilde{G}$. The converse is also true. Since any maximum stable set of G and any maximum vertex 3-multiclique of $\tilde{G}$ contain $\{v_1, v_2, v_3\}$ the proof is done. □

3. Compact formulation

In this section, we show that the vertex k-cut problem can be reformulated as a maximum stable set problem on a specific k-partite graph with additional requirements. We also derive a compact integer linear program based on this reformulation.

Let $G = (V, E)$ and $k \geq 2$ be an instance of the vertex k-cut problem. As previously noted, a subset $V_0 \subset V$ is a vertex k-cut of G if and only if $V \setminus V_0$ can be partionned into k nonempty pairwise disconnected sets. Hence, the vertex k-cut problem is equivalent to finding k nonempty disjoint sets $V_1, \dots, V_k$ of V which are pairwise disconnected such that $|\bigcup_{i \in K} V_i|$ is maximum.

We construct a k-partite graph $G' = (V', E')$ so that the vertex k-cut problem on G reduces to the maximum stable set on G'. Figure 3 gives an illustration of this equivalence. The graph on the left is G. The set V_0 of white vertices corresponds to a 3-vertex cut and $\{V_1, V_2, V_3\}$ with $V_1 = \{v_3\}$, $V_2 = \{v_4\}$ and $V_3 = \{v_2, v_5\}$ is a partition of $V \setminus V_0$ into 3 parwise disconnected sets. The graph on the right corresponds to G'. The white vertices form the stable set associated with $\{V_1, V_2, V_3\}$.

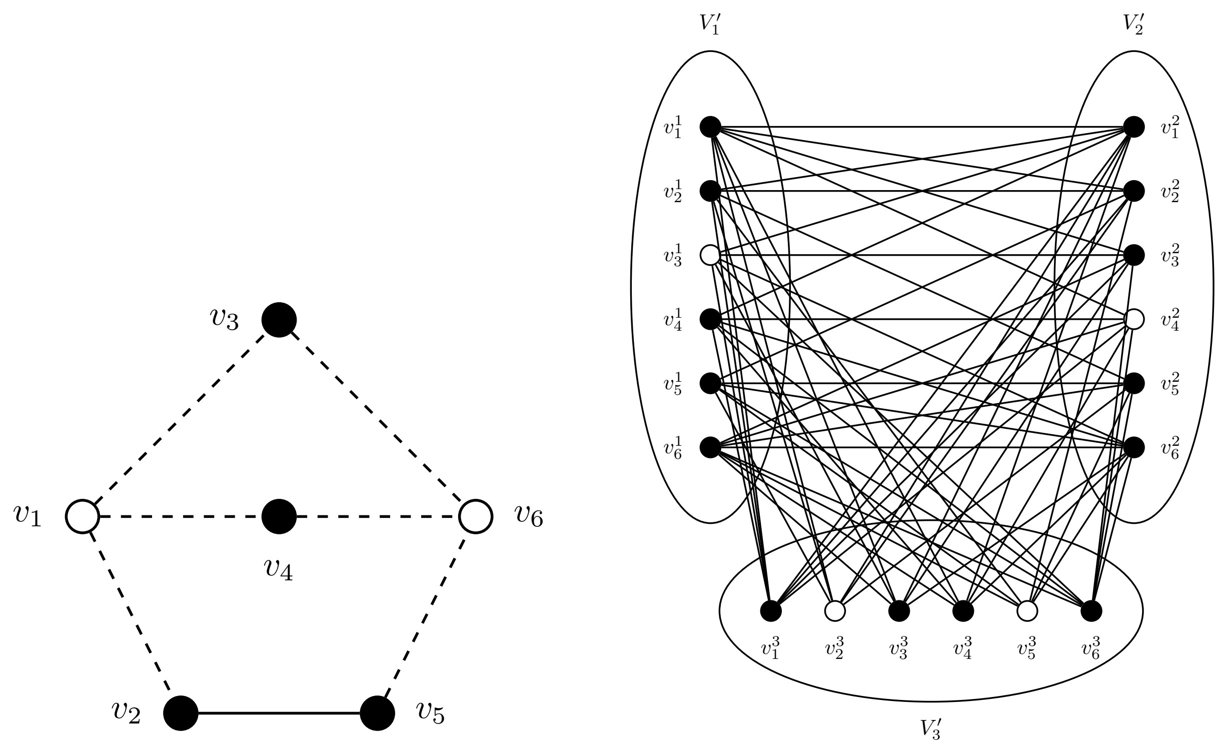


Figure 2: Transformation of the 3-vertex cut problem into a specific maximum stable set problem.

Formally the construction of G' is as follows. The set V' is obtained by considering k copies $v^1, \dots, v^k$ of every vertex $v \in V$. We define the k-partition of V' as $\pi = \{V'_1, \dots, V'_k\}$ with $V'_i = \{v^i : v \in V\}$ for all $i = 1, \dots, k$. In other words, V'_i corresponds to a copy of V. The edge set E' is the union of two sets E'_α and E'_β. $E'_\alpha = \{v^i v^j : i \neq j \in K\}$ is the edge set obtained by considering a clique between all the copies of a same vertex $v \in V$. For E'_β, we consider for each $uv \in E$ an edge between every copy of u and every copy of v. Hence, $E'_\beta = \{u^i v^j : uv \in E, i \neq j \in K\}$. There is a 1-to-1 correspondence between nonempty pairwise disconnected disjoint sets $V_1, \dots, V_k$ of V and stable sets of G' intersecting each V'_i, $i \in K$. Indeed, let $V_1, \dots, V_k$ satisfying the aforementioned requirements. Let $S \subseteq V'$ be the set obtained by taking in V'_i the copies of the vertices in V_i for all $i \in K$. S is a stable set because no edge exists between V_i and V_j and $V_i \cap V_j = \emptyset$ for $i \neq j \in K$. Moreover,

S intersects every V_i', $i \in K$, since $V_1, \dots, V_k$ are nonempty. Finally $S = |\bigcup_{i \in K} V_i|$. The converse also holds which implies the result.

We now give a formulation of the vertex k-cut problem with an integer linear program. By the previous reformulation, we look for a stable set S of G' intersecting every V_i' of the k-partition. For all vertices $v \in V$ and for all integers $i \in K$, let us associate a binary variable x_v^i such that:

$$x_v^i = \begin{cases} 1 & \text{if copy } v^i \in V_i' \text{ of vertex } v \in V \text{ belongs to } S \\ 0 & \text{otherwise} \end{cases} \qquad i \in K, v \in V,$$

the first natural compact ILP formulation (called ILP_C) reads as follows:

$$(\text{ILP}_\text{C}) \qquad \max \sum_{i \in K} \sum_{v \in V} x_v^i \tag{1}$$

$$\sum_{i \in K} x_v^i \leq 1 \qquad v \in V, \tag{2}$$

$$x_u^i + \sum_{j \in K \setminus \{i\}} x_v^j \leq 1 \qquad i \neq j \in K, uv \in E, \tag{3}$$

$$\sum_{v \in V} x_v^i \geq 1 \qquad i \in K, \tag{4}$$

$$x_v^i \in \{0, 1\} \qquad i \in K, v \in V. \tag{5}$$

The objective function maximizes the size of S. Constraints (2) and (3) are the clique constraints associated with cliques of E'_α and edges of E'_β respectively. Constraints (4) impose that S intersects each V_i' for $i \in K$.

By replacing constraints (5) with

$$x_v^i \geq 0 \qquad i \in K,\ v \in V \tag{6}$$

we obtain the Linear Programming relaxation of ILP_C, that will be denoted as LP_C in what follows. Descriptive natural ILP models are known to produce weak linear programming relaxations as the following proposition shows:

Proposition 2 *An optimal solution to LP_C is $x_v^i = \frac{1}{k},\ i = 1, \dots, k,\ v \in V$, and has value $n = |V|$.*

Proof. Constraints (2) impose a trivial upper bound of value n. By setting $x_v^i = \frac{1}{k},\ i = 1, \dots, k,\ v \in V$, the objective function obtains exactly the value n and all the other constraints are satisfied by construction. □

In order to improve the strength of the linear programming relaxation, and to remove the symmetry of ILP_C, we design a new formulation for the vertex k-cut problem.

4. Exponential-size formulation

In this section, we derive an alternative formulation for the vertex k-cut problem having an exponential number of variables with respect to the input size. Let $\mathscr{S} = \{S \subseteq V, S \neq \emptyset\}$ be the family of all non-empty subsets of vertices of V.

For all subsets $S \in \mathscr{S}$, let us associate a binary variable ξ_S such that:

$$\xi_S = \begin{cases} 1 & \text{if } S \text{ corresponds to one of the } k \text{ disconnected subsets of } G \\ 0 & \text{otherwise} \end{cases} \qquad S \in \mathscr{S}.$$

The vertices that do not appear in any selected subset are assigned to the vertex cut. In the following we let $\mathscr{C}$ be an edge-covering family of cliques of G, that is, a family of cliques so that for each edge $uv \in E$, there is at least one clique $C \in \mathscr{C}$ containing both $u, v \in C$. The exponential-size ILP formulation for the vertex k-cut problem reads as follows

$$(\text{ILP}_\text{E}) \qquad \max \sum_{S \in \mathscr{S}} |S| \xi_S \tag{7}$$

$$\sum_{S \in \mathscr{S} : v \in S} \xi_S \leq 1 \qquad v \in V, \tag{8}$$

$$\sum_{S \in \mathscr{S} : C \cap S \neq \emptyset} \xi_S \leq 1 \qquad C \in \mathscr{C}, \tag{9}$$

$$\sum_{S \in \mathscr{S}} \xi_S = k \tag{10}$$

$$\xi_S \in \{0, 1\} \qquad S \in \mathscr{S}. \tag{11}$$

The objective function (7) maximizes the sum of the cardinalities of the selected subsets S of vertices, which is equivalent to minimize the cardinality of the vertex cut. Constraints (8) impose that each vertex $i \in V$ does not appear in more than one of the selected subsets. Constraints (9) impose that, for each clique $C \in \mathscr{C}$, at most one subset containing any vertex of the clique can be selected. Constraint (10) imposes that exactly k subsets are selected. Constraints (11) impose the variables to be binary, so, finally, by relaxing the integrality of constraints (11) to

$$\xi_S \geq 0 \qquad S \in \mathscr{S}, \tag{12}$$

we obtain the Linear Programming relaxation of ILP_E, that is denoted as LP_E in what follows.

4.1 A Branch-and-Price Algorithm

In this section we describe a Branch-and-Price algorithm which is designed to solve ILP_E. The exact algorithm is composed by two main components, i.e., a Column Generation (CG) algorithm to solve LP_E, and a branching scheme. We treat these two aspects in the next sections.

4.1.1 Solving the Linear Programming Relaxation of ILP_E

Model (7)–(11) has exponential size, thus a *column generation* procedure is necessary to solve LP_E.

The *master problem* (MP) can be initialized with the n subsets of V containing a single vertex. Since we assumed that G contains a stable set of cardinality k, this initialization assures the existence of a feasible solution to start the column generation. Additional variables needed to optimally solve the MP are then generated by separating the associated dual constraints. The *pricing problem* (PP) (see, e.g., [16] for definition and more details on column generation) can be solve efficiently as described in the following.

At each column generation step, the optimal values $\lambda^* \in \mathbb{R}_+^V$, $\pi^* \in \mathbb{R}_+^{\mathscr{C}}$, $\gamma^* \in \mathbb{R}$ (respectively) of the dual variables associated with constraints (8), (9), (10) (respectively) are given. The separation of a violated dual constraint is equivalent to find a non-empty subset $S^* \in \mathscr{S}$ such that

$$\sum_{v \in S^*} \lambda_v^* + \sum_{C \in \mathscr{C} : C \cap S^* \neq \emptyset} \pi_C^* + \gamma^* < |S^*|$$

which can be reformulated as

$$\sum_{v \in S^*} \nu_v^* - \sum_{C \in \mathscr{C} : C \cap S^* \neq \emptyset} \pi_C^* > \gamma^*, \tag{13}$$

where $\nu_v^* = 1 - \lambda_v^*$.

If such a subset exists, the corresponding variable ξ_{S^*} is added to the MP, and the procedure is iterated; otherwise, the MP is solved to proven optimality. Hence PP amounts to find a S^* maximizing the left-term in (13) and to check whether or not it is bigger or not than the right-term. It can be modeled as a Binary Linear Program using variables x_v ($v \in V$), which define S^*, and variables y_C ($C \in \mathscr{C}$), each of which takes value 1 if clique C intersects set S^*, as follows:

$$\max \sum_{v \in V} \nu_v^* x_v - \sum_{C \in \mathscr{C}} \pi_C^* y_C \tag{14}$$

$$y_C \geq x_v \qquad v \in C \in \mathscr{C}, \tag{15}$$

$$\sum_{v \in V} x_v \geq 1 \tag{16}$$

$$x_v \in \{0, 1\} \qquad v \in V, \tag{17}$$

$$y_C \in \{0, 1\} \qquad C \in \mathscr{C}. \tag{18}$$

Constraints (15) impose $y_C = 1$ ($C \in \mathscr{C}$) if at least a vertex v of a clique C belongs to S^*; while constraints (16) impose S^* is not empty. If the value of the optimal solution of the PP is larger than γ^*, $S^* = \{v \in V, x_v^* = 1\}$, and the associated variable z_{S^*} is added to the MP. Note that, since $\pi_C \geq 0$ ($C \in \mathscr{C}$) and variables x_v ($v \in V$) are binary, we can relax constraints (18) to $y_C \geq 0$ ($C \in \mathscr{C}$).

The PP can be interpreted as follows: Given $G = (V, E)$, a profit ν_v^* for each $v \in V$ (possibly negative) and a penalty $\pi_C^* \geq 0$ for each $C \in \mathscr{C}$, the problem aims at selecting a non-empty subset of vertices of maximum profit; the penalty π_C^* associated with a clique C is paid if at least one of its vertices is selected. A vertex v with $\nu_v^* \leq 0$ can be removed together with its incident edges. If a clique $C \in \mathscr{C}$ is reduced to a single vertex u by the removal, then, π_C is subtracted from the profit ν_u^* of vertex u. The procedure is iterated until all vertices have positive profit. In case all vertices have negative profit ν^*, or all vertices are removed, the PP problem reduces to finding the vertex $u = \arg\max_{v \in V}\{\nu_v^* - \sum_{C \in \mathscr{C}: v \in C} \pi_C^*\}$.

The following proposition characterises the complexity of the PP.

Proposition 3 *The PP is polynomial-time solvable.*

Proof. In the PP we are looking for a non-empty subset of vertices. Let us define PP$\cup\emptyset$ a relaxation of the PP, where also the empty set is admitted as solution. Given a polynomial-time algorithm for the PP$\cup\emptyset$, we can select a vertex $v \in V$ which is forced to be in S^*, and then apply the algorithm to the subgraph of G induced by $V \setminus \{v\}$. By applying this procedure for each $v \in V$, in n iterations we obtain the optimal solution to the PP.

It remains to show that PP$\cup\emptyset$ is polynomially solvable. This can be done by observing that the PP$\cup\emptyset$ can be formulated by removing constraint (16) from model (14)–(18). The resulting model structure is the same as the generic structure described in [3], model (8). A solution method for the latter model is given in [3]. (The latter consists in solving a min-cut/max-flow problem on a network with one node associated with each variable, plus a source and a sink node as explained below). □

In the case of our problem, we have a network $N = (W, A)$, where the node set is $W = \{s, t\} \cup V \cup \{u_C,\ C \in \mathscr{C}\}$, i.e., in addition to the vertex set V of G, there are a source and a sink node, and a node u_C for each clique $C \in \mathscr{C}$.

The arc set is defined below:

- For each $v \in V$ there is an arc (s, v) with capacity ν_v^*,
- For each $C \in \mathscr{C}$ there is an arc (C, t) with capacity π_C^*,
- For each $C \in \mathscr{C}$ and each $v \in C$, there is an arc (v, C) with infinite capacity.

There is a one-to-one correspondence between st-cuts of finite capacity in N, and feasible solutions of (14)–(18). For a cut of capacity L, the responding solution of (14)–(18) has value $\sum_{C \in \mathscr{C}} \pi_C^* - L$. Hence, an optimal solution to (14)–(18) can be obtained from a min-cut on a network N having $n + |\mathscr{C}| + 2$ nodes. An example of this network is given in the left part of Figure 4 (at the end of Section 4.2).

Corollary 1 *An optimal solution to the linear relaxation LP_E of the exponential-size integer formulation ILP_E of the minimum vertex k-cut problem can be computed in polynomial time.*

Proof. Since the pricing problems PP ask for the solution of $n = |V|$ min-cut problems, then solving the master MP (and, eventually, $\mathrm{LP_E}$) is polynomial time solvable. □

4.1.2 Pricing as submodular function minimization

The PP$\cup\emptyset$ (i.e., the relaxation of the PP where also the empty set is admitted as solution) can also be tackled as the minimization of a submodular function and hence is polynomial-time solvable [23]. A list of submodular functions is reported in a list in [23] (section 44.1a).

First, given $S \subseteq V$, let us define the two following clique families:

$$I(S) := \{C \in \mathscr{C} : C \cap S \neq \emptyset\} \quad \text{and} \quad C(S) := \{C \in \mathscr{C} : C \subseteq S\}$$

Second, we use the short-hand notation:

$$\nu^*(S) := \sum_{v \in S} \nu^*_v \qquad \pi^*(\mathscr{C}') := \sum_{C \in \mathscr{C}'} \pi^*_C$$

The PP$\cup\emptyset$ can be formulated as

$$\max_{S \subseteq V} \nu^*(S) - \pi^*(I(S))$$

Observe that $C \in I(S)$ if and only if $C \notin C(\overline{S})$. Hence a set S maximizes $\nu^*(S) - \pi^*(I(S))$ if and only if its complementary set minimizes the set function

$$f(S) := \nu^*(S) - \pi^*(C(S)) \tag{19}$$

Proposition 4 implies that the set function $f(\cdot)$ is both submodular and supermodular.

Proposition 4 $f(S) + f(T) = f(S \cap T) + f(S \cup T)$, *for every* $S, T \subseteq V$.

Proof. Obviously, $\nu^*(S) + \nu^*(T) = \nu^*(S \cap T) + \nu^*(S \cup T)$. Clearly, $\pi^*(C(S)) + \pi^*(C(T)) = \pi^*(C(S \cap T)) + \pi^*(C(S \cup T))$. □

Finally, let us mention that the constraint matrix of (15) is totally unimodular, as observed in [11] for the case of edge constraints. This gives a third proof of polynomial-time solvability for the PP.

4.1.3 Pricing as a min-cut on a smaller network

In case $\mathscr{C}$ is exactly the set of edges of the graph G (which, for instance, the only possible form of $\mathscr{C}$ for a triangle free graph), a solution method based on solving a min-cut problem on a smaller network can be exploited.

First observe that, since the cliques in $\mathscr{C}$ are in fact the edges of G, then $I(S) \setminus C(S) = \delta(S)$ for all $S \subseteq V$. Furthermore, (19) can be rewritten in standard notation as $f(S) = \nu^*(S) - \pi^*(E(S))$, and since

$$2\pi^*(E(S)) + \pi^*(\delta(S)) \quad = \quad \sum_{v \in S} \pi^*(\delta(v)),$$

the PP$\cup\emptyset$ is equivalent to minimizing $2\nu^*(S) + \pi^*(\delta(S)) - \sum_{v \in S} \pi^*(\delta(v))$. Observe that the equation below holds where the third term in the last expression is a constant:

$$2\nu^*(S) + \pi^*(\delta(S)) - \sum_{v \in S} \pi^*(\delta(v)) \quad = \quad 2\nu^*(S) + \pi^*(\delta(S)) - \sum_{v \in V} \pi^*(\delta(v)) + \sum_{v \in S} \pi^*(\delta(v))$$

Hence, actually, the PP∪∅ amounts to

$$\min 2\nu^*(S) + \pi^*(\delta(S)) + \sum_{v \in S} \pi^*(\delta(v))$$

This problem can be solved as a min-cut problem on network with source node s, sink node t, one node for each $v \in V$ and the arc set defined below:

- For each $v \in V$ there is an arc (s, v) with capacity $2\nu_v^*$;
- For each edge $uv \in E$, there are an arc uv and an arc vu with capacity π_{uv}^*;
- For each node $v \in V$ there is an arc (v, t) with capacity $\pi^*(\delta(v))$.

In this case, the PP∪∅ can be solved in polynomial-time as a min-cut problem (equivalent to max-flow) on the above described network, having $n+2$ nodes. An example of this network is given in the right part of Figure 4 (at the end of the section).

4.1.4 A branching scheme for $\mathbf{ILP_E}$

When the optimal solution of the master problem (MP) associated with the linear relaxation of model $\mathrm{ILP_E}$ for the min vertex k-cut problem is fractional, a branching scheme is necessary in order to obtain an integer solution.

Let ξ^* be the current (fractional) solution of the MP. A two-level branching scheme has to be considered. First we branch by imposing that, for each vertex $v \in V$, either v is in the vertex k-cut V_0 or it belongs to the vertex-set S of some component of the subgraph of G induced by $V \setminus V_0$. This is in general not enough to define an integer solution, indeed, even if the vertex k-cut V_0 is well defined by

$$V_0 = \{v \in V : \sum_{S \in \mathscr{S} : v \in S} \xi_S^* = 0\}$$

it does not imply that the solution is 0-1 valued. We impose a second level branching, where, for two vertices u and v outside V_0, we impose that either u and v are in the same component, or they belong to different ones.

In the first branching, for each vertex $v \in V$, we check if it is partially included in the components and the vertex cut, more precisely, if

$$0 < \sum_{S \in \mathscr{S} : v \in S} \xi_S^* < 1. \tag{20}$$

In case of multiple partially included vertices, we branch on the vertex v for which the sum in (20) is closer to 1. Ties are broken randomly. Two subproblems are then created from the current one:

- in the first subproblem, we impose that v is in the vertex cut, by modifying the associated constraint (8) to
$$\sum_{S \in \mathscr{S} : v \in S} \xi_S = 0;$$
we also modify the pricing procedure in order to forbid the selection of vertex v by modifying the cost structure of the associated min-cut problem;

- in the second subproblem, we impose that v is not in the vertex cut, by modifying the associated constraint (8) to
$$\sum_{S \in \mathscr{S}: v \in S} \xi_S = 1;$$
the pricing procedure is unchanged.

Once V_0 is defined, then ξ^* is still fractional if and only if we can find two vertices u, v so that
$$0 < \sum_{S \in \mathscr{S}: u,v \in S} \xi^*_S < 1. \tag{21}$$
(It holds more generally for 0-1 constraints of the form $A\xi^* = \mathbf{1}$, see [4]).

In case more than one such pair of vertices exist, we branch on the pair for which the sum in inequality (21) is closer to 1. Ties are broken randomly. Two subproblems are then created from the current one:

- in the first subproblem, we impose that u and v are in the same component; this can be obtained by contracting $\{u, v\}$ in the pricing subproblem, that is, creating a supervertex w representing both u and v and such that $\delta(w) = \delta(u) \cup \delta(v)$ (and removing u, v);

- in the second subproblem, we impose that u and v are in different components; this can be obtained by adding to the pricing subproblem an incompatibility constraint between u and v. In this case the subproblem cannot be formulate as a min-cut/max-flow problem, and we have to solve the MIP formulation (14)–(18) with the additional constraint
$$x_v + x_u \leq 1.$$
In this case the modified pricing problem might be NP-hard.

In our Branch-and-Price algorithm we first define the vertices in the vertex cut, i.e., we apply the first branching rule. Then, in case the solution is still fractional, we apply the second branching rule. After branching, the variables that are incompatible with the branching decision are removed from the children nodes. The following proposition states that the two proposed branching rules define a complete branching scheme for ILP_E:

Proposition 5 *The two branching rules applied in sequence provide a complete branching scheme for model ILP_E.*

Proof. The rows of the constraints (8) associated with vertices forced out of the vertex cut, after the application of the first branching rule, are equalities with binary coefficients and right-and-side equal to 1. In this case, if a basic solution ξ^* is fractional, then there exist u and v such that (21) holds. This result allows to conclude that, if a solution is fractional after the first branching rule is applied, then we can determine two vertices for applying the second branching rule. □

4.2 Examples and comparision

This section discusses the relation between the two formulations proposed for the vertex k-cut problem.

Proposition 6 *Even when $\mathscr{C} = E$, the bound for the vertex k-cut problem provided by the optimal solution value of the extended formulaiton LP_E strictly dominates the corresponding bound provided by the compact formulation LP_C.*

Proof. Given a feasible solution $\tilde{\xi}$ of $\mathrm{LP_E}$, we can construct a feasible solution $\tilde{x}$ of $\mathrm{LP_C}$ with same objective function value as follow:

$$\tilde{x}_v^i = \frac{1}{k} \sum_{S \in \mathscr{S}: v \in S} \tilde{\xi}_S \qquad i \in K, v \in V.$$

We first show that the two solutions have the same objective function value:

$$\sum_{i \in K} \sum_{v \in V} \tilde{x}_v^i = \sum_{k \in K} \frac{1}{k} \sum_{v \in V} \sum_{S \in \mathscr{S}: v \in S} \tilde{\xi}_S = \sum_{S \in \mathscr{S}} \sum_{v \in S} \tilde{\xi}_S = \sum_{S \in \mathscr{S}} |S| \tilde{\xi}_S.$$

It is straightforward to check that constraints (2) are satisfied. For each edge $uv \in E$ and for $i \neq j \in K$:

$$\tilde{x}_u^i + \sum_{j \in K \setminus \{i\}} \tilde{x}_v^j = \left(\frac{1}{k} \sum_{S \in \mathscr{S}: u \in S} \tilde{\xi}_S \quad + \frac{k-1}{k} \sum_{S \in \mathscr{S}: v \in S} \tilde{\xi}_S \right) \leq 1,$$

i.e., constraints (3) are satisfied. Finally constraints (4) are satisfied since for each $i \in K$:

$$\sum_{v \in V} \tilde{x}_v^i = \sum_{v \in V} \frac{1}{k} \sum_{S \in \mathscr{S}: v \in S} \tilde{\xi}_S \geq \frac{1}{k} \sum_{S \in \mathscr{S}} \tilde{\xi}_S = 1$$

To see that the domination can be strict, consider now solving the vertex k-cut problem with $k = 3$ for a cycle of 6 vertices. An optimal solution to $\mathrm{LP_C}$ is $x_v^i = \frac{1}{3}$, $v \in V$, $i = 1, \ldots, 3$, with value 6, while an optimal solution to $\mathrm{LP_E}$ has value 3. □

In the remaining of this section we discuss with an example the quality of the linear relaxation of $\mathrm{ILP_E}$, when constraints (9) are expressed for a family of cliques $\mathscr{C}$ or for the edge set E, respectively.

Let us consider the graph $G = (V, E)$ of Figure 3. The example graph has 6 vertices $(v_1, v_2, v_3, v_4, v_5, v_6)$ and 6 edges $(v_1v_2, v_1v_3, v_1v_5, v_3v_4, v_4v_5, v_5v_6)$. One optimal solution to the min vertex 3-cut problem is obtained by removing vertices v_3 and v_5, and the maximum in $\mathrm{ILP_E}$ is then 4. By defining the clique family $\mathscr{C} = \{\{v_1, v_3, v_5\}, \{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\}\}$, the optimal solution of $\mathrm{LP_E}$ is integer of value 4 and it is given by $\xi_{S_1} = \xi_{S_2} = \xi_{S_3} = 1$ where $S_1 = \{v_4\}, S_2 = \{v_6\}, S_3 = \{v_1, v_2\}$. If we consider instead $\mathscr{C} = E$, the optimal solution of $\mathrm{LP_E}$ is not integer and has value 4.5. This second solution is given by $\xi_{S_1} = \xi_{S_2} = \xi_{S_3} = \xi_{S_3} = \xi_{S_4} = \xi_{S_6} = 0.5$ where $S_4 = \{v_2\}, S_5 = \{v_5, v_6\}, S_6 = \{v_3, v_4\}$.

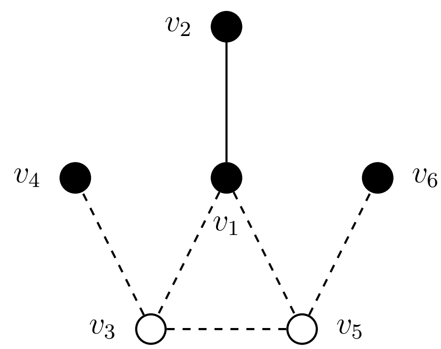


Figure 3: A graph G and an optimal solution to the vertex k-cut problem with $k = 3$. Removing the white vertices disconnects G in 3 components.

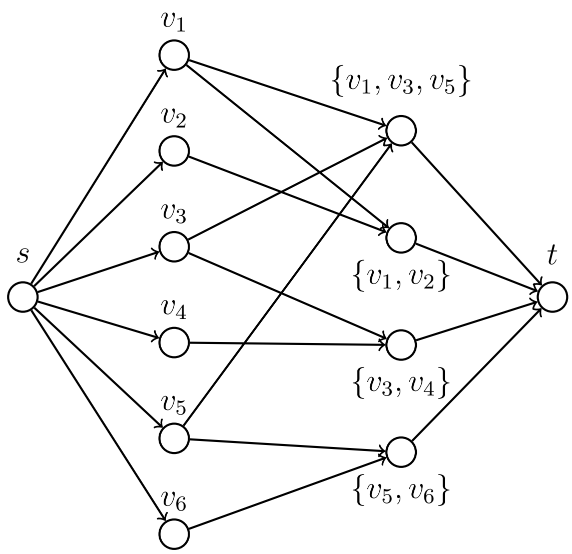
This example shows a case where a strictly better bound is obtained by considering maximal cliques in $\mathscr{C}$.

In Figure 4, we depict the two networks described in Section 4.1.1 and in Section 4.1.3, respectively, for the graph of Figure 3. On the left part of figure, we present the network used for the pricing problem in case the ILP_E is formulated with the clique family $\mathscr{C} = \{\{v_1, v_3, v_5\}, \{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\}\}$. The min-cut/max-flow problem in this case is solved for a network of 12 vertices and 19 arcs. On the right part of the figure, we present the network used for the pricing problem in case the ILP_E is formulated just using edges. The min-cut/max-flow problem in this case is solved for a network of 8 vertices and 24 arcs. As far as the number of nodes of the two networks is concerned, the second one is always smaller than the first it does not have vertices related to the edges of the original graph G. The second network is more effective for triangle-free graphs since, except for trees and trees plus one edge, it has $2(|V| + |E|)$ arcs while the first network has $|V| + 3|E|$ arcs.

5. Computational experiments

To the best of our knowledge, no previous computational study on exact approaches for the vertex k-cut problem appeared in the literature. Thus, with these experiments we wish to evaluate:

- The computational performance of the compact formulation ILP_C of Section 3, solved via a general purpose ILP solver;
- The computational performance of the extended formulation ILP_E of section 4, solved via the Branch-and-Price algorithm described in Section 4.1;
- The size of solvable vertex k-cut problem instances, in terms of number of vertices of the graph;
- The effect of the number of subsets k of the partition on the relative performance of the two mentioned exact methods.



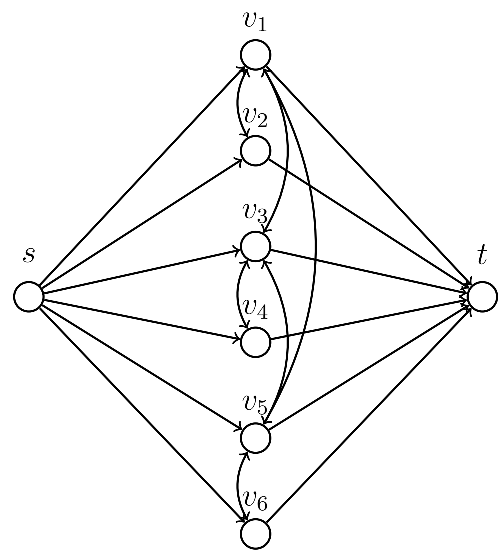


Figure 4: The two pricing networks associated to the example graph of Figure 3.

Experimental setting. The compact formulation ILP_C is enhanced by a preprocessing phase in which a subset of variables is removed so as to reduce the symmetry of the formulation and to improve the quality of the associated linear programming relaxation. In this preprocessing, we search for $k-1$ vertex-disjoint cliques $C_1, \ldots, C_i, \ldots, C_{k-1}$ of the graph G, and remove the following variables

$$x_v^h, \quad i = 1, \ldots, k-1, \quad v \in C_i, \quad h = i+1, \ldots, k.$$

The resulting model is then solved by using the MIP solver of `Cplex 12.6.0` in single-thread mode and default parameter setting. The resulting solution method is denoted as `Cplex + reduction` in what follows.

The extended formulation ILP_E is solved via the Branch-and-Price algorithm, initialized with n variables ξ_S, where $S = \{v\},\ v \in V$. At each column-generation iteration, linear programs are solved with `Cplex 12.6.0`. The pricing subproblem, formulated as a min-cut/max-flow problem, is solved by means of the pre-flow algorithm by Goldberg and Tarjan [20]. We very rarely observed a branching requiring to solve the subproblem as a MIP (i.e., introducing incompatibility constraints between vertices). The exploration of the branching tree is performed in a depth-first fashion.

The experiments have been performed on a computer with a 3.40 Ghz 8-core Intel Core i7-3770 processor and 16Gb RAM, running a 64 bits Linux operating system. Both exact approaches were tested with a time limit of 3600 seconds of computing time.

Test-bed of instances. In the computational experiments, we considered two sets of classical graph instances, having up to 150 vertices, listed in Table 1. In the table, after the instance name, we report the number of vertices n, the number of edges m, the density d, and the size of largest stable set in the graph $\alpha(G)$. This last parameter determines whether a graph instance is feasible for a given value of k, i.e., $\alpha(G) \geq k$; and the corresponding

	n	m	d	$\alpha(G)$		n	m	d	$\alpha(G)$
$2nd$-DIMACS					$2nd$-DIMACS				
myciel3	11	20	36.36	5	myciel6	95	755	16.91	47
myciel4	23	71	28.06	11	queen8_12	96	1368	30.00	8
queen5_5	25	160	53.33	5	mug100_1	100	166	3.35	33
1-FullIns_3	30	100	22.99	14	mug100_25	100	166	3.35	33
queen6_6	36	290	46.03	6	queen10_10	100	1470	29.70	10
2-Insertions_3	37	72	10.81	18	4-FullIns_3	114	541	8.40	55
myciel5	47	236	21.83	23	games120	120	638	8.94	22
queen7_7	49	476	40.48	7	queen11_11	121	1980	27.27	11
2-FullIns_3	52	201	15.16	25	r125.1	125	209	2.70	49
3-Insertions_3	56	110	7.14	27	DSJC125.1	125	736	9.50	34
queen8_8	64	728	36.11	8	r125.5	125	3838	49.52	5
1-Insertions_4	67	232	10.49	32	DSJC125.5	125	3891	50.21	10
huck	74	301	11.14	27	r125.1c	125	7501	96.79	7
4-Insertions_3	79	156	5.06	39	miles250	128	387	4.76	44
jean	80	254	8.04	38	miles500	128	1170	14.39	18
3-FullIns_3	80	346	10.95	37	miles750	128	2113	26.00	12
queen9_9	81	1056	32.59	9	miles1000	128	3216	39.57	8
david	87	406	10.85	36	miles1500	128	5198	63.95	5
mug88_1	88	146	3.81	29	anna	138	493	5.22	80
mug88_25	88	146	3.81	29	queen12_12	144	2596	25.21	12
1-FullIns_4	93	593	13.86	45	2-Insertions_4	149	541	4.91	74
$10th$-DIMACS					$10th$-DIMACS				
karate	34	78	13.90	20	polbooks	105	441	8.08	43
chesapeake	39	170	22.94	17	adjnoun	112	425	6.84	53
dolphins	62	159	8.41	28	football	115	613	9.35	21
lesmis	77	254	8.68	35					

Table 1: Instance Features

stable set provides a feasible vertex k-cut problem solution. Instances with $\alpha(G) < 5$ were removed from the test bed, since they allow a feasible vertex k-cut problem solution only for small values of k. The first set is composed by 42 instances originally proposed for Maximum Clique, Graph Coloring, and Satisfiability in the second DIMACS challenge [1]. They have from 11 to 149 vertices, with densities varying from 3.35 to 96.79. The $\alpha(G)$ parameter varies from 5 to 80. The second set is composed by 7 instances originally proposed for Graph Partitioning and Graph Clustering in the tenth DIMACS challenge [1]. They have from 34 to 115 vertices, with densities varying from 6.84 to 22.94. The $\alpha(G)$ parameter varies from 17 to 53.

Computational performance. In Tables 2 and 3 we consider values of $k = 5, 10, 15, 20$, and report, for `Branch and Price` and `Cplex + reduction`, the CPU time is seconds (*tl* for time limit) and the associated number of explored nodes. For each instance and for each value of k, we report in bold the fastest method. Missing lines correspond to infeasible instances. At the end of each block, we report the number of instances solved to optimality by each method, with respect to the total.

- For $k = 5$, there are 42 $2nd$-DIMACS and 7 $10th$-DIMACS instances, 49 feasible instances in total. For 9 instances, no method can find the optimal solution within time limit; the `Branch and Price` can solve 26 out of 49 instances and is the fastest method in 9 cases; the `Cplex + reduction` can solve 40 out of 49 instances and is the fastest method in 30 cases; 13 instances are solved by `Cplex + reduction` while `Branch and Price` fails. For the solved instances, the number of nodes explored by the `Branch and Price` is not larger than 9651 but typically smaller than 100, `Cplex + reduction` in contrast tends to explore a much larger number of nodes (up to 365825), and on average needs thousands of nodes.

- For $k = 10$, there are 31 $2nd$-DIMACS and 7 $10th$-DIMACS instances, 38 feasible instances in total. For 10 instances, no method can find the optimal solution within time limit; the `Branch and Price` can solve 24 out of 38 instances and is the fastest method in 18 cases; the `Cplex + reduction` can solve 20 out of 38 instances and is the fastest method in 10 cases; 8 instances are solved by `Branch and Price` while `Cplex + reduction` fails; 4 instances are solved by `Cplex + reduction` while `Branch and Price` fails. For the solved instances, the number of nodes explored by the `Branch and Price` is not larger than 288, `Cplex + reduction` explores up to 405857 nodes, and on average needs much more nodes to solve the same graph instance for $k = 10$ than for $k = 5$.

- For $k = 15$, there are 24 $2nd$-DIMACS and 7 $10th$-DIMACS instances, 31 feasible instances in total. For 8 instances, no method can find the optimal solution within time limit; the `Branch and Price` can solve 21 out of 31 instances and is the fastest method in 20 cases; the `Cplex + reduction` can solve 13 out of 31 instances and is the fastest method in 2 cases; 9 instances are solved by `Branch and Price` while `Cplex + reduction` fails; 1 instance is solved by `Cplex + reduction` while `Branch and Price` fails. For the solved instances, the number of nodes explored by the `Branch and Price` is not larger than 82. `Cplex + reduction` needs to explore on average several thousands of nodes.

- For $k = 20$, there are 22 $2nd$-DIMACS and 6 $10th$-DIMACS instances, 28 feasible instances in total. For 8 instances, no method can find the optimal solution within time limit; the `Branch and Price` can solve 20 out of 28 instances and is the fastest method in 19 cases; the `Cplex + reduction` can solve 7 out of 28 instances and is the fastest method in 1 case; 12 instances are solved by `Branch and Price` while `Cplex + reduction` fails. For the solved instances, the number of nodes explored by the `Branch and Price` is not larger than 249. `Cplex + reduction` needs to explore on average several thousands of nodes and is able to solve only few instances. All instances solved by `Branch and Price` require less than 12.23 CPU seconds, except one that needs 460.07 seconds.

From these results we can conclude that `Cplex + reduction` has an average good performance for $k = 5$, and has increasing difficulties for larger values of k. A partial explanation can be found in the increase in the number of variables (n more variables for each incremental value of k). For $k = 5$, `Cplex + reduction` outperforms `Branch and Price`. For `Branch and Price`, an opposite behaviour is experienced when increasing the value of k. In this case, the performance of the method is improved. For example, instance `polbooks` needs 2036.05 CPU seconds for $k = 5$, while 330.90, 25.94, and 3.13 CPU seconds are needed for

$k = 10, 15$ and 20, respectively. For $k = 10, 15$ and 20, `Branch and Price` outperforms then `Cplex + reduction`.

Gaps. In Table 4 we report, for each value of k, the value of the optimal or best known solution (column *opt**), and the linear relaxation and optimality gaps for the $10th$-DIMACS instances. The linear programming relaxation *lp gap* is computed with respect to the optimal solution value *opt* as $100 \cdot \frac{lp_{val} - opt}{opt}$, where lp_{val} is the value of the linear programming relaxation of the corresponding formulation. For instances for which the optimal solution value is not known, the *lp gap* is not reported. A "−" is reported when the time limit is incurred before the linear relaxation is computed. The optimality gap *opt gap* is computed as $100 \cdot \frac{UB_{val} - LB_{val}}{UB_{val}}$, where UB_{val} and LB_{val} are the values of the best upper bound and of the incumbent solution of the corresponding method when the time limit is reached (0.00 for solved instances). The same figures are omitted for the $2th$-DIMACS instances, because they have a similar pattern.

From the table we observe that the formulation ILP_E is characterized by a much stronger linear programming relaxation. The value of its *lp gap* is not affected by the value of k, and ranges between 0.0 and 6.43. This explains the fact that `Branch and Price` explores on average a much smaller number of nodes, and justifies the computational effort spent in column generation. On the other hand, the quality of the *lp gap* of ILP_C deteriorates when k increases, and can be as large as 42.60. Clearly, computing this bound is associated with a smaller computational effort, and many nodes can be explored in short CPU time. For $k = 5$, the generic cuts embedded in the `cplex` MIP solver compensate the poor quality of the linear relaxation, while starting from $k = 10$ the increasing *lp gap* cannot be effectively reduced and `Cplex + reduction` struggles in solving the associated instances.

6. Conclusions

In this paper we considered the minimum vertex k-cut problem, a variant of graph partitioning which consists in finding a vertex k-cut of minimum cardinality. We studied two alternative ILP formulations and analysed their properties in terms of linear programming relaxation. The first formulation is a natural compact formulation while the second one is an exponential-size formulation which requires Column Generation techniques to be effectively solved. We proposed a Branch-and-Price algorithm and we showed how to solve the Linear Programming relaxation of the exponential-size formulation in polynomial time via a series of Min-Cut Max-Flow problems. We computationally compared the performances of the two formulations on benchmark instances from the literature. The outcome of these experiments is that the Branch-and-Price algorithm outperforms the direct use of a general-purpose ILP solver on the compact formulation for large values of k (high number of disconnected subsets of the partition). For small values of k instead, directly tackling the compact formulation remains the best option.

References

[1] Dimacs implementation challenges. `http://http://dimacs.rutgers.edu/Challenges/`. Accessed: 2017-07-01.

	$k=5$				$k=10$			
	`Branch and Price`		`Cplex + reduction`		`Branch and Price`		`Cplex + reduction`	
	time	nodes	time	nodes	time	nodes	time	nodes
`myciel3`	0.00	5	0.00	41				
`myciel4`	**0.15**	24	0.30	330	**0.05**	26	1.88	2241
`queen5_5`	0.05	34	**0.01**	0				
`1-FullIns_3`	0.99	45	**0.36**	229	**0.21**	24	1.76	2464
`queen6_6`	**0.92**	304	1.18	577				
`2-Insertions_3`	**0.35**	26	1.13	1210	**0.03**	2	60.79	43344
`myciel5`	782.73	38	**2.50**	1423	95.34	106	**48.98**	12739
`queen7_7`	1941.08	9651	**40.04**	17950				
`2-FullIns_3`	*tl*	543	**2.52**	1481	**66.57**	117	235.00	160892
`3-Insertions_3`	**6.59**	38	9.78	5848	**0.49**	15	*tl*	1343150
`queen8_8`	*tl*	6729	**904.87**	365825				
`1-Insertions_4`	1659.91	82	**6.30**	2139	**12.74**	29	1242.08	394816
`huck`	0.20	6	**0.03**	0	**0.16**	8	5.90	4745
`4-Insertions_3`	76.38	54	**24.41**	11224	**6.11**	32	*tl*	654743
`jean`	0.38	10	**0.04**	0	0.29	2	**0.24**	6
`3-FullIns_3`	*tl*	54	**30.69**	8351	*tl*	190	**2621.40**	405857
`queen9_9`	*tl*	3657	*tl*	692134				
`david`	1927.06	10	**0.03**	0	21.26	5	**0.08**	0
`mug88_1`	**0.44**	1	54.96	39195	**0.62**	1	*tl*	1011301
`mug88_25`	**1.45**	12	30.55	20791	**0.51**	3	*tl*	1225319
`1-FullIns_4`	*tl*	8	**33.70**	3425	*tl*	8	*tl*	162882
`myciel6`	*tl*	4	**72.64**	7904	*tl*	3	**2000.39**	183177
`queen8_12`	*tl*	2768	*tl*	385175				
`mug100_1`	**2.09**	14	113.58	99468	**2.71**	33	*tl*	810120
`mug100_25`	**2.51**	26	160.96	141665	**1.35**	7	*tl*	1081903
`queen10_10`	*tl*	2507	*tl*	374396	**0.01**	3	1.06	0
`4-FullIns_3`	*tl*	17	**91.15**	9053	*tl*	17	*tl*	92823
`games120`	*tl*	67	*tl*	763536	*tl*	690	*tl*	262064
`queen11_11`	*tl*	2120	*tl*	201943	*tl*	8639	*tl*	39063
`r125.1`	49.66	1	**0.07**	0	372.47	1	**0.13**	0
`DSJC125.1`	*tl*	14	*tl*	323047	*tl*	14	*tl*	44951
`r125.5`	*tl*	52	**1992.94**	178069				
`DSJC125.5`	*tl*	14	*tl*	151754	*tl*	141	*tl*	5380
`r125.1c`	*tl*	12	**1.21**	32				
`miles250`	38.30	1	**0.04**	0	2.74	1	**0.22**	0
`miles500`	**5.10**	1	117.71	14803	**873.90**	200	*tl*	242578
`miles750`	*tl*	5	**639.56**	49132	*tl*	155	*tl*	188792
`miles1000`	*tl*	4	**1012.08**	200247				
`miles1500`	*tl*	1	**6.78**	465				
`anna`	*tl*	8	**0.15**	0	*tl*	17	**0.42**	0
`queen12_12`	*tl*	1631	*tl*	96305	*tl*	7733	*tl*	11992
`2-Insertions_4`	*tl*	5	**236.10**	17285	*tl*	4	*tl*	46205
solved	21/42		34/42		19/31		15/31	
`karate`	0.11	13	**0.03**	0	**0.03**	4	0.06	0
`chesapeake`	1.28	86	**0.80**	793	**0.10**	15	11.20	11755
`dolphins`	0.72	1	**0.29**	30	**0.07**	4	8.91	2887
`lesmis`	17.53	11	**0.14**	0	1.01	2	**0.42**	8
`polbooks`	2036.05	359	**58.83**	20170	**330.90**	288	*tl*	631201
`adjnoun`	*tl*	1	**1.88**	40	*tl*	10	**139.90**	7030
`football`	*tl*	35	*tl*	615634	*tl*	149	*tl*	189697
solved	5/7		6/7		5/7		5/7	

Table 2: Formulation performance comparison ($k = 5$ and $k = 10$)

	$k=15$				$k=20$			
	Branch and Price		Cplex + reduction		Branch and Price		Cplex + reduction	
	time	nodes	time	nodes	time	nodes	time	nodes
2-Insertions_3	**0.10**	22	683.56	676931				
myciel5	**35.87**	82	304.86	71515	**1.71**	33	2434.94	1105526
2-FullIns_3	**1.03**	25	191.99	29987	**1.46**	40	992.17	382558
3-Insertions_3	**0.51**	14	*tl*	783072	**0.44**	17	*tl*	766107
1-Insertions_4	**36.48**	49	*tl*	143540	**12.33**	50	*tl*	104232
huck	**0.14**	4	14.76	6619	**0.07**	2	4.98	1745
4-Insertions_3	**4.29**	29	*tl*	369502	**3.13**	38	*tl*	201377
jean	**0.63**	8	0.64	0	**0.33**	7	5.20	2085
3-FullIns_3	*tl*	79	*tl*	182041	**460.07**	249	*tl*	105272
david	**0.27**	2	14.75	4111	**1.73**	18	*tl*	2581828
mug88_1	**1.14**	21	*tl*	540602	**0.85**	13	*tl*	348214
mug88_25	**0.49**	1	*tl*	459082	**1.10**	33	*tl*	218404
1-FullIns_4	*tl*	14	*tl*	72206	*tl*	22	*tl*	25797
myciel6	*tl*	3	*tl*	59139	*tl*	6	*tl*	25128
mug100_1	**1.73**	14	*tl*	413118	**1.51**	12	*tl*	247039
mug100_25	**1.97**	22	*tl*	376089	**1.64**	18	*tl*	217955
4-FullIns_3	*tl*	14	*tl*	69442	*tl*	33	*tl*	16830
games120	*tl*	1779	*tl*	139497	*tl*	1	*tl*	81290
r125.1	**0.96**	1	275.62	35565	**2.32**	1	*tl*	526415
DSJC125.1	*tl*	15	*tl*	18733	*tl*	22	*tl*	4915
miles250	**0.74**	1	*tl*	614993	**1.85**	8	*tl*	397307
miles500	*tl*	1769	*tl*	156294				
anna	57.52	7	**0.78**	17	84.68	7	**2.06**	30
2-Insertions_4	*tl*	8	*tl*	24929	*tl*	8	*tl*	6093
solved	16/24		8/24		16/22		5/22	
karate	**0.02**	4	0.06	0	**0.01**	3	0.08	100
chesapeake	**0.06**	9	8.46	4903				
dolphins	**0.16**	8	316.13	195342	**0.10**	4	*tl*	1688935
lesmis	**0.58**	6	1.23	804	**0.41**	4	7.44	2226
polbooks	**25.94**	44	*tl*	279125	**3.13**	11	*tl*	267568
adjnoun	*tl*	12	**2337.24**	157113	*tl*	28	*tl*	90669
football	*tl*	1544	*tl*	117103	*tl*	9228	*tl*	57614
solved	5/7		5/7		4/6		2/6	

Table 3: Formulation performance comparison ($k = 15$ and $k = 20$)

	$k=5$					$k=10$				
		`Branch and Price`		`Cplex + reduction`			`Branch and Price`		`Cplex + reduction`	
	*opt**	*lp gap*	*opt gap*	*lp gap*	*opt gap*	*opt**	*lp gap*	*opt gap*	*lp gap*	*opt gap*
`karate`	32	1.42	0.00	5.25	0.00	30	1.76	0.00	10.99	0.00
`chesapeake`	32	6.43	0.00	17.54	0.00	27	4.26	0.00	30.13	0.00
`dolphins`	60	0.00	0.00	3.22	0.00	55	0.00	0.00	11.29	0.00
`lesmis`	76	0.56	0.00	1.26	0.00	75	0.88	0.00	2.57	0.00
`polbooks`	97	3.21	0.00	7.61	0.00	90	4.50	0.00	14.28	4.63
`adjnoun`	110	-	52.68	1.77	0.00	106	0.00	50.00	5.34	0.00
`football`	94		10.88		6.40	71		22.09		30.00
	$k=15$					$k=20$				
		`Branch and Price`		`Cplex + reduction`			`Branch and Price`		`Cplex + reduction`	
	*opt**	*lp gap*	*opt gap*	*lp gap*	*opt gap*	*opt**	*lp gap*	*opt gap*	*lp gap*	*opt gap*
`karate`	28	1.75	0.00	16.49	0.00	23	1.43	0.00	28.31	0.00
`chesapeake`	22	0.90	0.00	42.60	0.00					
`dolphins`	49	1.41	0.00	20.97	0.00	43	2.16	0.00	30.64	6.32
`lesmis`	74	0.80	0.00	3.85	0.00	72	0.69	0.00	6.41	0.00
`polbooks`	86	2.57	0.00	18.08	6.88	80	2.79	0.00	23.79	11.26
`adjnoun`	101	0.00	47.52	9.81	0.00	96		1.04		3.87
`football`	54		37.04		42.75	44		23.76		39.25

Table 4: LP relaxations and optimality gaps (DIMACS-10 instances).

[2] E. Balas and C. C. de Souza. The vertex separator problem: a polyhedral investigation. *Mathematical Programming*, 103(3):583–608, 2005.

[3] F. Barahona and D. Jensen. Plant location with minimum inventory. *Mathematical Programming*, 83(1):101–111, 1998.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[5] W. Ben-Ameur and M. Didi Biha. On the minimum cut separator problem. *Networks*, 59(1):30–36, 2012.

[6] W. Ben-Ameur, M.-A. Mohamed-Sidi, and J. Neto. The k-separator problem. *In Computing and Combinatorics*, pages 337–348, 2013.

[7] A. Berger, A. Grigoriev, and R. v. d. Zwaan. Complexity and approximability of the k-way vertex cut. *Networks*, 63(2):170–178, 2014.

[8] M. D. Biha and M.-J. Meurs. An exact algorithm for solving the vertex separator problem. *Journal of Global Optimization*, 49(3):425–434, 2011.

[9] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42(3):153 -159, 1992.

[10] S. Chopra and M. R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991.

[11] D. Cornaz, F. Furini, M. Lacroix, E. Malaguti, A. R. Mahjoub, and S. Martin. Mathematical formulations for the balanced vertex k-separator problem. *In Control, Decision and Information Technologies (CODIT14)*, pages 176–181, 2014.

[12] D. Cornaz, Y. Magnouche, A. R. Mahjoub, and S. Martin. The multi-terminal vertex separator problem: Polyhedral analysis and branch-and-cut. *Conference on Computers & Industrial Engineering (CIE45)*, pages 857–864, 2015.

[13] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. On multiway cut parameterized above lower bounds. *In Parameterized and Exact Computation*, pages 1–12, 2011.

[14] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakiss. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[15] C. de Souza and E. Balas. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 103(3):609–631, 2005.

[16] G. Desaulniers, J. Desrosiers, and M. Solomon, editors. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[17] J. Fukuyama. Np-completeness of the planar separator problems. *Journal of Graph Algorithms and Applications*, 10(2):317–328, 2006.

[18] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. *In Automata, Languages and Programming*, pages 487–498, 1994.

[19] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms*, 50(1):49–61, 2004.

[20] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35:921–940, 1988.

[21] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

[22] C. Phillips and T.J. Warnow. The asymmetric median tree: a new model for building consensus trees. *Combinatorial Pattern Matching*, pages 234–252, 1996.

[23] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer Ed, 2003.

[24] M. Thorup. Minimum k-way cuts via deterministic greedy tree packing. *In Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 159–166, 2008.