Exact and Heuristic Solutions to the Connected k-Partitioning Problem

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

Exact and Heuristic Solutions to the Connected k-Partitioning Problem

Patrick Healy[1], and Pierre Laroche, Franc Marchetti, Sébastien Martin and Zsuzsanna Róka[2]

***Abstract*— We study the problem of partitioning a graph into k connected components, which may also be referred to as the maximum k-cutset problem. Firstly, we present an exact algorithm and a variant, both implemented as integer linear programming (ILP) models. We then present a heuristic approach that will be seen to be extremely competitive with the exact algorithm for the ranges of graph under consideration.**

I. INTRODUCTION

As we describe below, the graph partitioning problem has a long history of study. In its most frequent presentation the problem is to decompose a given graph into two subgraphs by removing edges so that the weight of the edges removed is optimal. In this work we generalize the problem to require the graph to be partitioned into $k \geq 2$ subgraphs and, further, that each subgraph be connected. We seek to *maximize* the sum of weights of the removed edges although a minimizing variant is possible, too.

The problem we describe can be seen to impose connectivity requirements naturally such as those that would arise in graph problems representing geographical entities. In fact, the problem was initially motivated by an electoral districting problem where the goal was to agglomerate electoral units into parliamentary divisions so that naturally occurring geographical features were respected.

In the section below we introduce the problem formally and describe terminology. We then proceed, in Section III, to describe work done on simpler variants and to give performance guarantees when known. Sections IV and V describe, respectively, our exact ILP model and our heuristic algorithm. The outcomes of these two approaches on our generated experimental input cases is described and discussed in Section VI. Section VII concludes the paper where we discuss the algorithms' performance further.

II. 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 self-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 $N_G(V') = \bigcup_{v \in V'} N_G(v)$, *i.e.* the set of all neighbours of vertices in V'. In a graph, a positive 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 [7] 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 $\{1, \ldots, k\}$.

The *max k-cut problem* (*max-cut* when $k = 2$) consists in finding a k-cut of maximum weight. In this paper we consider the *max k-cutset problem* (max-cutset when $k = 2$), 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 in Figure 1, the 3-cutset value is 6 (the dotted edges) 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

[1]University of Limerick, Limerick, Ireland patrick.healy@ul.ie

[2]Laboratoire de Conception, Optimisation et Modélisation des Systèmes, LCOMS EA 7306 Université de Lorraine, Metz 57000, FRANCE firstname.lastname@univ-lorraine.fr

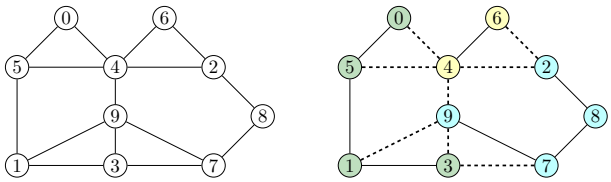

Fig. 1. A graph and a maximal 3-cutset.

maximum $(k+1)$-cutset of cardinality $|E| - |V| + (k+1)$. In a related work [9], 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. Hence, every simple connected graph $G = (V, E)$ admits a maximum k-cutset of cardinality $|E| - |V| + k$ if and only if $k \geq \mathrm{ta}(G)$.

III. STATE OF THE ART

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

a) 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 [16]. Different approaches are presented below.

Within the field of approximation, an efficient algorithm has been proposed in [13], providing a quality of at least 0.878 times the optimal solution. Based on these results, Bertoni et al. [6] 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 [19], 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. [11] have presented a heuristic algorithm using local search to improve a feasible solution. In [17], 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 [10]. A Branch-and-Bound approach is proposed in [21]. 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 [20] and by Hadlock in [14]. 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.

b) k-cut and max k-cut: Some approaches by approximation of the max k-cut are presented in [23] and [12]. In [22], 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]. More recently, Barreiro-Gomez *et al.* studied the partitioning problem [3,4] for large-scale systems. They use a multi-objective criterion, one being a min k-cut and another, a balanced partition.

c) max-cutset: Haglin and Venkatesan have shown in [15] 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 [8].

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.

IV. INTEGER LINEAR PROGRAM MODEL

We propose an Integer Linear Programming (ILP) formulation 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. It is based on a label structure inducing k spanning trees.

We use 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 is 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 proposed model, called LMR (for *Label Model with Representatives*), uses both G and the *support graph*, D. The 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.

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

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

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

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

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.

$$\max \sum_{uv \in E} w_{uv} y_{uv} \tag{1}$$

$$\sum_{u \in V} z_u^u = k, \tag{2}$$

$$z_v^u \leq z_u^u, \qquad \forall u \in V, \forall v \in V, v > u \tag{3}$$

$$\sum_{u \leq v} z_v^u = 1, \qquad \forall v \in V, \tag{4}$$

$$z_u^w + z_v^w + y_{uv} \leq 2, \qquad \forall uv \in E,\ w \in V, \tag{5}$$

$$z_u^w - z_v^w \leq y_{uv}, \qquad \forall uv \in E, v < u, \forall w \in V, \tag{6}$$

$$z_v^w - z_u^w \leq y_{uv}, \qquad \forall uv \in E, v < u, \forall w \in V, \tag{7}$$

$$x_{uv} + x_{vu} + y_{uv} \leq 1, \qquad \forall uv \in E, \tag{8}$$

$$z_v^v + \sum_{(u,v) \in A} x_{uv} = 1, \qquad \forall v \in V, \tag{9}$$

$$l_u \leq M(1 - z_u^u), \qquad \forall u \in V, \tag{10}$$

$$-M(1 - x_{uv}) \leq l_v - l_u - 1, \qquad \forall (u,v) \in A, \tag{11}$$

$$l_v - l_u - 1 \leq M(1 - x_{uv}), \qquad \forall (u,v) \in A, \tag{12}$$

$$x_{uv} \in \{0,1\}, \qquad \forall uv \in E, \tag{13}$$

$$y_{uv} \in \{0,1\}, \qquad \forall uv \in E, \tag{14}$$

$$z_u^v \in \{0,1\}, \qquad \forall u \in V, \forall v \in V, \tag{15}$$

$$l_u \in \mathbb{N}, \qquad \forall u \in V. \tag{16}$$

Equality (2) ensures that there are exactly k representatives. Inequalities (3) express the fact that if a vertex v has u as a representative, then u is a representative of itself. Equality (4) ensures that each vertex has exactly one representative: itself or a smaller vertex. Equations (2)–(4) guarantee, then, that the smallest vertex of each subset is its unique representative. Inequalities (5) indicate that either both ends of an edge are in the same subset, or the edge is in the cut set. Inequalities (6) and (7) ensure that if two vertices u and v are in different classes, uv is a cut edge. Inequalities (8) guarantee that each edge uv is in one of three states: (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 (9) 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 (10)–(12) 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.

Among cycles, some of them are triangles which can be avoided by the following tree widening inequalities (TWI for short), hence helping to reduce symmetrical solutions. They are defined for all $uv \in E$ and for all $w \in N(u) \cap N(v)$:

$$x_{uv} + x_{vw} \leq 1, \tag{17}$$

$$x_{wv} + x_{vu} \leq 1, \tag{18}$$

$$x_{uw} + x_{wv} \leq 1, \tag{19}$$

$$x_{vw} + x_{wu} \leq 1, \tag{20}$$

$$x_{vu} + x_{uw} \leq 1, \tag{21}$$

$$x_{wu} + x_{uv} \leq 1. \tag{22}$$

These constraints also ensure that the spanning tree is wide and shallow.

V. HEURISTICS

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

In [22], 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.* [17] 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 [11]. 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 [22] and [17]. 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 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 connected k-partitions using score functions ensuring connectivity. In our algorithm, the local search phase is embedded in the building phase, in contrast to Festa's [11] algorithm.

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 the score function $\Delta_i(u) = \sum_{v \in V_i \cap N(u)} w(uv)$ as the total weight of the edges of $G[V_i]$ being incident to u. Let $\bar{\Delta}_i(u) = \sum_{v \in N(u) \setminus V_i} w(uv)$ the total weight of the remaining edges being incident to u. Then, $\Delta(u) = \Delta_i(u) + \bar{\Delta}_i(u)$ is the total weight of the edges being incident to u, for all i in K and u in V.

In the following, we present the building and the local search phases of our heuristic maximizing the weight of the k-cutset defined by $w(C(\mathcal{V})) = \sum_{i \in K} \sum_{u \in V_i} \bar{\Delta}_i(u)$. The heuristic minimizing $\sum_{i \in K} \sum_{u \in V_i} \Delta_i(u)$ would be similar.

a) 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. w_C denotes the current cut weight 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 $w_C = 0$. 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 $\delta_i(v) \geq 1$ where $\delta_i(v) = |V_i \cap N(v)|$. Notice that a vertex can satisfy these conditions for different partitions V_i. Let (v, i) be randomly chosen in the set S of all possible assignments. Then, v is assigned to subset V_i ($\rho[v] = i$) and the cut weight is set to $w_C + \bar{\Delta}_i(v)$ (see Heuristic – part 1).

Heuristic - part 1 Main loop of the building phase

```
while |V'| > 0 do
  V'' ← N(V̄') \ V̄'
  Δ̄_max ← max{Δ̄_i(v) | (v,i) ∈ V'' × K;  δ_i(v) ≥ 1}
  S ← {(v,i) ∈ V'' × K | Δ̄_i(v) = Δ̄_max;  δ_i(v) ≥ 1}
  randomly choose (v,i) in S
  V_i ← V_i ∪ {v} ; ρ[v] ← i
  V' ← V' \ {v}; V̄' ← V̄' ∪ {v}
  w_C ← w_C + Δ̄_max
  local_search(ρ, V, w_C)
end while
```

b) 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 w_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 ← false
  improved ← false
  for v ∈ L do
    if v is a not an articulation vertex of G[V_ρ[v]] then
      Δ̄_max ← max{Δ̄_i(v) | i ∈ K;  δ_i(v) ≥ 1}
      S ← {i ∈ K | Δ̄_i(v) = Δ̄_max;  δ_i(v) ≥ 1}
      if (Δ̄_ρ[v](v) ≤ Δ̄_max) then
        randomly choose i in S
        if i ≠ ρ(v) then
          w_C ← w_C + Δ̄_max − Δ̄_ρ[v]
          V_i ← V_i ∪ {v} ; V_ρ[v] ← V_ρ[v] \ {v}
          ρ[v] ← i
          change ← true
          improved ← Δ̄_ρ[v](v) < Δ̄_max
        end if
      end if
    end if
  end for
until ¬change or ¬improved for the last n_max iterations
```

VI. EXPERIMENTAL RESULTS

The experiments were run using Java 11 with CPLEX 12.10.0 on a Linux computer, with Intel Core I5-370 3.4GHz quad core processors and 8GB memory. CPLEX's working memory, tree sizes and number of threads were not limited; a time limit of 3600 seconds was imposed on each run.

A. LMR results

In Tables I–V the results obtained by LMR on randomly generated instances, for different values of $|V|$: 20, 30 and 40 and percentage densities[1] 10, 15, 20 and 25 are presented. For each pair of parameters, we have generated 10 instances, obtaining a benchmark of 120 instances.

In the tables, we present as $\%opt$ the percentage of instances solved to optimality within the given time (one hour). The running time *G-mean time* (in seconds) is given in shifted geometric mean $\prod_{i=1}^{n}(t_i + s)^{1/n} - s$, where n is the number of instances to be solved and $s = 10$. This way to compute the running times decreases the outliers' influence. *Mean time* (in seconds) is the average running time by instance based on the total running time.

Table I shows the global results over all instances, for 3, 6 and 9 partitions, without/with the tree widening inequalities (17)–(22).

[1]Strictly speaking, by densities we mean the probability that a pair of vertices are adjacent, expressed as a percentage.

k	without TWI			with TWI		
	%opt	G-Mean time	Mean time	%opt	G-Mean time	Mean time
3	94.17	46.85	433.64	96.67	37.30	376.35
6	90.00	46.82	576.63	94.17	32.51	398.73
9	97.50	20.31	234.82	100.00	12.75	94.42
all	93.89	36.09	415.03	96.94	25.77	289.83

TABLE I

AGGREGATE COMPLETIONS, G-MEAN AND MEAN RUNNING TIMES, ON ALL INSTANCES.

We can observe that, as expected, adding the tree widening inequalities improves the results for all k, both concerning the running times and the percentage of instances solved to optimality.

Tables II–III show the number of completions (#completion) on 40 instances for different k values and numbers of vertices, without and with the tree inequalities, respectively. Obviously, instances with higher number of vertices are more difficult to solve.

$\|V\|$	#completion on 40 instances			aggregated results on 120 instances		
	$k=3$	$k=6$	$k=9$	%opt	G-Mean time	Mean time
20	40	40	40	100.00	0.38	0.40
30	40	40	40	100.00	12.00	27.07
40	33	28	37	81.67	418.87	1217.64

TABLE II

AGGREGATE COMPLETIONS, G-MEAN AND MEAN RUNNING TIMES, FOR $|V| \in \{20, 30, 40\}$, WITHOUT TWI.

$\|V\|$	#completion on 40 instances			aggregated results on 120 instances		
	$k=3$	$k=6$	$k=9$	%opt	G-Mean time	Mean time
20	40	40	40	100.00	0.18	0.18
30	40	40	40	100.00	8.23	17.07
40	36	33	40	90.83	236.64	852.25

TABLE III

AGGREGATE COMPLETIONS, G-MEAN AND MEAN RUNNING TIMES, FOR $|V| \in \{20, 30, 40\}$, WITH TWI.

Tables IV–V show the results for different densities. In a similar way, instances with higher density are more difficult to solve.

density	#completion on 30 instances			aggregated results on 90 instances		
	$k=3$	$k=6$	$k=9$	%opt	G-Mean	Mean
10	30	30	30	100.00	11.94	63.75
15	30	30	30	100.00	26.11	147.96
20	27	27	30	93.33	51.33	558.46
25	26	21	27	82.22	82.89	889.96

TABLE IV

AGGREGATE COMPLETIONS, G-MEAN AND MEAN RUNNING TIMES, FOR DENSITY$\in \{10, 15, 20, 25\}$, WITHOUT TWI.

density	#completion on 30 instances			aggregated results on 90 instances		
	$k=3$	$k=6$	$k=9$	%opt	G-Mean time	Mean time
10	30	30	30	100.00	7.49	21.38
15	30	30	30	100.00	18.25	91.50
20	30	28	30	97.78	38.08	398.15
25	26	25	30	90.00	58.87	648.30

TABLE V

AGGREGATE COMPLETIONS, G-MEAN AND MEAN RUNNING TIMES, FOR DENSITY$\in \{10, 15, 20, 25\}$, WITH TWI.

We observe that, in all cases, the highest number of completions is when $k = 9$.

B. Heuristic results

In Tables VI–VIII, we present the results obtained by our heuristic. On each instance, $200 \times |V|$ tests were run and the best result was kept. To evaluate the quality of our heuristic, we compute two values. The value *%opt* represents the percentage of instances for which at least one run has found the optimal solution. The second value is the gap between the value computed by the heuristic (H) and the solution computed by LMR with TWI (O): $Gap = (O - H)/O$.

Mean time is the average time needed to launch $200 \times |V|$ tests, over all instances of the benchmark.

We may observe in Table VI, that both Mean Gap and Mean time decrease with greater k. The Mean Gap is very small in general with the largest gap observed being 0.03. This can be considered very acceptable. With k increasing, the number of successes is lower but the running time is better comparatively. This may be explained as follows. As the number of local optima increases with k, the heuristic, which always converges to one such local optimum, is likely to find it sooner but it is more unlikely that this optimum is the global one.

In Table VII, we observe that our heuristic gives good results for a small number of vertices. The number of local optima also increases for larger instances. Hence, it is more unlikely that the heuristic reaches the global optimum.
Regarding densities in Table VIII, we observe that, for the pattern of $k = 9$, the number of success is low for density 10. In this case, the local search phase of the heuristic is of no help because the number of articulation vertices in each sub-graph is higher.

k	%opt	Mean Gap	Mean time
3	60.83	0.0023	3.96
6	60.00	0.0018	3.07
9	56.67	0.0019	2.62
all	59.17	0.0020	3.22

TABLE VI

PERCENTAGE OF SUCCESS, GAP FROM THE OPTIMUM AND MEAN RUNNING TIME, ON ALL INSTANCES.

	#success on 40 instances			aggregated results on 120 instances		
$\|V\|$	$k=3$	$k=6$	$k=9$	%opt	Mean Gap	Mean time
20	39	38	40	97.50	0.0001	0.43
30	25	26	20	59.17	0.0021	2.13
40	9	8	8	20.83	0.0038	7.10

TABLE VII

PERCENTAGE OF SUCCESS, GAP FROM THE OPTIMUM AND MEAN RUNNING TIME, FOR $|V| \in \{20, 30, 40\}$.

	#success on 30 instances			aggregated results on 90 instances		
density	$k=3$	$k=6$	$k=9$	%opt	Mean Gap	Mean time
10	26	19	15	66.67	0.0024	2.70
15	19	19	18	62.22	0.0023	3.02
20	15	17	17	54.44	0.0021	3.38
25	13	17	18	53.33	0.0012	3.78

TABLE VIII

PERCENTAGE OF SUCCESS, GAP FROM THE OPTIMUM AND MEAN RUNNING TIME, FOR DENSITY$\in \{10, 15, 20, 25\}$.

VII. CONCLUSION

In this paper, we proposed exact and heuristic approaches to solve the Maximum k-Cutset or Connected k-Partitioning Problem. We proposed an ILP model to solve instances exactly and a heuristic local search-based algorithm. In order to address the problem of generating symmetric solutions for the exact approach we a) enforced the nomination of a representative vertex for each partition and, further, b) reduced alternate spanning trees (that ensured intra-partition connectivity). It is clear from Tables I – III that the latter contributes positively.

Comparing the two general approaches it is clear that on all but the smallest of graph orders the heuristic approach runs faster than the exact. For all graphs of order 40 the average G-Mean time for ILP is 236.64s while the *mean* time for the heuristic is 7.10s (see Table VII). Since the optimum was reached in one fifth of all runs one might argue that for an average investment of 35s of computing time the optimum could be confidently attained heuristically. Of course, further analysis would be required to verify that the %opt of 20.83 holds broadly.

REFERENCES

[1] Z. Ales, A. Knippel and A. Pauchet, Polyhedral combinatorics of the k-partitioning problem with representative variables, Discrete Applied Mathematics vol. 211, 2016, pp. 1-14.

[2] F. Barahona, A. R. Mahjoub, On the Cut Polytope, Mathematical Programming vol. 36, 1986 pp. 157-173.

[3] J. Barreiro-Gomez, C. Ocampo-Martinez and N. Quijano, Time-varying partitioning for predictive control design: Density-games approach, Journal of Process Control, vol. 75, 2019, pp. 1–14.

[4] J. Barreiro-Gomez, C. Ocampo-Martinez and N. Quijano, Partitioning for Large-scale Systems: A Sequential Distributed MPC Design, 20th IFAC World Congress, vol. 50 (1), 2017, pp. 8838—8843.

[5] F. Barahona and A. R. Mahjoub, On the cut polytope, Mathematical Programming vol. 36, 1986 pp. 157–173.

[6] A. Bertoni, P. Campadelli and G. Grossi, An approximation algorithm for the maximum cut problem and its experimental analysis, Discrete Applied Mathematics vol. 110, 2001, pp. 3-12.

[7] J.A. Bondy and U.S.R Murty, Graph Theory, Graduate Texts in Mathematics series, Springer, 2008, pp. 62.

[8] S. Borne, P. Fouilhoux, R. Grappe, M. Lacroix and P Pesneau, Branch-and-Cut algorithm for the connected-cut problem, 2014, in BookROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision Bordeaux, France

[9] Chang, J. Gérard, C. Chen and Y. Chen, Vertex and Tree Arboricities of Graphs, Journal of Combinatorial Optimization vol. 8, 2004, pp. 295-306.

[10] F. Della Croce, M. J. Kaminski and V. Th. Paschos, An exact algorithm for MAX-CUT in sparse graphs, Operations Research Letters vol. 35, 2007, pp. 403-408.

[11] P. Festa, P. M. Pardalos, M. G. C. Resende and C. C. Ribeiro, Randomized heuristics for the max-cut problem, Optimization Methods and Software, vol. 17, 2002, pp. 1033-1058.

[12] A. Frieze and M. Jerrum, I mproved Approximation Algorithms for MAX k-CUT and MAX BISECTION, in Integer Programming and Combinatorial Optimization. Publisher Balas E., Clausen J. (eds) IPCO 1995. LNCS, Springer, Berlin, Heidelberg, vol. 920, pp. 1-13.

[13] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefine programming, Journal of the Association for Computing Machinery, vol. 42, 1995, pp. 1115-1145.

[14] F. Hadlock, Finding a Maximum Cut of a Planar Graph in Polynomial Time, SIAM J. Comput., vol. 4 (3), 1975, pp. 221-225.

[15] D. J. Haglin and S. M. Venkatesan, Approximation and intractability results for the maximum cut problem and its variants, IEEE Transactions on Computers, vol. 40 (1), 1991, pp. 110-113.

[16] R. M. Karp, R.E. Miller, J.W. Thatcher, Reducibility among Combinatorial Problems, in Complexity of Computer Computations. The IBM Research Symposia Series. Springer, Boston, MA, vol. 1, 1972, pp. 85-103.

[17] S. Kahruman,E. Kolotoglu, S. Butenko, and I. V. Hicks, On Greedy Construction Heuristics for the MAX-CUT problem, Int. J. of Computational Science and Engineering, vol. 3 (3), 2007, pp. 211-218.

[18] G. A. Kochenberger, J.-K Hao,. L. H. Zhipeng and F. Glover, Solving large scale Max Cut problems via tabu search, J Heuristics, vol. 19, 2013, pp. 565-571.

[19] A. F. Ling, C. X. Xu and F. M. Xu, A discrete filled function algorithm embedded with continuous approximation for solving max-cut problems, European Journal of Operational Research, vol. 197, 2009, pp. 519-531.

[20] G. I. Orlova and Y. G. Dorfman, Finding the maximum cut in a graph, Engineering Cybernetics, vol. 10 (3), 1972, pp. 502-506.

[21] F. Rendl, G. Rinaldi and A. Wiegele, Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations, Mathematical Programming, vol. 121, 2008, pp. 307-335.

[22] S. Sahni and T. Gonzales, P-complete approximation problems, Journal of the ACM, vol. 23 (3), 1976, pp. 555-565.

[23] W. Zhu and C. Guo, A Local Search Approximation Algorithm for Max-k-Cut of Graph and Hypergraph, in proceedings of 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, Tianjin, China, 2011, pp. 236-240.