Exact and Heuristic Algorithms for the Maximum k-Cutset Problem

Patrick Healy, Pierre Laroche, Franc Marchetti, Sébastien Martin, Zsuzsanna Roka

Exact and Heuristic Algorithms for the Maximum k-Cutset Problem

Patrick Healy[a], Pierre Laroche[b], Franc Marchetti[b], Sébastien Martin[b], Zsuzsanna Róka[b]

[a] *University of Limerick, Limerick, Ireland*
[b] *Laboratoire de Conception, Optimisation et Modélisation des Systèmes, LCOMS EA 7306 Université de Lorraine, Metz 57000, FRANCE*

Abstract

Given a connected unit-weighted graph, we study the *Maximum k-Cutset Problem*, consisting in cutting a graph into k vertex-disjoint sub-graphs, each connected. We propose some exact and heuristic solutions to solve the problem. The first Integer Linear Program is based on a combination of a cut model and an assignment model. The two other Integer Linear Programs are based on the existence of spanning trees. The presented heuristics are clustering algorithms using local search. Experiments have been run on randomly generated instances and a specific set of instances. We observe that the running times are related to the tree-arboricity of graphs. The tree-arboricity is also used to ensure the optimality of solutions found by our heuristics.

Key words: Combinatorial optimization, Heuristics, Integer programming, Graph partitioning

1. Introduction

The problem of decomposing a graph (network) into smaller pieces, more commonly known as graph partitioning, has a long history dating back to the early 70s [19]. Many variants of the problem have been proposed since then and we discuss several below. While the problem of partitioning a graph into $k > 2$ subgraphs has received some attention, the additional constraint of maintaining connectedness is less common. Yet in the context of decomposition of geographical entities into sub-regions, this is a natural requirement. In this paper we consider *connected k-cut*, the problem of cutting (decomposing) a graph into k vertex-disjoint sub-graphs, each connected.

We investigate several algorithms for this problem, of which some are exact and some heuristic. With weights attached to the graph edges that represent the *undesirability* of assigning two adjacent vertices to the same partition, our algorithms can be viewed as finding a maximum k-cut in the graph – that is, remove as many "bad" (large) edges as possible. Alternatively, one may model the problem as finding a *minimum* k-cut where (larger) edge weights now represent the desirability or attraction of two adjacent vertices being co-assigned. Since both of these problems are $\mathcal{NP}$-hard [10, 12] in the general case, we model the problem as one of maximization in what follows.

Many problems in a geographical setting require decomposition of a network into *connected* entities and this work will be of relevance here. Examples include telecommunications or cable TV networks. The siting of municipal service facilities will also require that the service center be connected to each "client". However, our immediate application is motivated by electoral districting. The Irish parliament, the Dil, is composed of elected members from 40 electoral constituencies created from an agglomeration of 3409 atomic *electoral divisions* (EDs), of average area, 20km^2. The composition of electoral constituencies is reviewed and revised periodically on the basis of population census data. Figure 1 illustrates the distribution of EDs.

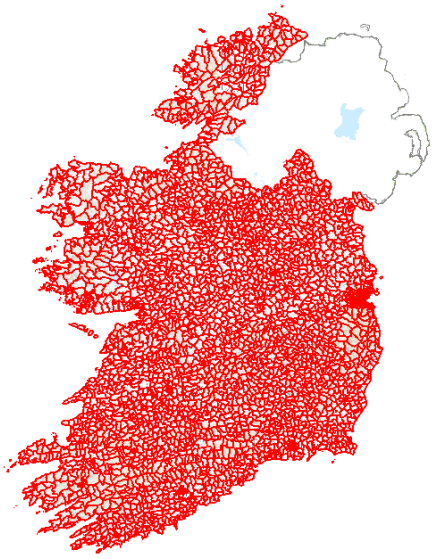

Figure 1: The 3409 Electoral Divisions of the Republic of Ireland.

Adjacency relations amongst EDs are naturally modeled as a graph and a 40-partition of the 3409-vertex graph is a basic requirement of any constituency map. Due to EDs being under different administrative regions it is more desirable for some adjacent EDs to be in the same electoral constituency than others. Conversely, if logically adjacent EDs are separated by a significant physical feature (large lake or river) then it may be desirable to assign such EDs to different constituencies. The desirability or undesirability of nominating logically adjacent EDs to be in the same constituency may be controlled by the assignment of weights to the edges. Crucially, it has been longstanding custom for electoral constituencies to be a *single* geographical region. The connected k-partitioning defined in the next section addresses exactly this problem.[1]

The paper is organised as follows. After defining the problem and briefly describing notation below, we discuss some simplifications of the problem. We describe then related work and the present status of the field in Section 3.

[1]While it may seem sufficient to solve the maximum spanning forest problem (or some variant [7]) we wish for the entire connected component to be a tight, cohesive unit, and not alone the spanning tree.

Section 4 proposes three Integer Linear Program models to solve the problem exactly. It will be seen that the models exhibit distinct behaviors according to graph density. A heuristic algorithm is then proposed and we discuss 3 variants. In Section 6 we present our experimental work and results. The paper is concluded in Section 7.

2. Basics and Problem Definition

Let $G = (V, E)$ be an undirected graph of size $m = |E|$ and order $n = |V|$, where V is the set of vertices and E is a set of edges which are unordered pairs of vertices of V. An edge $\{u, v\}$, where u and v are two vertices of V, is denoted uv (or vu). In this paper, we only consider simple graphs, *i.e.* graphs with neither multiple edges nor loops. For each $uv \in E$, the vertices u and v are said to be neighbours. In the following, the neighbourhood of a vertex $v \in V$ is denoted by $N_G(v) \subseteq V$. The degree of v is denoted by $\delta_G(v) = |N_G(v)|$. We extend the notion of neighbourhood to a subset of vertices $V' \subseteq V$ by $N_G(V') = \bigcup_{v \in V'} N_G(v)$. In a graph, a real number $w(e)$, called weight, can be associated with each edge $e \in E$. For a subset $E' \subseteq E$, $w(E') = \sum_{e \in E'} w(e)$ is the weight associated with E'.

Let $\mathcal{V} = \{V_1, V_2\}$ be a partition of V. A *cut* of G associated with $\mathcal{V}$, denoted by $C(\mathcal{V})$, is the set of all edges of G having one end vertex in V_1 and the other in V_2. Removing the edges of a cut from a connected graph G results in a graph having at least two components. If it has exactly two components, the cut is called a *cutset* or *bond*. Equivalently, a cutset is a cut of minimal size (none of its non-empty proper subsets is a cut). A well-known theorem states that a cut in a connected graph G is a cutset or union of edge-disjoint cutsets of G. Let T be a spanning tree of a connected graph G, and let b be a branch of T. The graph obtained by deleting b from T is denoted $T - b$. If V_1 and V_2 are the vertex-sets of the two components of $T - b$ then the cut $C(\mathcal{V})$, $\mathcal{V} = \{V_1, V_2\}$, is a cutset of G. This cutset is called the *fundamental cutset* of G with respect to the branch b of T. For each spanning tree T of G, there are $n - 1$ fundamental cutsets, one for each branch of T.

Let k be an integer such that $2 \leq k \leq n$ and let $\mathcal{V} = \{V_1, \ldots, V_k\}$ be a *k-partition* of V. The set $C(\mathcal{V})$ of all those edges of G having their end vertices in distinct subsets of $\mathcal{V}$ is called a *k-cut* of G. A k-cut of a connected graph is called a *k-cutset* if the removal of the edges in the k-cut results in a graph having exactly k components. The associated partition is then called *connected k-partition*. In the following, we will refer to K as the set $\{1, \ldots, k\}$.

Proposition 1. *Let C be a k-cutset in a connected graph G. Then, there exists at least one spanning tree of G such that C is the union of $k - 1$ fundamental cutsets associated with this spanning tree.*

Proof. Let $C(\mathcal{V})$ be a k-cutset of a connected graph G where $\mathcal{V} = \{V_1, \ldots, V_k\}$ is a *k-partition* of V. We denote by $\partial(V_i, V_j)$ the set of all edges of G having one end vertex in V_i and the other in V_j, $i < j$. Note that $\cup\{\partial(V_i, V_j)\ ;\ i < j, (i, j) \in K^2\} = C(\mathcal{V})$. Let $G_R = (V_R, E_R)$ be the contraction of G obtained as follows. For each $i \in K$, all edges of $G[V_i]$ are contracted in one vertex v_i, which is added

to V_R. Then, the edge v_iv_j is added to E_R if and only if $\partial(V_i, V_j)$ is not empty. Let T_R be a spanning tree of G_R. Let B be one of the $(k-1)$ fundamental cutsets of G_R associated with T_R. Then $\cup\{\partial(V_i, V_j)\ ;\ i < j, v_iv_j \in B\}$ is a cutset of G. This construction is illustrated by an example on Figure 2, where $B_1 = \{e_1, e_2, e_5, e_{10}\}$, $B_2 = \{e_1, e_8, e_{11}\}$ and $C(\mathcal{V}) = B_1 \cup B_2$.

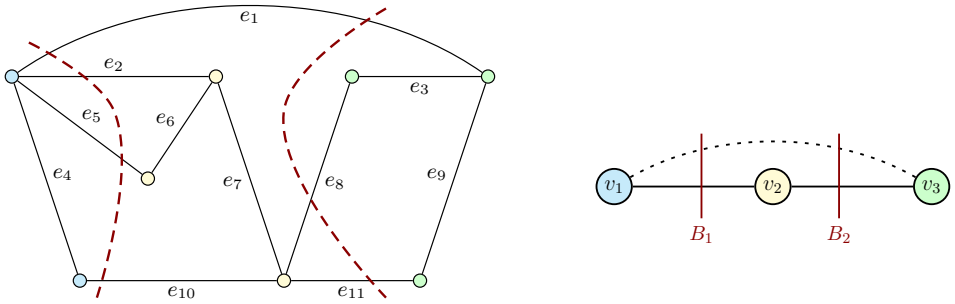

Figure 2: Construction of G_R and T_R.

□

The *maximum cut problem (max-cut)* consists in finding a cut of maximum weight. The *maximum cutset problem (max-cutset)* consists in finding a cutset of maximum weight. The *maximum k-cut problem (max k-cut)* consists in finding a k-cut of maximum weight. In this paper we consider the *maximum k-cutset problem* defined as follows.

Definition 1. The *maximum k-cutset problem (max k-cutset)* consists in finding a k-cutset of maximum weight or, equivalently, to find a connected k-partition $\mathcal{V}$ such that $w(C(\mathcal{V}))$ is maximum or $w(E) - w(C(\mathcal{V}))$ is minimum.

Let us now consider unweighted graphs $G = (V, E)$, *i.e.* $w(e) = 1$ for all $e \in E$. Then, the weight of a subset of edges is its cardinality. For the graph presented on Figure 2, the 3-cutset value is 6 and it is maximum (all subgraphs are trees).

Let $\mathcal{V} = \{V_1, \ldots, V_k\}$ be a k-partition of V where each induced subgraph $G[V_i]$, $i \in K$, is a tree. Such a partition is called a *k-tree-partition*. Then, $|C(\mathcal{V})| = |E| - \sum_{i \in K}(|V_i| - 1) = |E| - |V| + k$ and $C(\mathcal{V})$ is a maximum k-cutset. Furthermore, adding any edge in $E \setminus C(\mathcal{V})$ to $C(\mathcal{V})$ results in a $(k+1)$-tree-partition and a maximum $(k+1)$-cutset of cardinality $|E| - |V| + (k+1)$. In a related work [5], the authors define the tree-arboricity $\mathrm{ta}(G)$ of a graph G as the minimum integer value k such that G admits a k-tree-partition. This leads to the following obvious result.

Proposition 2. *Every simple connected graph $G = (V, E)$ admits a maximum k-cutset of cardinality $|E| - |V| + k$ if and only if $k \geq \mathrm{ta}(G)$.*

As mentioned earlier, our study was motivated by the Irish electoral constituency problem. As each constituency is expected to be topologically connected agglomeration of electoral divisions, the problem can be seen as a max

k-cutset for a connected graph. For disconnected graphs, the problem has to be solved by taking into account that each component has its own partition. Let us suppose that G is a union of c disjoint components, denoted by $G_1, \ldots, G_c$. Let C_j be a k_j-cutset of G_j, $1 \leq j \leq c$. If $\sum_{1 \leq j \leq c} k_j = k$ then $\cup_{1 \leq j \leq c} C_j$ is a k-cutset of G. To solve the max k-cutset for such graphs is not obvious. Indeed, finding the k_j values is a part of the problem which makes it more difficult to solve. This could be the subject of further studies.

3. State of the art

Several problems of immediate relevance to the k-cutset have been previously studied. We now describe this work.

Cut and max-cut. The max-cut is a classical problem in combinatorial optimization and is among the twenty one problems whose NP-hardness was established in [17]. Different approaches are presented below.
Within the field of approximation, an efficient algorithm has been proposed in [11], providing a quality of at least 0.878 times the optimal solution. Based on these results, Bertoni et al. [3] proposed a new algorithm which is very efficient in practice. It ensures similar quality in reduced computing times, although only providing a 0.39 quality ratio. In [20], the authors propose an efficient algorithm to get approximate global solutions of max-cut. Their algorithm is based on a discrete filled function algorithm embedded with continuous approximation.
Festa et al. [9] have presented a heuristic algorithm using local search to improve a feasible solution. In [16], the authors give a survey of different heuristics and compare the results on some well-known benchmarks. A tabu search algorithm is presented in [18].
Other researchers focused on exact methods. An exponential-time exact algorithm using polynomial space is described in [8]. A Branch-and-Bound approach is proposed in [22]. In [2], the authors propose some valid inequalities for the cut polytope and study their facial structure.
The max-cut problem has also been studied for planar graphs by Orlova and Dorfman in [21] and by Hadlock in [13]. They proposed a polynomial-time algorithm by translating the max-cut into a maximum weighted matching problem for which there exists a polynomial bounded algorithm.

k-cut and max k-cut. Some approaches by approximation of the max k-cut are presented in [24] and [10]. In [23], Sahni et al. have proposed a heuristic built-in algorithm to solve the max k-cut. This simple algorithm can be considered as the first known approximation algorithm guaranteeing good quality solutions.
Among exact methods, Ales *et al.* introduce a mixed integer linear programming formulation with edge variables, and representative variables and analyse the associated polytope [1].

max-cutset. Haglin and Venkatesan have shown in [14] that the max-cutset is NP-complete even for planar graphs in contrast to the max-cut which can be solved in polynomial time on planar graphs as we mentioned above. Several

formulations of the problem have been presented and used in Branch-and-Cut algorithms in [4].

The *max k-cutset* (Definition 1) is a generalization of the previous problem for k partitions with $k \geq 2$. To the best of our knowledge, no result has been published yet concerning this problem. Indeed, in the mentioned works studying k-cut, the connectivity of the induced subgraphs is not a required property. When this property is studied, the number k of subsets is limited to two. In a very recent preprint [15], the authors present some models using a different approach from ours to take into account the connectivity of the subgraphs. They study two alternative mixed-integer linear formulations, a cut model and a flow model. They also propose improvements, some of them making the models more efficient. As there are some similarities with our models, especially the cut model, we focus on the differences of the models as well as on the performance behaviour.

4. Integer Linear Program models

We propose three Integer Linear Programming (ILP) formulations to solve the max k-cutset for a simple connected graph $G = (V, U)$: our goal is to find a connected k-partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of V such that $w(C(\mathcal{V})) = |C(\mathcal{V})|$ is maximum. The first ILP is based on the combination of a cut model and an assignment model, and the two others are based on label structures inducing k spanning trees.

4.1. Cut / Assignment model (CAM)

In the ILP below, the variables z_v^i indicate if a vertex v is assigned to a subset V_i of the partition, for all $i \in K$. The variables y_{uv} indicate if uv is in the cutset, for all $uv \in E$.

$$z_v^i = \begin{cases} 1 & \text{if } v \in V_i, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall v \in V, \forall i \in K$$

$$y_{uv} = \begin{cases} 1 & \text{if } uv \in C(\mathcal{V}), \\ 0 & \text{otherwise,} \end{cases} \qquad \forall uv \in E.$$

For the sake of simplicity, we introduce variables $\bar{y}_{uv} = 1 - y_{uv}$, for all $uv \in E$, indicating if both ends of an edge uv are in the same partition. The ILP is defined as follows.

$$
\begin{aligned}
& \max \sum_{uv \in E} y_{uv} && && (1)\\
& \sum_{i \in K} z_u^i = 1, && \forall u \in V, && (2)\\
& \sum_{u \in V} z_u^i \geq 1, && \forall i \in K, && (3)\\
(P_{\mathrm{CAM}}) \quad & z_u^i - z_v^i \leq y_{uv}, && \forall uv \in E,\ \forall i \in K, && (4)\\
& z_v^i - z_u^i \leq y_{uv}, && \forall uv \in E,\ \forall i \in K, && (5)\\
& z_u^i + z_v^i \leq 1 + \bar{y}_{uv}, && \forall uv \in E,\ \forall i \in K, && (6)\\
& y_{uv} \in \{0,1\}, && \forall uv \in E, && (7)\\
& \bar{y}_{uv} \in \{0,1\}, && \forall uv \in E, && (8)\\
& z_u^i \in \{0,1\}, && \forall u \in V,\ \forall i \in K. && (9)
\end{aligned}
$$

The inequalities (2) ensure partitioning. Each subset of the partition must be non-empty, which is guaranteed by (3). Inequalities (4)–(6) are classically used to model the cutset associated to the partition.

Then, we only have to ensure the connectivity of each induced subgraph $G[V_i]$, $V_i \in \mathcal{V}$. Let $\mathcal{C}_{uv}$ be the set of all $u-v$ cuts in G. Recall that a $u-v$ cut partitions V into two subsets, the first one containing u and the second one containing v. Thus, we introduce the following inequalities:

$$
\sum_{u'v' \in C} \bar{y}_{u'v'} \geq z_u^i + z_v^i - 1, \forall i \in K, \forall u \in V, \forall v \in V, \forall C \in \mathcal{C}_{uv}. \qquad (10)
$$

If u and v are in the same set V_i, then at least one edge of each $u-v$ cut must belong to E_i, the set of edges of $G[V_i]$. Otherwise, there is no path connecting u and v in $G[V_i]$, and thus $G[V_i]$ is not connected.

There is an exponential number of inequalities of type (10) and thus, to solve this model, we must be able to separate infeasible solutions in an efficient way. Let $\bar{y}^* \in \{0,1\}^m$, $u, v \in V$ and $i \in K$. The separation problem associated with inequalities (10) entails finding a $u-v$ cut C such that

$$
\sum_{u'v' \in C} \bar{y}^*_{u'v'} < z_u^i + z_v^i - 1. \qquad (11)
$$

This problem can be solved in polynomial time using a min $s-t$ cut algorithm between each pair of vertices of G where each edge uv is weighted by $\bar{y}^*_{uv}$. If the inequality (11) is verified, then we add (10) to the model.

This model has been studied in [15] where the authors added various improvements to the basic model making it more efficient. We report on our results with several variants of this model in Section 6.

4.2. Label Models

These models are based on the fact that a graph admits a spanning tree if and only if it is connected. Finding a connected k-partition $\mathcal{V}$ is hence equivalent to find a spanning forest of G, composed of $n-k$ edges in k disjoint trees. Note that for a partition V_i there are an exponential number of spanning trees, each giving a different, yet symmetric, solution.

To eliminate some (but not all) of these symmetric solutions, we fix a representative for each partition V_i. Let $D = (V, A)$ be the directed graph (digraph) obtained from G, where each edge $uv \in E$ is replaced in A by two arcs (u, v) and (v, u). Let $F = \cup_{i \in K} T_i$ be a directed spanning forest of D. Then, the roots of the directed spanning trees $T_i = (V_i, A_i)$ of $D[V_i]$ can be chosen as representatives of the subsets V_i. Edges uv of $G[V_i]$ such that neither (u, v) nor (v, u) is an arc of T_i are called *friends*. The models proposed in this section use both G and the *support graph*, D.

We present two slightly different approaches to define representatives: in the first one they are chosen among the vertices of V, and the second one uses k dummy vertices.

4.2.1. Label model with representatives (LMR)

In this model, representatives are designated directly among the vertices of V. Vertices are indexed by natural numbers and we say that a vertex u is smaller than a vertex v, $u < v$, if the index of u is smaller than the index of v. The representative of a subset V_i (and all of its vertices) is the smallest vertex, chosen as the root of its spanning tree T_i. In this model, the following variables are used.

$$x_{uv} = \begin{cases} 1 & \text{if } (u,v) \text{ is an arc in } F, \\ 0 & \text{otherwise} \end{cases} \qquad \forall (u,v) \in A,$$

$$z_v^u = \begin{cases} 1 & \text{if } u \text{ is the representative of } v, \\ 0 & \text{otherwise} \end{cases} \qquad \forall u, v \in V,$$

$$y_{uv} = \begin{cases} 1 & \text{if } e \in C(\mathcal{V}), \\ 0 & \text{otherwise} \end{cases} \qquad \forall uv \in E,$$

$$l_u = \text{length of the path to } u \text{ from its representative} \qquad \forall u \in V.$$

The variable z_u^u indicates if u is the representative of one of the subsets of $\mathcal{V}$. We define the ILP as follows.

$$
\begin{aligned}
& \max \sum_{uv \in E} y_{uv} && && (12)\\
& \sum_{u \in V} z_u^u = k, && && (13)\\
& z_v^u \leq z_u^u, && \forall u \in V, \forall v \in V, v > u && (14)\\
& \sum_{u \leq v} z_v^u = 1, && \forall v \in V, && (15)\\
& z_u^w + z_v^w + y_{uv} \leq 2, && \forall uv \in E,\ w \in V, && (16)\\
& z_u^w - z_v^w \leq y_{uv}, && \forall uv \in E, v < u, \forall w \in V, && (17)\\
(P_{\text{LMR}}) \quad & z_v^w - z_u^w \leq y_{uv}, && \forall uv \in E, v < u, \forall w \in V, && (18)\\
& x_{uv} + x_{vu} + y_{uv} \leq 1, && \forall uv \in E, && (19)\\
& z_v^v + \sum_{(u,v) \in A} x_{uv} = 1, && \forall v \in V, && (20)\\
& l_u \leq M(1 - z_u^u), && \forall u \in V, && (21)\\
& -M(1 - x_{uv}) \leq l_v - l_u - 1, && \forall (u,v) \in A, && (22)\\
& l_v - l_u - 1 \leq M(1 - x_{uv}), && \forall (u,v) \in A, && (23)\\
& x_{uv} \in \{0,1\}, && \forall uv \in E, && (24)\\
& y_{uv} \in \{0,1\}, && \forall uv \in E, && (25)\\
& z_u^v \in \{0,1\}, && \forall u \in V, \forall v \in V, && (26)\\
& l_u \in \mathbb{N}, && \forall u \in V. && (27)
\end{aligned}
$$

Equality (13) ensures that there are exactly k representatives. Inequalities (14) express the fact that if a vertex v has u as a representative, then u is a representative of itself. Equality (15) ensures that each vertex has exactly one representative: itself or a smaller vertex. Equations (13)–(15) guarantee, then, that the smallest vertex of each subset is its unique representative. Inequalities (16) indicate that either both ends of an edge are in the same subset, or the edge is in the cut set. Inequalities (17) and (18) ensure that if two vertices u and v are in different classes, uv is a cut edge. Inequalities (19) guarantee that each edge uv is in one of the three situations: (i) uv is in the cut set, (ii) uv is a friend edge, (iii) (u, v) or (v, u) is an arc of the forest. Equality (20) ensures that the representatives are the roots of the spanning trees. Furthermore, a vertex not being a representative itself has exactly one direct predecessor in the forest. Inequalities (21)–(23) define some conditions on labels: each representative is labelled 0; the further a vertex is from the root, the greater its label is. They also ensure that each tree of the forest is connected and hence cycles are forbidden. The constant M can be bounded by n.

4.2.2. Label model with dummy vertices (LMD)

In this model, k dummy vertices are added to V. The representative of each subset of the partition is one of these dummy vertices.

Let $D = (V', A')$ be the (support) digraph defined by

- $V' = V \cup V_d$ where $V_d = \{v_1, ..., v_k\}$ is a set of dummy vertices;
- $A' = A \cup A_d$ where $A_d = \{(v_i, u) | u \in V, v_i \in V_d, i \in K\}$.

The spanning directed forest F' is defined in an analogous way as in the previous model. The root of each tree of F' is one of the dummy vertices. The variables are defined as follows:

$$x_{uv} = \begin{cases} 1 & \text{if } (u,v) \text{ is an arc in } F', \\ 0 & \text{otherwise} \end{cases} \qquad \forall (u,v) \in A'.$$

$$z_v^i = \begin{cases} 1 & \text{if } v \in V_i, \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in V, \forall i \in K.$$

$$y_e = \begin{cases} 1 & \text{if } uv \in C(\mathcal{V}), \\ 0 & \text{otherwise} \end{cases} \qquad \forall uv \in E.$$

$$l_u = \text{the pseudo-length of the path to } u \text{ in } F' \qquad \forall u \in V.$$

As in the previous model, the variables x_{uv} indicate if (u, v) is an arc of the forest and l_u is the pseudo-distance (one edge less than the geodesic distance) between u and the root (a dummy vertex) in a spanning tree. Variables z_v^i are defined in the same way as in the Cut/Assignment model, indicating if a vertex v is in a subset V_i of the partition. Each dummy vertex v_i can be considered as the root of a spanning tree. In all models, variables y_{uv} indicate if uv is a cut edge.

The ILP corresponding to this model is defined as follows:

$$
\begin{aligned}
& \max \sum_{uv \in E} y_{uv} && && (28)\\
& \sum_{i \in K} z_u^i = 1, && \forall u \in V, && (29)\\
& \sum_{u \in V} z_u^i \geq 1, && \forall i \in K, && (30)\\
& \sum_{(v,u) \in A'} x_{vu} = 1, && \forall u \in V, && (31)\\
& \sum_{u \in V} x_{v_i u} = 1, && \forall i \in K, && (32)\\
(P_{\text{LMD}}) \quad & z_u^i + z_v^i + y_{uv} \leq 2, && \forall uv \in E,\ i \in K, && (33)\\
& z_u^i - z_v^i \leq y_{uv}, && \forall uv \in E, v < u, \forall i \in K, && (34)\\
& z_v^i - z_u^i \leq y_{uv}, && \forall uv \in E, v < u, \forall i \in K, && (35)\\
& x_{uv} + x_{vu} + y_{uv} \leq 1, && \forall uv \in E, && (36)\\
& l_u \leq M(1 - x_{v_i u}), && \forall (v_i, u) \in A_d, \forall i \in K, && (37)\\
& -M(1 - x_{uv}) \leq l_v - l_u - 1, && \forall (u,v) \in A \setminus A_d, && (38)\\
& l_v - l_u - 1 \leq M(1 - x_{uv}), && \forall (u,v) \in A \setminus A_d, && (39)\\
& x_{uv} \in \{0,1\}, && \forall uv \in E, && (40)\\
& y_{uv} \in \{0,1\}, && \forall uv \in E, && (41)\\
& z_u^i \in \{0,1\}, && \forall u \in V, \forall i \in K, && (42)\\
& l_u \in \mathbb{N}, && \forall u \in V. && (43)
\end{aligned}
$$

Equalities (29) ensure that each vertex belongs to exactly one subset. Inequalities (30) ensure that none of the subsets is empty, (29)–(30) thus ensure the partition. Equalities (31) indicates that a true vertex has exactly one direct predecessor in the forest and (32) that each dummy vertex has exactly one direct successor in the forest. Inequalities (33) are analogous to (16): either both edges are in the same subset or the edge is in the cutset. As with (17) and (18), inequalities (34) and (35) ensure that if two vertices u and v are in different partitions, uv is a cut edge. Inequalities (36) are the same as (19) (3 possible situations for an edge). Inequalities (37)–(39) define the same labelling conditions as in the previous model, with value 0 for vertices succeeding a dummy vertex in a spanning tree.

5. Heuristics

We now propose some heuristics which are inspired by the following works.

In [23], Sahni and Gonzales proposed a $1/k$ approximate algorithm for the max-k-cut problem, which we will refer to below as the *S&G algorithm*. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ be a connected k-partition of V. The algorithm starts by assigning one vertex to each partition. Then, the $|V| - k$ remaining vertices

are examined one by one. A vertex v is added to the partition V_i for which the score function $score(v,i) = \sum_{u \in V_i \cap N(v)} w(uv)$ is minimal (*i.e.* the contribution to the objective function is optimal).

Kahruman *et al.* [16] proposed some variations of the S&G algorithm for the max-cut problem for complete weighted graphs. At each step, they select the non-assigned vertex with the best score. Three different score definitions were proposed and experimentally compared. They have also proposed an edge contraction heuristic. Starting with the initial graph, each step consists in contracting an edge into one vertex. The process ends when only 2 vertices remain, each one corresponding to a partition.

Festa *et al.* present another algorithm based on the S&G algorithm [9]. The authors define a set of not yet assigned vertices called the *restricted candidate list*. At each step, this set is rebuilt by using a similar score to [23] and [16]. Then, a vertex is chosen randomly from this list and is assigned to the corresponding partition. The k-partitions obtained by this greedy randomized algorithm are then improved by different local search phases. The shared idea of these local searches is to find a vertex such that its reassignment improves the objective function.

In this paper, we focus on the maximum k-cutset problem for unweighted connected graphs. The algorithms we propose guarantee the connectivity of the partitions which is not the case of the algorithms presented above. We use a randomized building phase of k connected partitions using score functions ensuring connectivity. In our algorithm, the local search phase is embedded in the building phase, in contrast to Festa's [9] algorithm. The local search phase is called only if the current partition is not a k-tree-partition, thus reducing the running time of the first iterations.

Formally, we define the algorithms as follows. Let $G = (V, E)$ be an undirected weighted graph and let $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ be a connected k-partition of V. We denote by $\delta_i(v) = |N(v) \cap V_i|$ the number of neighbours of v belonging to V_i and $\bar{\delta}_i(v) = \delta(v) - \delta_i(v)$ denotes the number of neighbours of v belonging to $V \setminus V_i$, for all i in K and v in V. If $\mathcal{V}$ is a connected k-partition of V, then $|E| = |C(\mathcal{V})| + |\bar{C}(\mathcal{V})|$, where $|C(\mathcal{V})| = \frac{1}{2}\sum_{i \in K}\sum_{v \in V_i} \bar{\delta}_i(v)$ and $|\bar{C}(\mathcal{V})| = \frac{1}{2}\sum_{i \in K}\sum_{v \in V_i} \delta_i(v)$ is the total size of the k induced connected subgraphs $G[V_i]$, $i \in K$. Let us also introduce the quantity $d(\mathcal{V}) = |E| - 2|C(\mathcal{V})| = 2|\bar{C}(\mathcal{V})| - |E|$.

In the following, we present the building and the local search phases of our heuristic maximizing $|C(\mathcal{V})|$. The heuristics maximizing $d(\mathcal{V})$ and minimizing $|\bar{C}(\mathcal{V})|$ are similar.

Building phase. At each iteration, let us consider V' as being the set of not yet assigned vertices and let $\bar{V}' = V \setminus V'$ be the set of assigned vertices. C denotes the current cut value of $G[\bar{V}']$. The initialization of the building phase consists in randomly choosing k vertices $\{v_1, \ldots, v_k\}$ and setting $V_i = \{v_i\}$ for all $i \in K$. We also use a classification indicator table of the vertices, denoted ρ. At the beginning of the algorithm, we set $\bar{V}' = \{v_1, \ldots, v_k\}$, $\rho[v_i] = i$ for all $i \in K$ and $C = 0$. At each step, we consider the vertices of $V'' = N(\bar{V}') \setminus \bar{V}'$. A vertex $v \in V''$ can be assigned to a partition V_i, if $\bar{\delta}_i(v)$ is maximum on K and satisfies the connectivity constraint $\delta_i(v) \geq 1$. Notice that a vertex can satisfy

these conditions for different partitions V_i. Let (v', i') be randomly chosen in the set $\{(v, i)\}$ of all possible assignments. Then, v' is assigned to subset $V_{i'}$ ($\rho[v'] = i'$) and the cut value is set to $C + \bar{\delta}_{i'}(v')$ (see Heuristic – part 1).

Heuristic - part 1 Main loop of the building phase

while $|V'| > 0$ **do**
  $V'' \leftarrow N(\bar{V}') \setminus \bar{V}'$
  $\bar{\delta}_{\max} \leftarrow \max\{\bar{\delta}_i(v) \mid (v, i) \in V'' \times K;\ \delta_i(v) \geq 1\}$;
  $S \leftarrow \{(v, i) \in V'' \times K \mid \bar{\delta}_i(v) = \bar{\delta}_{\max};\ \delta_i(v) \geq 1\}$;
  randomly choose (v, i) in S
  $V_i \leftarrow V_i \cup \{v\}$; $\rho[v] \leftarrow i$; $V' \leftarrow V' \setminus \{v\}$; $\bar{V}' \leftarrow \bar{V}' \cup \{v\}$
  $C \leftarrow C + \bar{\delta}_{\max}$
  if $|E| - C > |\bar{V}'| - K$ **then**
    local_search$(\rho, \mathcal{V}, C)$
  end if
end while

Local search phase. We first construct a list of vertices $L = (v_1, \ldots, v_n)$ where the vertices are sorted in an increasing order of their contribution: $\bar{\delta}_{\rho[v_i]}(v_i) \leq \bar{\delta}_{\rho[v_j]}(v_j)$, for all $1 \leq i < j \leq n$. We intend to first reassign vertices having the worst contribution. For each $v \in L$, v is reassigned to another partition whenever the objective function C is improved or unchanged after the reassignment. This loop is repeated until no reassignment has been done or no improvement occurred during the previous $n_{\max}$ iterations. The parameter $n_{\max}$ is used to avoid infinite back-and-forth between equivalently scored partitions ($n_{\max} = \log n$ in our algorithm). Notice that if a vertex is an articulation vertex, it cannot be reassigned to another partition. The local search phase is presented below (see Heuristic – part 2).

Heuristic - part 2 Local search phase

repeat
  change $\leftarrow$ false
  improved $\leftarrow$ false
  for $v \in L$ **do**
    if v is a not an articulation vertex of $G[V_{\rho[v]}]$ **then**
      $\bar{\delta}_{\max} \leftarrow \max\{\bar{\delta}_i(v) \mid i \in K;\ \delta_i(v) \geq 1\}$;
      $S \leftarrow \{i \in K \mid \bar{\delta}_i(v) = \bar{\delta}_{\max};\ \delta_i(v) \geq 1\}$;
      if $(\bar{\delta}_{\rho[v]}(v) \leq \bar{\delta}_{\max})$ **then**
        randomly choose i in S
        if $i \neq \rho(v)$ **then**
          $C \leftarrow C + \bar{\delta}_{\max} - \bar{\delta}_{\rho[v]}$
          $V_i \leftarrow V_i \cup \{v\}$; $V_{\rho[v]} \leftarrow V_{\rho[v]} \setminus \{v\}$; $\rho[v] \leftarrow i$
          change $\leftarrow$ true
          improved $\leftarrow \bar{\delta}_{\rho[v]}(v) < \bar{\delta}_{\max}$
        end if
      end if
    end if
  end for
until $\neg$*change* or $\neg$*improved* for the last $n_{\max}$ iterations

6. Experimental results

The experiments were run using Java 8 with CPLEX 12.63 on a Linux computer, with Intel Core I5-370 3.4GHz quad core processors and 8GB memory. CPLEX's working memory and tree sizes were not limited; a time limit of 3600 seconds was imposed on each run; all runs were single-threaded, that is, they were run without parallelism.

We present some comparisons between the models presented in [15] and ours. The computations of [15] were run on a Linux cluster with Intel Xeon E5 3.5GHz quad core processors and 32GB memory and the authors used SCIP 5.0.1 with the LP solver CPLEX 12.7.1 to solve the instances. Thus, our environment being less performant, our results cannot be compared to theirs in an exact way. Nevertheless, we can assume that using a more efficient environment the results could only be better.

Our experiments consider the exact and heuristic models introduced in the previous sections. Two main types of data were used as input.

- To examine the behaviour of the models as a function of parameters such as vertex count, edge density of the input graph and partition counts, we generated pseudo-random graphs of varying sizes.

- We also compare our results to the results of the authors of [15]. As their code is not available currently, we cannot make an exhaustive comparison between our models and theirs. Nevertheless, we ran some tests on the same set of unweighted graph instances from the Color02 symposium [6] and could compare the number of completions and the running times.

Finally, as a more realistic input, we used the 3409-vertex Electoral Divisions graph described in the Introduction. As it is beyond the abilities of the exact methods to solve this graph, we only used it as an input for heuristics methods.

6.1. Instances

Our pseudo-random instances are generated as follows. For each couple $(|V|, d)$, $|V| \in \{15, 20, 25\}$ and density $d \in \{0.25, 0.50, 0.75\}$, 10 instances were generated. In order to ensure connectivity, each instance was built by generating an initial random tree. Then, random edges are added until the appropriate density is reached. All edges weights are unitary. Therefore, 90 input graphs were generated in total. On each of these graphs a maximum k-cutset problem was solved for $k \in \{2, 3, \ldots, 8\}$. This yielded 630 problem instances for each of the models.

The `Color02` instances correspond to 51 of the smallest instances from the Color02 symposium on coloring problems [6]. The number of vertices varies from 11 to 282 except for one instance, which has 2368 vertices. The density varies from 3.4% to 89.6%.

6.2. Exact methods

Recall that our ILP algorithms are based on three models: a cut/assignment model (CAM), a label model where the representative vertices are vertices of the graph (LMR) and a label model where dummy vertices determine the partitioning (LMD). As mentioned earlier, the CAM proposed in [15] is more comprehensive than ours, so we focus on LMR and LMD.

Each model was run on each of the 90 graph instances with k specified from 2 to 8. Table 1 compares the two label models proposed in this paper. We observe that LMD is more efficient than LMR: more instances are solved to optimality.

630 runs	LMD	LMR
#opt	616	608
Total time	92474	97428
Mean time	146.8	154.65

Table 1: Aggregate completions, total and mean running times (in seconds) for each model, on all randomly generated instances ($k \in \{2, \ldots, 8\}, |V| \in \{15, 20, 25\}$ and graph density $\in \{25, 50, 75\}$).

In Tables 2, 3 and 4, we give some more details concerning the results on randomly generated instances for the LMD and LMR, for different values of $|V|$: 15, 20 and 25. We present as *#opt* the number of instances solved to optimality within the given time (one hour). The running time *G-mean time* (also used in [15]) is given in shifted geometric mean $\prod_{i=1}^{n}(t_i + s)^{1/n} - s$, where $n = 51$ and $s = 10$. This way to compute the running times decreases the outliers' influence. *Mean time* is the average running time by instance based on the total running time. We can observe that all instances are solved to optimality for $|V| = 15$ for both models. For $|V| = 20$, only one instance is not solved by LMD. For $|V| = 25$, none of the models can solve all of the instances. We observe that

LMR is more efficient than LMD for smaller number of vertices. Indeed, for greater values, the number of variables in LMR increases more, inducing bigger running times, especially for $k < \text{ta}(G)$. For both models, it decreases in a significant way when the tree-arboricity is reached.

LMD						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	1.16	1.36
3	10	10	10	30	2.13	2.23
4	10	10	10	30	1.25	1.90
5	10	10	10	30	0.61	0.63
6	10	10	10	30	0.26	0.27
7	10	10	10	30	0.15	0.16
8	10	10	10	30	0.13	0.13
Sum	70	70	70	210	5.69	0.95

LMR						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	0.34	0.34
3	10	10	10	30	0.71	0.72
4	10	10	10	30	0.39	0.49
5	10	10	10	30	0.09	0.09
6	10	10	10	30	0.05	0.05
7	10	10	10	30	0.04	0.04
8	10	10	10	30	0.03	0.03
Sum	70	70	70	210	1.65	0.25

Table 2: Aggregate completions, G-mean and mean running times (in seconds) for LMD and LMR, $|V| = 15$, on randomly generated instances.

LMD						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	5.73	47.15
3	10	10	**9**	**29**	45.96	155.14
4	10	10	10	30	108.82	151.08
5	10	10	10	30	7.32	7.85
6	10	10	10	30	2.94	3.42
7	10	10	10	30	1.67	1.79
8	10	10	10	30	1.03	1.11
Sum	70	70	69	209	173.47	52.51

LMR						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	2.09	2.10
3	10	10	10	30	40.59	44.80
4	10	10	10	30	27.18	36.74
5	10	10	10	30	0.50	0.51
6	10	10	10	30	0.32	0.32
7	10	10	10	30	0.21	0.21
8	10	10	10	30	0.15	0.15
Sum	70	70	70	210	71.04	12.12

Table 3: Aggregate completions, G-mean and mean running times (in seconds) for LMD and LMR, $|V| = 20$, on randomly generated instances.

LMD						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	23.43	30.85
3	10	10	10	30	176.82	619.71
4	10	10	**0**	**20**	112.86	1211.63
5	10	10	**7**	**27**	74.84	192.16
6	10	10	10	30	16.91	28.69
7	10	10	10	30	9.59	17.46
8	10	10	10	30	5.94	7.70
Sum	70	70	57	197	420.39	301.17

LMR						
k	Density			#opt	G-Mean time	Mean time
	25	50	75			
2	10	10	10	30	27.33	60.01
3	10	10	**0**	**20**	227.63	1373.33
4	10	10	**0**	**20**	80.05	1207.48
5	10	10	**8**	**28**	41.11	519.05
6	10	10	10	30	1.96	2.27
7	10	10	10	30	0.98	1.01
8	10	10	10	30	0.96	1.02
Sum	70	70	48	188	380.02	504.92

Table 4: Aggregate completions, G-mean and mean running times (in seconds) for LMD and LMR, $|V| = 25$, on randomly generated instances.

Figure 3 allows to more precisely analyze the running times, showing the influence of the density of the graphs. We observe that there is a spike for each instance G. This spike corresponds to $k \in \{\text{ta}(G), \text{ta}(G) - 1, \text{ta}(G) - 2\}$, depending on the model and the instance. When $k \geq \text{ta}(G)$, any optimal partition is a k-tree-partition. Then, as the number of k-partitions increases further with k, finding a k-partition becomes easier still.

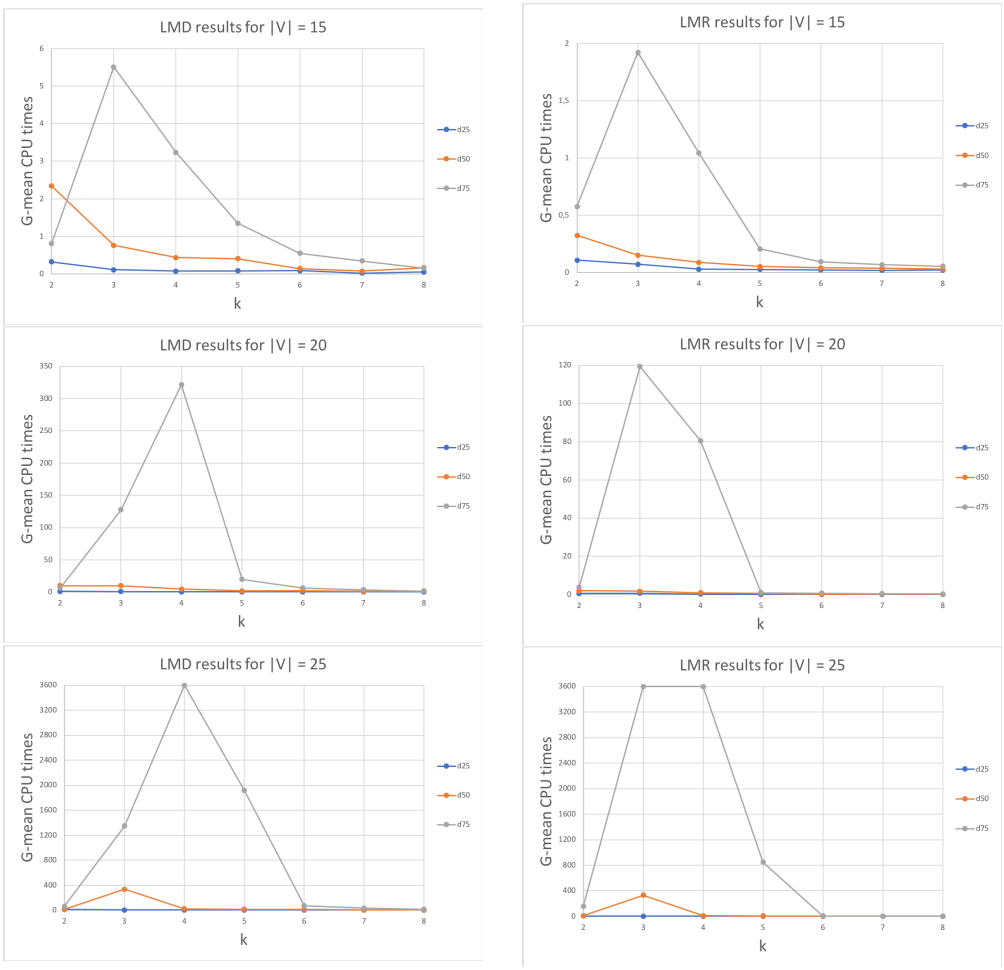

Figure 3: G-mean CPU times (in secs) for LMD and LMR, $|V| \in \{15, 20, 25\}$ and graph density $\in \{25, 50, 75\}$.

In Tables 5 and 6, we compare the best results obtained by the authors of [15] to ours obtained by LMD and LMR. Tests were run on the `Color02` instances. Columns *Best of [15]* of Table 5 present the best results obtained by the authors of [15], whatever the formulation and improvement, columns *Flow form. no impr [15]* present their results using a flow formulation (applying no improvement), and columns *Cut form. no impr [15]* present their results with a cut formulation (also with no improvement). We present their results with no improvements to provide a fair comparison between raw formulations.

51 instances	Best of [15]			Flow form. no impr [15]			Cut form. no impr [15]		
	k=2	k=5	k=10	k=2	k=5	k=10	k=2	k=5	k=10
#opt	20	17	21	13	1	1	15	1	1
G-mean time	796.4	1298.6	698.4	1323.0	3251.0	3206.3	1039.4	3217.3	3207.0

Table 5: Aggregate completions and mean running times (in seconds) for models of [15], on the `Color02` instances.

51 instances	LMD			LMR		
	k=2	k=5	k=10	k=2	k=5	k=10
#opt	8	15	18	7	12	18
G-mean time	1818.3	1287.1	1044.6	1992.4	1305.2	957.7

Table 6: Aggregate completions and mean running times (in seconds) for models LMD and LMR, on the `Color02` instances.

We observe that, comparing the raw formulations, LMD and LMR solve more instances than the flow or cut formulations of [15] when $k \neq 2$. We can also notice that the efficiency of the formulations [15] decreases with increasing k, whereas our formulations show an opposite behaviour. Using all improvements, their results are better than ours but the difference is not that significant when $k \neq 2$. Our models are less efficient for $k = 2$ due to the large search space.

6.3. Heuristics

In this section, we present the results obtained by heuristics. For randomly genarated instances, see Tables 7, 8 and 9. On each instance, $100 \times |V|$ tests were run. The *percentage of success* represents the percentage of runs over all runs for which an optimal solution is found. *Running times* correspond to the total time needed to launch $100 \times |V|$ tests for all 10 instances of the benchmark. We observe that, as for the ILPs, the tree-arboricity has an influence on the results. Indeed, we can see that the percentage of success decreases until the tree-arboricity is reached. After that threshold, the percentage of success significantly increases. The ratio between the number of optimal solutions and the number of feasible solutions is larger when the tree-arboricity is reached. Hence, the probability for one run to find an optimal solution is bigger.

For the Color02 instances, see Table 10. For $k = 2$, the optimal solutions found by LMR are also found by heuristics. For the other values of k, #opt is the number of optimal solutions corresponding to a k-tree-partition. Again, the tree-arboricity has an important impact on the results. The corresponding values are in bold. When it is reached, the problem becomes easier to solve.

k	Percentage of success				Running times				
	Density			Average	Density			Total	Average
	25	50	75	over 4500 runs	25	50	75		per instance
2	**55.13**	**31.32**	51.47	45.97	0.78	1.39	2.08	4.25	0.14
3	79.91	76.50	**17.48**	57.97	0.55	0.64	1.39	2.58	0.08
4	89.05	98.79	72.20	86.68	0.45	0.45	0.69	1.60	0.05
5	93.21	99.89	99.69	97.59	0.40	0.41	0.44	1.25	0.04
6	95.49	99.99	99.99	98.49	0.38	0.39	0.40	1.16	0.04
7	97.17	100	100	99.05	0.35	0.36	0.39	1.10	0.04
8	98.34	100	100	99.45	0.35	0.37	0.37	1.10	0.04

Table 7: Percentage of success and running times (in seconds) for heuristics, $|V| = 15$, on randomly generated instances, over 1500 runs on each instance.

k	Percentage of success				Running times				
	Density			Average	Density			Total	Average
	25	50	75	over 6000 runs	25	50	75		per instance
2	**19.35**	33.25	43.31	31.97	2.23	4.71	6.89	13.83	0.46
3	84.68	**8.97**	29.84	41.17	1.07	2.45	5.02	8.54	0.28
4	95.24	92.26	**2.86**	63.45	0.80	0.99	3.18	4.96	0.16
5	98.32	99.53	65.14	87.67	0.69	0.74	1.39	2.82	0.09
6	99.16	99.97	99.04	99.39	0.64	0.67	0.78	2.09	0.07
7	99.61	99.98	100	99.86	0.58	0.62	0.67	1.88	0.06
8	99.73	99.99	100	99.91	0.57	0.62	0.63	1.82	0.06

Table 8: Percentage of success and running times (in seconds) for heuristics, $|V| = 20$, on randomly generated instances, over 2000 runs on each instance.

k	Percentage of success				Running times				
	Density			Average	Density			Total	Average
	25	50	75	over 7500 runs	25	50	75		per instance
2	**10.28**	40.1	31.24	27.20	6.99	14.80	20.77	42.56	1.42
3	84.66	**4.78**	28.41	39.28	2.34	9.80	16.84	28.98	0.97
4	97.16	49.69	**8.29**	51.72	1.45	4.25	12.21	17.92	0.60
5	98.86	98.89	**0.57**	66.11	1.16	1.44	7.60	10.21	0.34
6	99.39	99.98	65.448	88.27	1.02	1.09	3.00	5.10	0.17
7	99.73	99.99	99.512	99.74	0.92	0.98	1.31	3.21	0.11
8	99.84	99.99	100	99.95	0.86	0.93	1.04	2.83	0.09

Table 9: Percentage of success and running times (in seconds) for heuristics, $|V| = 25$, on randomly generated instances, over 2500 runs on each instance.

51 instances	Best of [15]			Heuristics			
	k=2	k=5	k=10	k=2	k=5	k=10	k=15
# opt	20	17	21	7	22	37	43
G-mean time	796.4	1298.6	698.4	168.7	59.4	17.4	8.4

Table 10: Aggregate completions and mean running times (in seconds) for models of [15] and heuristics, on the `Color02` instances (heuristics: 1500 runs for each instance).

We observe that heuristics provide better results than the best ones of [15], except for $k = 2$. The tree-arboricity property proves the optimality of the solutions, the running times are very small and decrease with k increasing.

Finally, we ran some tests on the graph representing the electoral divisions of the Republic of Ireland (3409 vertices and 9638 edges). Ten runs have been launched, taking 7606 seconds. The best objective value obtained for the 40-cutset is 5992. As the found solution is not a 40-tree partition, we cannot ensure its optimality. However, a solution is found, which is impossible with any of the exact methods studied in this paper.

7. Conclusion

In this paper, we proposed exact and heuristic approaches to solve the Maximum k-Cutset Problem. Concerning the exact models, LMR is more efficient than LMD for instances with a small or medium number of vertices since symmetrical solutions are avoided. For higher numbers of vertices, LMD performs better than LMR thanks to the smaller number of variables. The proposed

heuristics provide good results: the running times are very small, and the solutions are optimal (guaranteed by the tree-arboricity or confirmed by an exact method ran using four threads with no time limit).

In future work, we aim to extend the study on weighted graphs. These weights can be associated with edges or vertices, to represent distances between cities or their demography, respectively.

References

[1] Ales, Z., Knippel, A., & Pauchet, A. (2016). Polyhedral combinatorics of the k-partitioning problem with representative variables. *Discrete Applied Mathematics*, *211*, 1–14.

[2] Barahona, F., & Mahjoub, A. R. (1986). On the cut polytope. *Mathematical Programming*, *36*, 157–173.

[3] Bertoni, A., Campadelli, P., & Grossi, G. (2001). An approximation algorithm for the maximum cut problem and its experimental analysis. *Discrete Applied Mathematics*, *110*, 3–12.

[4] Borne, S., Fouilhoux, P., R., G., Lacroix, M., & Pesneau, P. (2014). Branch-and-cut algorithm for the connected-cut problem. In *ROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision Bordeaux*. France.

[5] Chang, G. J., Chen, C., & Chen, Y. (2004). Vertex and tree arboricities of graphs. *Journal of Combinatorial Optimization*, *8*, 295–306.

[6] Color02 (2002). Computational symposium: Graph coloring and its generalizations. Avalaible on `http://http://mat.gsia.cmu.edu/COLOR02`.

[7] Cordone, R., & Maffioli, F. (2004). On the complexity of graph tree partition problems. *Discrete Applied Mathematics*, *134*, 51–65.

[8] Della Croce, F., Kaminski, M. J., & Paschos, V. T. (2007). An exact algorithm for max-cut in sparse graphs. *Operations Research Letters*, *35*, 403–408.

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

[10] Frieze, A., & Jerrum, M. (1995). Improved approximation algorithms for max k-cut and max bisection. (pp. 1–13). Balas E., Clausen J. (eds) Integer Programming and Combinatorial Optimization. IPCO 1995. LNCS, Springer, Berlin, Heidelberg volume 920.

[11] Goemans, M. X., & Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefine programming. *Journal of the Association for Computing Machinery*, *42*, 1115–1145.

[12] Goldschmidt, O., & Hochbaum, D. S. (1988). Polynomial algorithm for the k-cut problem. (pp. 444–451). 29th Ann. Symp. Foundation of Computer Science.

[13] Hadlock, F. (1975). Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, *4*, 221–225.

[14] Haglin, D. J., & Venkatesan, S. M. (1991). Approximation and intractability results for the maximum cut problem and its variants. *IEEE Transactions on Computers*, *40*, 110–113.

[15] Hojny, C., Joormann, I., Lüthen, H., & Schmidt, M. (2018). Mixed-integer programming techniques for the connected max-k-cut problem. Unpublished. Preprint avalaible on `http://www.optimization-online.org`.

[16] Kahruman, S., Kolotoglu, E., Butenko, S., & Hicks, I. V. (2007). On greedy construction heuristics for the max-cut problem. *Int. J. of Computational Science and Engineering*, *3*, 211–218.

[17] Karp, R. M. (1972). Reducibility among combinatorial problems. *Miller R.E., Thatcher J.W., Bohlinger J.D. (eds) Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA*, *1*, 85–103.

[18] Kochenberger, G. A., Hao, J.-K., Zhipeng, L. H., & Glover, F. (2013). Solving large scale max cut problems via tabu search. *J Heuristics*, *19*, 565–571.

[19] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*, 498–516.

[20] Ling, A. F., Xu, C. X., & Xu, F. M. (2009). A discrete filled function algorithm embedded with continuous approximation for solving max-cut problems. *European Journal of Operational Research*, *197*, 519–531.

[21] Orlova, G. I., & Dorfman, Y. G. (1972). Finding the maximum cut in a graph. *Engineering Cybernetics*, *10*, 502–506.

[22] Rendl, F., Rinaldi, G., & Wiegele, A. (2008). Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, *121*, 307–335.

[23] Sahni, S., & Gonzales, T. (1976). P-complete approximation problems. *Journal of the ACM*, *23*, 555–565.

[24] Zhu, W., & Guo, C. (2011). A local search approximation algorithm for max-k-cut of graph and hypergraph. (pp. 236–240). 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, Tianjin, China.