## Sum-Max Graph Partitioning Problem

Rémi Watrigant, Marin Bougeret, Rodolphe Giroudeau, Jean-Claude König

## To cite this version:

Rémi Watrigant, Marin Bougeret, Rodolphe Giroudeau, Jean-Claude König. Sum-Max Graph Partitioning Problem. ISCO: International Symposium on Combinatorial Optimization, Apr 2012, Athens, Greece. pp.297-308, 10.1007/978-3-642-32147-4_27 . lirmm-00738554

## HAL Id: lirmm-00738554

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00738554
Submitted on 4 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Sum-Max Graph Partitioning Problem^ 

R. Watrigant, M. Bougeret, R. Giroudeau, and J.-C. König<br>LIRMM-CNRS-UMR 5506-161, rue Ada 34090 Montpellier, France


#### Abstract

In this paper we consider the classical combinatorial optimization graph partitioning problem, with Sum-Max as objective function. Given a weighted graph $G=(V, E)$ and a integer $k$, our objective is to find a $k$-partition $\left(V_{1}, \ldots, V_{k}\right)$ of $V$ that minimizes $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k}$ $\max _{u \in V_{i}, v \in V_{j}} w(u, v)$, where $w(u, v)$ denotes the weight of the edge $\{u, v\} \in E$. We establish the $\mathcal{N} \mathcal{P}$-completeness of the problem and its unweighted version, and the $W[1]$-hardness for the parameter $k$. Then, we study the problem for small values of $k$, and show the membership in $\mathcal{P}$ when $k=3$, but the $\mathcal{N} \mathcal{P}$-hardness for all fixed $k \geq 4$ if one vertex per cluster is fixed. Lastly, we present a natural greedy algorithm with an approximation ratio better than $\frac{k}{2}$, and show that our analysis is tight.


## 1 Introduction

### 1.1 Description of the Problem

Graph partitioning problems are classical combinatorial optimization problems, where the objective is to partition vertices of a given graph into $k$ clusters, according to one or several criteria. In this article we focus on minimizing the sum of the heaviest edge between each pair of clusters. More formally, we study the following optimization problem:

SUM-MAX GRAPH PARTITIONING
Input: a connected graph $G=(V, E), w: E \rightarrow \mathbb{N}, k \leq|V|$
Output: a $k$-partition $\left(V_{1}, \ldots, V_{k}\right)$ of $V$ with $V_{i} \neq \emptyset \forall i=1, \ldots, k$
Goal: minimize $\sum_{\substack{i, j=1 \\ i>j}}^{k} \max _{\substack{u \in V_{i} \\ v \in V_{j}}} w(u, v)$
We denote by U-sum-max graph partitioning the unweighted version of the problem, where $w(e)=1 \forall e \in E$. The threshold value for the associated decision versions will be denoted by $C$.

All graphs studied here are supposed to be simple, non oriented and connected, unless otherwise stated. For a graph $G=(V, E)$, we define $n=|V|$ and $m=|E|$.

[^0]
### 1.2 Related Work

Graph partitioning problems are the heart of many practical issues, especially for applications where some items must be grouped together, as in the design of VLSI layouts [11], clustering of social and biological networks [12], or software re-modularization [16]. Because of the wide range of applications, several constraints and objective functions are considered. For instance, one can fix some vertices in clusters (like in the multicut problem), force equal-sized clusters etc., while optimizing (minimizing or maximizing) the sum of all edge weights between each pair of clusters (like in MIN-K-CUT and MAX-K-CUT), the sum of the edge weights (or the heaviest one) inside each cluster [8], or optimizing the cut ratio [12]. Some studies generalize many of these problems though one natural formalization: [7] gives computational lower bounds when the objective is to maximize some function over the inner edges of the clusters, [10] designs an $O^{*}\left(2^{n}\right)$ algorithm for a whole class of partition problems such as MAX-K-CUT, K-DOMATIC PARTITION or K-COLOURING, and [3] defines the M-PARTITIONING problem where the objective is to find a partition of the vertices respecting some constraints defined by a matrix $M$.

From a practical point of view, several heuristics for solving graph partitioning problems have been designed (some of them are surveyed in [14]) using many different techniques, as for example hierarchical algorithms, metaheuristics, spectral methods or modular decomposition.

Concerning complexity and approximation results, to the best of our knowledge sum-max graph partitioning has still not been studied directly. Among all of the previous problems, the two most relevant seem to be MIN-K-CUT (for SUM-MAX GRAPH PARTITIONING) and M-PARTITIONING (for U-SUM-MAX GRAPH PARTITIONING).

The only difference between SUM-MAX GRAPH PARTITIONing and MIN-KCUT is that the contribution of a pair of clusters is no longer the sum of all edge weights, but the heaviest one between these two clusters. MIN-K-CUT is $\mathcal{N} \mathcal{P}$-hard when $k$ is part of the input [5], but polynomial for every fixed $k$, with a $O\left(n^{k^{2}}\right)$ algorithm [6]. It is also $W[1]$-hard for the parameter $k[1]$, and there are several approximation algorithms, with ratios smaller than 2 [13]. Even if MIN-K-CUT and SUM-MAX GRAPH PARTITIONING seem related, it is not straightforward to directly re-use exact or approximation algorithms for MIN-K-CUT for our problem. Indeed, optimal solutions may have very different structure, as the number of edges between two clusters does not matter for SUM-MAX GRAPH PARTITIONING.

On the other hand, U-SUM-MAX GRAPH PARTITIONING is related to the problem of finding an homomorphism from a given graph $G$ to a fixed pattern graph $H$ [9], or equivalently to the M-Partitioning problem [3] (with 1's on the diagonal of the matrix $M$, and 0's and 1's elsewhere using notations of [3]). Indeed, given an input ( $G, k$ ) of U-SUM-MAX GRAPH PARTITIONING, the objective of our problem is to find the smallest graph $H$ (in terms of number of edges) with $k$ vertices such that $G$ is homomorphic to $H$. However, as one could expect targeting a fixed graph $H$ with $m^{*}$ edges may be harder than constructing any $k$ partition
of cost $m^{*}$. Thus, as discussed in details in Section 2.2, it will not be possible to directly use graph homomorphism to solve U-SUM-MAX GRAPH PARTITIONING.

### 1.3 Our Contributions

We show the following complexity results for SUM-MAX GRAPH PARTITIONING:

- when $k$ is part of the input, the problem and its unweighted variant are:
- $\mathcal{N} \mathcal{P}$-hard (and even $\frac{k}{k-1}$ non-approximable),
- $W[1]$-hard for the parameter $k$,
- for fixed $k=3$, the problem is solvable in polynomial time,
- for fixed $k \geq 4$, the problem is $\mathcal{N} \mathcal{P}$-hard if we fix one vertex per cluster in the input.

Then, we consider a natural greedy algorithm and prove that its approximation ratio is better than $k / 2$, and that the analysis is tight.

This article is organized as follows: the next section is devoted to the computational complexity of the general and restricted cases (with small values of $k$ ), while Section 3 is devoted to approximability.

## 2 Computational Complexity

In this section, we study the complexity of the problem and some variants. We prove that when $k$ is part of the input, the problem and its unweighted version are $\mathcal{N} \mathcal{P}$-hard, and $W[1]$-hard for the parameter $k$. The reduction used also leads to an non-approximability bound. Then, we investigate the complexity for small values of $k$, and show that it is polynomial for $k=3$, but $\mathcal{N} \mathcal{P}$-hard (even in the unweighted case) for all fixed $k \geq 4$ if we fix one vertex per cluster.

### 2.1 Hardness of SUM-MAX GRAPH PARTITIONING

Theorem 1. U-SUM-Max graph partitioning is $\mathcal{N} \mathcal{P}$-hard, and cannot be approximated within a factor $\rho<\frac{k}{k-1}$ (unless $\mathcal{P}=\mathcal{N P}$ ).

Proof. We reduce from the well-known $\mathcal{N} \mathcal{P}$-hard problem independent set. Let $G=(V, E)$ and $k \leq|V|$ be an instance of INDEPENDENT SET. We construct the following instance of U-SUM-max Graph partitioning: $G^{\prime}=\left(V^{\prime}, E^{\prime}\right)$ is a copy of $G$ plus a universal vertex $\alpha$, (i.e. $\alpha$ is connected to each vertex of $G$ ). We define the number of clusters $k^{\prime}=k+1$ and the cost of the desired partition $C^{\prime}=k$. This construction can clearly be computed in polynomial time.

- Let $S=\left\{s_{1}, \ldots, s_{k}\right\}$ be an independent set of size $k$ in $G$, with $s_{i} \in V$ for all $i \in\{1, \ldots, k\}$. We construct the following $k^{\prime}$-partition of $V^{\prime}$ :
- for all $i \in\{1, \ldots, k\}$, we define $V_{i}=\left\{s_{i}\right\}$
- $V_{k+1}=V^{\prime} \backslash S$

Since every pair of clusters in $\left\{V_{1}, \ldots, V_{k}\right\}$ is not adjacent, and since the set $V_{k+1}$ contains the vertex $\alpha$ which is connected to every other vertices, we have $\sum_{\substack{i, j=1 \\ i>j}}^{k^{\prime}} \max _{\substack{u \in V_{i} \\ v \in V_{j}}} w(u, v)=k=C^{\prime}$

- Suppose now that $G$ does not contain an independent set of size at least $k$ and let $\left(V_{1}, \ldots, V_{k+1}\right)$ be any $k^{\prime}$-partition of $G^{\prime}$. W.l.o.g., suppose that $\alpha \in V_{k+1}$. Since $\alpha$ is a universal vertex, the contribution of $V_{k+1}$ is $k$. Then, as the size of the maximum independent set is strictly lower than $k$, at least one pair of clusters among $\left(V_{1}, \ldots, V_{k}\right)$ is adjacent. Thus, we have $\sum_{\substack{i, j=1 \\ i>j}}^{k^{\prime}} \max _{\substack{u \in V_{i} \\ v \in V_{j}}} w(u, v) \geq k+1$,
which completes the $\mathcal{N} \mathcal{P}$-hardness proof. Moreover, notice that the previous reduction is a gap introducing reduction, where the gap between YES and NO instances is $\frac{k}{k-1}$, leading to the non-approximability result.

Corollary 1. sum-max graph partitioning is $\mathcal{N} \mathcal{P}$-hard.
Moreover, notice that the polynomial-time transformation given in Theorem 1 is also an FPT reduction [4] from INDEPENDENT SET parameterized by $k$ (which is a known $W$ [1]-hard problem) to U-SUM-max Graph Partitioning parameterized by the number of clusters. Indeed, the output parameter is clearly polynomial in the input parameter $\left(k^{\prime}=k+1\right)$, and the reduction can be computed in polynomial time. Thus, we deduce the following proposition.

Proposition 1. SUM-MAX GRAPH PARTITIONING (and its unweighted version) parameterized by the number of clusters is $W[1]$-hard.

### 2.2 Analysis of the Problem for Small $k$ Values

Enumerating Patterns Given the $\mathcal{N} \mathcal{P}$-hardness of the problem when $k$ is part of the input, it is natural to investigate the complexity of the problem for some small values of $k$.

Theorem 2. SUM-MAX GRAPH PARTITIONING is polynomial if $k=3$.
Proof. Let $G=(V, E)$ be a graph. The principle of the following algorithm is to enumerate all pairs (or triplets) of edges in order to find the heaviest edges between the clusters in an optimal solution (i.e. edges that will be taken into account in the solution value). Thus, for each fixed pairs (or triplets) of edges the algorithm tries to arrange all remaining vertices in clusters without changing the solution value.

Let us now distinguish two cases: one where an optimal solution contains only two edges between the clusters (the partition forms a path over the three clusters), and one where any optimal solution contains three edges (the partition forms a clique over the three clusters). Let ( $V_{1}, V_{2}, V_{3}$ ) be the partition we are building, and $\left(V_{1}^{*}, V_{2}^{*}, V_{3}^{*}\right)$ an optimal solution.

First case: one optimal solution contains only two edges Let us first assume that we know the two edges $e_{a}^{*}$ and $e_{b}^{*}$ that are taken into account in the optimal solution value (as depicted in Figure 1a). Let $a$ be the weight of the edge $e_{a}^{*}=\left\{a_{1}, a_{2}\right\}$ between $V_{1}^{*}$ and $V_{2}^{*}$, and $b$ be the weight of the edge $e_{b}^{*}=\left\{b_{1}, b_{2}\right\}$ between $V_{2}^{*}$ and $V_{3}^{*}$. Notice that four cases are possible, depending of the orientation of $e_{a}^{*}$ and $e_{b}^{*}$ (for example $a_{1}$ could be in $V_{1}^{*}$ or $V_{2}^{*}$ ). We assume that $a_{i} \in V_{i}^{*}$ and $b_{i} \in V_{i+1}^{*}$, and thus the algorithm will have to enumerate these four cases. Without loss of generality, we suppose $a \leq b$. In the first step, the algorithm mimics the optimal solution and adds $a_{1}$ to $V_{1}, a_{2}$ and $b_{1}$ to $V_{2}$, and $b_{2}$ to $V_{3}$. Let $S_{1}$ (resp. $S_{3}$ ) be the set of all vertices reachable from $V_{1}$ (resp. $V_{3}$ ) using edges of weight strictly greater than $a$ (resp. $b$ ). As the cost of the considered optimal solution is $a+b$, we know that (1) $S_{1} \subset V_{1}^{*}$ and $S_{3} \subset V_{3}^{*}$, (2) $S_{1} \cap S_{3}=\emptyset$ and (3) there is no edge between $S_{1}$ and $S_{3}$. Thus, in the second step the algorithm adds $S_{1}$ to $V_{1}$ and $S_{3}$ to $V_{3}$.

Finally, the algorithm assigns all remaining vertices to $V_{2}$. It is easy to see that this strategy will not create any forbidden edge (i.e. edge that increases the weight of the maximum edge between two clusters), as the remaining vertices were not adjacent to any vertex of $V_{1}$ (resp. $V_{3}$ ) using edges of weight strictly greater than $a$ (resp. $b$ ).

Second case: any optimal solution contains three edges Here again suppose that we know the three edges $e_{a}^{*}, e_{b}^{*}$ and $e_{c}^{*}$ that are taken into account in an optimal solution value (as depicted in Figure 1b). As before, we assume a fixed orientation of the guessed edges, to the price of the enumeration of a fixed number of cases. Let $a$ be the value of the edge $e_{a}^{*}=\left\{a_{1}, a_{2}\right\}$ between $V_{1}^{*}$ and $V_{3}^{*}$ (where $a_{1} \in V_{1}^{*}$, $a_{2} \in V_{3}^{*}$ ), $b$ be the value of the edge $e_{b}^{*}=\left\{b_{1}, b_{2}\right\}$ between $V_{1}^{*}$ and $V_{2}^{*}$ (where $b_{i} \in V_{i}^{*}$ ), and $c$ be the value of the edge $e_{c}^{*}=\left\{c_{1}, c_{2}\right\}$ between $V_{2}^{*}$ and $V_{3}^{*}$ (where $\left.c_{i} \in V_{i+1}^{*}\right)$. Without loss of generality, we suppose $a \leq b \leq c$.

Again, in the first step, the algorithm mimics the optimal solution and adds $a_{1}$ and $b_{1}$ to $V_{1}, b_{2}$ and $c_{1}$ to $V_{2}$, and $a_{2}$ and $c_{2}$ to $V_{3}$. Let $S_{1}$ (resp. $S_{3}$ ) be the set of vertices reachable from $V_{1}$ (resp. $V_{3}$ ) using edges of weight strictly greater than $b$ (resp. $c$ ). Using the same kind of arguments, we know that (1) $S_{i} \subset V_{i}^{*}$ (for $i \in\{1,3\}$ ), (2) $S_{1} \cap S_{3}=\emptyset$ and (3) there is no edge between $S_{1}$ and $S_{3}$ of weight strictly larger than $a$. Thus, we add $S_{i}$ to $V_{i}$.

Finally, the algorithm assigns all remaining vertices to $V_{2}$. As before, it is straightforward to see that this will not create any forbidden edge.

Overall complexity The overall algorithm consists in re-executing the previous routine for any pair and any triplet of edges, taking the best execution. Thus, the overall complexity is clearly polynomial, with a main factor in $\mathcal{O}\left(m^{3}\right)$ due to the enumeration.

A natural way to solve the problem would be to extend the previous algorithm by enumerating all edges between clusters (or all $k$-uplets of vertices), and then arranging the remaining vertices using the same kind of "dominating rules". Moreover, the corresponding complexity (in $\Omega\left(n^{f(k)}\right)$ ) would be satisfying, as the problem is $W[1]$-hard. Here we show that this strategy is hopeless


Fig. 1: Illustration of the polynomial algorithm for $k=3$. Bold arrows represent assignments to clusters. (1a): One optimal solution contains 2 edges (1b): Any optimal solution contains 3 edges
(even for the unweighted case), because of the $\mathcal{N} \mathcal{P}$-hardness of the following problem (the proof is available in [15]):

SUM-MAX GRAPH PARTITIONING WITH FIXED VERTICES
Input: a graph $G=(V, E), w: E \rightarrow \mathbb{N}, k \leq|V|, C \in \mathbb{N}$, a set $\left\{v_{1}, \ldots, v_{k}\right\} \subseteq V$ Question: Is there a $k$-partition $\left(V_{1}, \ldots, V_{k}\right)$ of $V$ such that $\sum_{\substack{i, j=1 \\ i>j}}^{k} \max _{\substack{u \in V_{i} \\ v \in V_{j}}}$ $w(u, v) \leq C$ and $v_{i} \in V_{i} \forall i \in\{1, \ldots, k\} ?$

Proposition 2. SUM-MAX GRAPH PARTITIONING WITH FIXED VERTICES (and its unweighted version) is $\mathcal{N \mathcal { P }}$-hard for all fixed $k \geq 4$.

Link with Graph Homomorphisms As said before, U-SUM-max GRAPH PARTITIONING is related to the problem of finding an homomorphism between a graph $G$ (our input) and a fixed graph $H$ that has $k$ vertices ${ }^{1}$.

Indeed, the existence of a $k$-partition of cost $C$ for a given graph $G$ implies that there exists an homomorphism from $G$ to some graph with $k$ vertices and $C$ edges. Conversely, an homomorphism from $G$ to a graph $H$ with $k$ vertices and $C$ edges implies that there exists a $k$-partition of cost at most $C$.

Let us now recall the List-Graph homomorphism to H (L-HOMH) problem [2], given a fixed pattern graph $H=\left(V_{H}, E_{H}\right)$ :

LIST-GRAPH HOMOMORPHISM TO $H$
Input: a graph $G=\left(V_{G}, E_{G}\right)$ and for all $v \in V_{G}$, a list $L(v) \subseteq V_{H}$
Question: Is there a graph homomorphism $h: V_{G} \rightarrow V_{H}$ such that for all $v \in V_{G}$ $h(v) \subseteq L(v)$ ?

[^1]Thus, U-SUM-MAX GRAPH PARTITIONING is related to a special case of LHOMH, where all lists are equal to $V_{H}$.

In [2], the authors study a variant of L-HOMH, called ONE OR ALL LIST homomorphism to $H$ (OAL-HOMH), where for all $v \in V_{G}, L(v)$ is either a singleton or $V_{H}$. Thus, U-SUM-max graph partitioning with fixed verTICES consists in finding the minimum $k$ vertices graph $H$ (in terms of number of edges) such that $G$ is homomorphic to $H$, with singletons for vertices that are fixed, and $V_{H}$ for others.

It is clear that a polynomial algorithm for OAL-HOMH would imply a $O\left(n^{f(k)}\right)$ algorithm for U-SUM-MAX GRAPH PARTITIONING (by enumerating all possible patterns for any possible value of the optimal). Unfortunately, the authors show that depending on the shape of $H$, OAL-HOMH (and thus HOMH) can be $\mathcal{N} \mathcal{P}$-hard. More formally, they show that OAL-HOMH is $\mathcal{N} \mathcal{P}$-hard if $H$ contains a chord-less cycle of size $k \geq 4$ as an induced sub-graph, and is polynomial otherwise. Actually, it appears that the reduction presented in [2] is very close to our proof of Proposition 2.

## 3 A Polynomial-Time Approximation Algorithm

In this section we consider a simple greedy algorithm for SUM-MAX GRAPH PARTITIONING and prove that its approximation ratio is better than $k / 2$. Moreover, we show that our analysis is tight.

### 3.1 Presentation of the Greedy Algorithm

It is clear that a feasible solution can be obtained by removing edges, until the number of connected components (which will represent clusters) reaches $k$. As the cost of such a solution depends on the weight of removed edges, it is natural to consider them in non decreasing order of weights. Thus, we consider the greedy algorithm given by Algorithm 1, whose running time is clearly bounded by $O(|E| \log |E|)$. Actually, this algorithm corresponds to the SPLIT algorithm of [13], which gives a $(2-2 / k)$-approximation algorithm for MIN-K-CUT.

```
Algorithm 1 a greedy algorithm for SUM-MAX GRAPH PARTITIONING
    Sort \(E\) in non decreasing order of weights (ties are broken arbitrarily)
    \(j \leftarrow 0\)
    for \(i=1\) to \(k-1\) do
        while \(G\) has \(i\) connected components do
            \(G \leftarrow G \backslash\left\{e_{j}\right\}\)
            \(j \leftarrow j+1\)
        end while
        // we denote by \(w_{i}\) the weight of the last removed edge
    end for
    return connected components of \(G\)
```


### 3.2 Analysis of the Algorithm

Notations Let $\mathcal{I}=(G, k)$ be an instance of SUM-max Graph Partitioning. We define $\Omega_{k}=\frac{k(k-1)}{2}$, and $\theta=\max \left\{\frac{w(e)}{w\left(e^{\prime}\right)}: e, e^{\prime} \in E, e \neq e^{\prime}, w\left(e^{\prime}\right) \geq w(e)\right\}$. For a solution $S=\left\{S_{1}, \ldots, S_{k}\right\}$ of the problem, we associate the set $C_{S}=\left\{c_{1}, \ldots, c_{p_{s}}\right\}$ of edges of maximum weight between each pair of clusters, with $p_{S} \leq \Omega_{k}$. The value of the solution is then defined by $\operatorname{val}(S)=\sum_{i=1}^{p_{S}} w\left(c_{i}\right)$.

Let $A=\left\{A_{1}, \ldots, A_{k}\right\}$ be the solution returned by Algorithm 1 , and $\left\{{ }^{i} A_{1}, \ldots\right.$, $\left.{ }^{i} A_{i}\right\}$ the partial solution at the beginning of step $i$. The while loop consists in separating a cluster ${ }^{i} A_{t}$ (for some $t \in\{1, \ldots, i\}$ ) into two clusters ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$. Thus, when separating ${ }^{i} A_{t}$, we add to $C_{A}$ the edge of maximum weight between ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$, and at most $(i-1)$ edges (called the unexpected edges) between ${ }^{i} A_{t}^{1}$ or ${ }^{i} A_{t}^{2}$ and the other clusters (cf Figure 2). We thereby add to the solution value one term $w_{i}$ (between ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$ ) and $(i-1)$ terms $\left(\alpha_{i}^{j}\right)_{j=1 . .(i-1)}$. For $j \in\{1, \ldots,(i-1)\}$, if the edge of maximum weight between ${ }^{i} A_{t}$ and ${ }^{i} A_{j}$ has one endpoint in ${ }^{i} A_{t}^{1}$ (resp. ${ }^{i} A_{t}^{2}$ ), then $\alpha_{i}^{j}$ is equal to the edge of maximum weight between ${ }^{i} A_{t}^{2}$ (resp. ${ }^{i} A_{t}^{1}$ ) and ${ }^{i} A_{j}$, or 0 if the two clusters are not adjacent. By definition, we have $\operatorname{val}(A)=\sum_{i=1}^{k-1}\left(w_{i}+\sum_{j=1}^{i-1} \alpha_{i}^{j}\right)$.


Fig. 2: Dashed lines represent edges of maximum weight between ${ }^{i} A_{t}$ and other clusters, already in $C_{A}$, solid lines represent the at most $(i-1)$ new edges added to $C_{A}$.

Preliminaries Let us now state several properties of the algorithm that will be the base of the approximation result (Theorem 3). First, It is clear by construction that $w_{1} \leq w_{2} \leq \ldots \leq w_{k-1}$. Then, we have the following result:
Lemma 1. Let us consider the beginning of step $i$, and the corresponding $i$ partition $\left\{{ }^{i} A_{1}, \ldots,{ }^{i} A_{i}\right\}$. Then, for any $t \in\{1, \ldots, i\}$ we have $\sum_{\substack{j=1 \\ j \neq t}}^{i} w\left(e_{t, j}\right) \leq$ $\sum_{j=1}^{i-1} w_{j}$, where $e_{t, j}$ denotes the edge of maximum weight between ${ }^{i} A_{t}$ and ${ }^{i} A_{j}$.

Proof. We prove it by induction over $i$. Statement is clearly true for the first steps (case $i=1$ is meaningless since we have only 1 cluster, and case $i=2$ is true since there is only two clusters, and thus only one edge of maximum weight between them). We are at the beginning of Step $i+1$ : during Step $i,{ }^{i} A_{t}$ has been separated into ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$, thus incurring an additional weight of $w_{i}$.

For $j_{0} \neq t$, notice that edge $e_{j_{0}, t}$ (edge between ${ }^{i} A_{j_{0}}$ and ${ }^{i} A_{t}$, before the split) is now replaced by two edges $e_{j_{0}, t_{1}}$ and $e_{j_{0}, t_{2}}$, with $\max \left(w\left(e_{j_{0}, t_{1}}\right), w\left(e_{j_{0}, t_{2}}\right)\right)=$ $w\left(e_{j_{0}, t}\right)$. Let us now bound the weight of edges out-coming from ${ }^{i} A_{j_{0}}$. W.l.o.g., suppose that $w\left(e_{j_{0}, t_{1}}\right)=w\left(e_{j_{0}, t}\right)$, and let ${ }^{i} S_{j_{0}}$ be the sum of all heaviest edges linking ${ }^{i} A_{j_{0}}$ to each one of the other clusters (including ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$ ). Thus, we have ${ }^{i} S_{j_{0}}=\sum_{\substack{j \neq j_{0}, j \neq t}}^{i=1} w\left(e_{j_{0}, j}\right)+\underbrace{w\left(e_{j_{0}, t_{1}}\right)}_{w\left(e_{j_{0}, t}\right)}+\underbrace{w\left(e_{j_{0}, t_{2}}\right)}_{\leq w_{i}} \leq \sum_{j=1}^{i-1} w_{j}+w_{i}$ (using the induction hypothesis).

Same arguments hold for sets ${ }^{i} A_{t}^{1}$ and ${ }^{i} A_{t}^{2}$, which completes the proof.
Corollary 2. Let us consider the beginning of step $i$, and the corresponding $i$ partition $\left\{{ }^{i} A_{1}, \ldots,{ }^{i} A_{i}\right\}$. When splitting ${ }^{i} A_{t}$, the total weight of the unexpected edges is upper bounded as follows: $\sum_{j=1}^{i-1} \alpha_{i}^{j} \leq \theta \sum_{j=1}^{i-1} w_{j}$.

Proof. We re-use notation $e_{j, t}$ of Lemma 1. Let $\tilde{e}_{j, t}$ (with $j \neq t$ ) be the unexpected edge between ${ }^{i} A_{j}$ and ${ }^{i} A_{t}$. For example, if $e_{j, t}$ was in fact an edge between ${ }^{i} A_{j}$ and ${ }^{i} A_{t}^{1}, \tilde{e}_{j, t}$ is the edge between ${ }^{i} A_{j}$ and ${ }^{i} A_{t}^{2}$. By definition of $\theta$, we have $w\left(\tilde{e}_{j, t}\right) \leq \theta w\left(e_{j, t}\right)$, and thus $\sum_{j=1}^{i-1} \alpha_{i}^{j}=\sum_{j=1, j \neq t}^{i} w\left(\tilde{e}_{j, t}\right) \leq \theta \sum_{j=1}^{i-1} w_{j}$ (by Lemma 1).

Let us now prove the following lower bound on the optimal value.
Lemma 2. Let $S$ be any $(i+1)$-partition, with $C_{S}=\left\{c_{1}, \ldots, c_{p_{S}}\right\}$. We have: $\sum_{j=1}^{p_{S}} w\left(c_{j}\right) \geq \sum_{j=1}^{i} w_{j}$

Proof. We prove it by induction over $i$. The statement is clearly true for the first step, since Algorithm 1 gives an optimal 2-partition. Consider now an $(i+1)$ partition $S$, with $C_{S}=\left\{c_{1}, \ldots, c_{p_{S}}\right\}$. Let $w_{M}=\max _{j=1 \ldots p_{S}} w\left(c_{j}\right)$, and let $\left(S_{i_{1}}, S_{i_{2}}\right)$ be the two sets in $S$ containing both endpoints of an edge of weight $w_{M}$. Considering the $i$-partition created when merging $S_{i_{1}}$ and $S_{i_{2}}$ in $S$, and using the induction hypothesis, we have: $\sum_{j=1}^{p_{S}} w\left(c_{j}\right)-w_{M} \geq \sum_{j=1}^{i-1} w_{j}$. Finally, notice that by construction any $(i+1)$-partition must have an edge of weight at least $w_{i}$, since after removing all edges of weight strictly smaller than $w_{i}$ in our algorithm, we still not have an $(i+1)$-partition. This leads to $w_{M} \geq w_{i}$ and to the desired inequality.

Proof of the Approximation Ratio We now turn to our main theorem, and prove that Algorithm 1 has an approximation ratio better than $\frac{k}{2}$.

Theorem 3. Algorithm 1 is a $\left(1+\left(\frac{k}{2}-1\right) \theta\right)$-approximation algorithm.

Proof. Using Lemma 2 with an optimal solution, it is sufficient to show that $\operatorname{val}(A) \leq\left(1+\left(\frac{k}{2}-1\right) \theta\right) \sum_{i=1}^{k-1} w_{i}$. Let us prove it by induction over $k$. Statement is clear for $k=2$. Suppose now that the result is true for all $k=1,2, \ldots, t$ and let us show that it remains true for $k=t+1$. By the induction hypothesis, we have:

$$
\begin{aligned}
\operatorname{val}(A) & \leq\left(1+\left(\frac{t}{2}-1\right) \theta\right) \sum_{i=1}^{t-1} w_{i}+w_{t}+\sum_{j=1}^{t-1} \alpha_{t}^{j} \\
& =\left(1+\left(\frac{t}{2}-1\right) \theta\right) \sum_{i=1}^{t-1} w_{i}+w_{t}+\frac{1}{2} \sum_{j=1}^{t-1} \alpha_{t}^{j}+\frac{1}{2} \sum_{j=1}^{t-1} \alpha_{t}^{j} \\
& \leq\left(1+\left(\frac{t}{2}-1\right) \theta\right) \sum_{i=1}^{t-1} w_{i}+w_{t}+\frac{1}{2} \theta \sum_{j=1}^{t-1} w_{j}+\frac{1}{2} \sum_{j=1}^{t-1} \alpha_{t}^{j} \quad \text { using Lemma } 1 \\
& \leq\left(1+\left(\frac{t}{2}-1\right) \theta\right) \sum_{i=1}^{t-1} w_{i}+w_{t}+\frac{1}{2} \theta \sum_{j=1}^{t-1} w_{j}+\frac{1}{2}(t-1) \theta w_{t} \quad \text { as } \alpha_{t}^{j} \leq \theta w_{t} \\
& \leq\left(1+\left(\frac{t+1}{2}-1\right) \theta\right) \sum_{i=1}^{t-1} w_{i}+w_{t}+\left(\frac{t+1}{2}-1\right) \theta w_{t}
\end{aligned}
$$

Which gives the desired inequality.
Thus, Algorithm 1 becomes arbitrarily good as $\theta$ tends to 0 , i.e. when the gap on the weight of any pair of edges becomes arbitrarily large. This is not surprising, as Algorithm 1 only focuses on edge weights, rather than the structure of the graph. Moreover, notice that SUm-max graph partitioning remains $\mathcal{N} \mathcal{P}$-hard even if all edge weights are different (and thus even when $\theta$ tends to 0 ). Indeed, the reduction presented in the proof of Theorem 1 can be adapted using classical scaling arguments (assigning weight $1+i \epsilon$ to edge $i$ ).

It appears from the previous proof that the $\frac{k}{2}$ factor is mainly due to the excessive number of edges in the solution given by Algorithm 1. Indeed, in the worst case (of the unweighted problem) this solution forms a clique of size $k$ over the clusters, while the optimal forms a tree, resulting in a $\frac{k(k-1)}{2} /(k-1)=\frac{k}{2}$ ratio on the number of edges. This insight is the key point of the following tightness result, where the instance is designed such that the lower bound $\left(\sum\left(w_{j}\right)\right)$ becomes tight.

Proposition 3. Approximation ratio of Algorithm 1 is tight.
Proof. Let $k \in \mathbb{N}$. We define the instance $I_{k}$, composed of a split graph $G=$ $(C \cup S, E, w)$ (with $C$ as an induced clique and $S$ as an induced stable set) with as many edges as possible. We define $C=\left\{c_{1}, \ldots, c_{k}\right\}$ and $S=\left\{s_{1}, \ldots, s_{k}\right\}$. Finally, $w(e)=1$ for all $e \in E$. Let us now define three categories of edges:

- first category: $X=\left\{\left\{c_{i}, s_{j}\right\}\right.$ such that $i \neq j$ or $\left.j=1\right\}$,
- second category: $Y=\left\{\left\{c_{i}, c_{j}\right\}\right.$ such that $\left.i \neq j\right\}$,
- third category: $Z=\left\{\left\{c_{i}, s_{j}\right\}\right.$ such that $i=j$ and $\left.j \neq 1\right\}$.

An example of such a graph is presented in Figure 3.


Fig. 3: (a): Example of a graph that reaches the ratio. First category of edges is represented with dashed lines, second category with solid lines, third category with bold lines (b): Solution given by Algorithm 1 (c): Optimal solution

Since Algorithm 1 sort edges of equal weight arbitrarily, suppose that it starts by removing edges from $X$, then those from $Y$. At this point, it is easy to see that a $(k+1)$-partition is created. Then, since each pair of clusters is adjacent, the value of this solution is $\frac{(k+1) k}{2}$. On the contrary, consider the following $(k+1)$ partition $\left(V_{1}, \ldots, V_{k}\right)$ : for all $j \in\{1, \ldots, k\}, V_{j}=\left\{s_{j}\right\}$, and $V_{k+1}=C$. The value of this solution is $k$, (it is thus an optimal one). Then, notice that $\theta=\max \left\{\frac{w(e)}{w\left(e^{\prime}\right)}\right.$ : $\left.e, e^{\prime} \in E, e \neq e^{\prime}, w\left(e^{\prime}\right) \geq w(e)\right\}=1$. Let $\mathcal{A}\left(I_{k}\right)$ and $\mathcal{O P} \mathcal{T}\left(I_{k}\right)$ denote respectively the value of the solution given by Algorithm 1 and the value of an optimal solution for $I_{k}$. We have $\frac{\mathcal{A}\left(I_{k}\right)}{\mathcal{O P \mathcal { T }}\left(I_{k}\right)}=\frac{k+1}{2}$, which proves the result (we are looking for a $(k+1)$-partition). Notice that it is possible to obtain the same result without using the fact that edges of equal weight are sorted arbitrarily in Algorithm 1, by assigning different edge weights that will respect the order of removed edges presented above, and are large enough compared with $|E|$.

## 4 Conclusion

In this paper we investigated the complexity and approximability of a variant of the classical graph partitioning problem with sum-max as objective function. Concerning exact solving, we showed that the pattern enumeration strategy leads to a polynomial algorithm for $k=3$ but becomes hopeless for $k \geq 4$, since the problem becomes $\mathcal{N} \mathcal{P}$-hard when fixing one vertex per cluster. Thus, it remains now to close the complexity study of the problem for fixed $k$ by either providing a $O\left(n^{f(k)}\right)$ algorithm (like for miN-K-CUT [6]), or getting an $\mathcal{N} \mathcal{P}$-hardness result. From the point of view of approximability, we showed that the greedy algorithm
presented in this paper behaves correctly regarding to the weights but neglects somehow the structure of the graph, which should encourage other investigations in this sense.

## References

1. R. G. Downey, V. Estivill-castro, M. R. Fellows, E. Prieto, and F. A. Rosamond. Cutting up is hard to do: The parameterized complexity of k -cut and related problems. In Electronic Notes in Theoretical Computer Science 78, pages 205218. Elsevier Science Publishers, 2003.
2. T. Feder and P. Hell. List homomorphisms to reflexive graphs. Journal of Combinatorial Theory, Series B, 72(2):236-250, 1998.
3. T. Feder, P. Hell, S. Klein, and R. Motwani. Complexity of graph partition problems. In Proceedings of the thirty-first annual ACM symposium on Theory of computing, STOC '99, pages 464-472, New York, NY, USA, 1999. ACM.
4. J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006.
5. M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.
6. O. Goldschmidt and D. S. Hochbaum. Polynomial algorithm for the k-cut problem. In Proceedings of the 29th Annual Symposium on Foundations of Computer Science, SFCS '88, pages 444-451, Washington, DC, USA, 1988. IEEE Computer Society.
7. T. Gonzalez. On the computational complexity of clustering and related problems. In R. Drenick and F. Kozin, editors, System Modeling and Optimization, volume 38 of Lecture Notes in Control and Information Sciences, pages 174-182. Springer Berlin / Heidelberg, 1982. 10.1007/BFb0006133.
8. M. Hansen, P.and Delattre. Complete-link cluster analysis by graph coloring. Journal of the American Statistical Association, 73(362):pp. 397-403, 1978.
9. P. Hell. Graphs and Homomorphisms. Oxford University Press, 2004.
10. M. Koivisto. An $\mathrm{O}\left(2^{n}\right)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In in Proceedings of the 47 th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), IEEE, pages 583-590, 2006.
11. J. Li, L. Behjat, and B. Schiffner. A structure based clustering algorithm with applications to vlsi physical design. In Proceedings of the Fifth International Workshop on System-on-Chip for Real-Time Applications, IWSOC '05, pages 270-274, Washington, DC, USA, 2005. IEEE Computer Society.
12. S. B. Patkar and H. Narayanan. An efficient practical heuristic for good ratio-cut partitioning. In Proceedings of the 16th International Conference on VLSI Design, VLSID '03, pages $64-$, Washington, DC, USA, 2003. IEEE Computer Society.
13. H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. SIAM J. Comput., 24(1):101-108, February 1995.
14. S. E. Schaeffer. Graph clustering. Computer Science Review, 1(1):27-64, 2007.
15. R. Watrigant, M. Bougeret, R. Giroudeau, and J.-C. König. Sum-max graph partitioning problem. Technical Report RR-12015, LIRMM-CNRS-UMR 5506, 2012.
16. T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97), WCRE '97, pages 33-, Washington, DC, USA, 1997. IEEE Computer Society.

[^0]:    * This work has been funded by grant ANR 2010 BLAN 021902

[^1]:    ${ }^{1}$ recall that $G=\left(V_{G}, E_{G}\right)$ is homomorphic to $H=\left(V_{H}, E_{H}\right)$ iff there is a function $h: V_{G} \rightarrow V_{H}$ such that for all $\{u, v\} \in E_{G},\{f(u), f(v)\} \in E_{H}$

