Sum-Max Graph Partitioning Problem

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

Sum-Max Graph Partitioning Problem*

R. Watrigant, M. Bougeret, R. Giroudeau, and J.-C. König

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 $(V_1, \ldots, V_k)$ 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 $\mathcal{NP}$-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 $\mathcal{NP}$-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 $G = (V, E)$, $w : E \to \mathbb{N}$, $k \leq |V|$
Output: a k-partition $(V_1, ..., V_k)$ of V with $V_i \neq \emptyset \ \forall i = 1, ..., k$
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 $m = |E|$.

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 [12], clustering of social and biological networks [13], 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

* This work has been funded by grant ANR **2010** BLAN **021902**

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 [9], or optimizing the cut ratio [13]. Some studies generalize many of these problems though one natural formalization: [8] gives computational lower bounds when the objective is to maximize some function over the inner edges of the clusters, [11] designs an $O^*(2^n)$ algorithm for a whole class of partition problems such as MAX-K-CUT, K-DOMATIC PARTITION or K-COLOURING, and [4] 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 [15]) using many different techniques, as for example hierarchical algorithms, meta-heuristics, 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-K-CUT 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 $\mathcal{NP}$-hard when k is part of the input [6], but polynomial for every fixed k, with a $O(n^{k^2})$ algorithm [7]. It is also $W[1]$-hard for the parameter k [2], and there are several approximation algorithms, with ratios smaller than 2 [14]. 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 [10], or equivalently to the M-PARTITIONING problem [4] (with 1's on the diagonal of the matrix M, and 0's and 1's elsewhere using notations of [4]). 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:
  - $\mathcal{NP}$-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 $\mathcal{NP}$-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 $\mathcal{NP}$-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 $\mathcal{NP}$-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 $\mathcal{NP}$-hard, and cannot be approximated within a factor $\rho < \frac{k}{k-1}$ (unless $\mathcal{P} = \mathcal{NP}$).*

Proof. We reduce from the well-known $\mathcal{NP}$-hard problem INDEPENDENT SET:

> INDEPENDENT SET
> **Input:** a graph $G = (V, E)$, $k \in \mathbb{N}$,
> **Question:** Does G contain an independent set of size at least k ?

Let Φ be an instance of INDEPENDENT SET. We construct Φ', an instance of U-SUM-MAX GRAPH PARTITIONING as follows: $G' = (V', E')$ is a copy of G plus a universal vertex α, (i.e. α is connected to each vertex of G). We define the number of clusters $k' = k + 1$ and the cost of the desired partition $C' = k$. This construction can clearly be computed in polynomial time.

- Let $S = \{s_1, ..., s_k\}$ be an independent set of size k in G, with $s_i \in V$ for all $i \in \{1, ..., k\}$. We construct the following k'-partition of V':
  - for all $i \in \{1, ..., k\}$, we define $V_i = \{s_i\}$
  - $V_{k+1} = V' \backslash S$

  Since every pair of clusters in $\{V_1, ..., V_k\}$ is not adjacent, and since the set V_{k+1} contains the vertex α which is connected to every other vertices, we have

$$\sum_{\substack{i,j=1 \\ i>j}}^{k'} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v) = k = C'$$

- Suppose now that G does not contain an independent set of size at least k and let $(V_1, ..., V_{k+1})$ be any k'-partition of G'. W.l.o.g., suppose that $\alpha \in V_{k+1}$. Since α 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 $(V_1, ..., V_k)$ is adjacent. Thus, we have

$$\sum_{\substack{i,j=1 \\ i>j}}^{k'} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v) \geq k + 1,$$

which completes the $\mathcal{NP}$-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 $\mathcal{NP}$-hard.*

Moreover, notice that the polynomial-time transformation given in Theorem 1 is also an *FPT reduction* [5] 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 ($k' = k + 1$), 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.*

Thus, there is no hope to find an exact algorithm to solve the problem in time $O(f(k).p(n))$, for some function f and some polynomial p.

2.2 Analysis of the Problem for Small k Values

Enumerating Patterns Given the $\mathcal{NP}$-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. *There exists a polynomial-time algorithm for* SUM-MAX GRAPH PARTITIONING *for the special case $k = 3$.*

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 (V_1^*, V_2^*, V_3^*) 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 $e_a^* = \{a_1, a_2\}$ between V_1^* and V_2^*, and b be the weight of the edge $e_b^* = \{b_1, b_2\}$ 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 $e_a^* = \{a_1, a_2\}$ between V_1^* and V_3^* (where $a_1 \in V_1^*$, $a_2 \in V_3^*$), b be the value of the edge $e_b^* = \{b_1, b_2\}$ between V_1^* and V_2^* (where $b_i \in V_i^*$), and c be the value of the edge $e_c^* = \{c_1, c_2\}$ between V_2^* and V_3^* (where $c_i \in V_{i+1}^*$). 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 $\mathcal{O}(m^3)$ due to the enumeration.

□

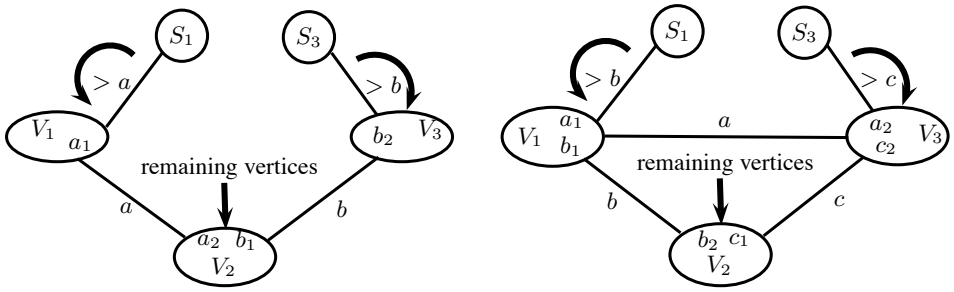

Fig. 1: Illustration of the polynomial algorithm for $k = 3$. Bold arrows represent assignments to clusters. (1a): One optimal solution contains 2 edges (1b): Any optimal solution contains 3 edges

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 $\Omega(n^{f(k)})$) would be satisfying, as the problem is $W[1]$-hard. Here we show that this strategy is hopeless (even

for the unweighted case), because of the $\mathcal{NP}$-hardness of the following problem (and its unweighted version):

SUM-MAX GRAPH PARTITIONING WITH FIXED VERTICES
Input: a graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \leq |V|$, $C \in \mathbb{N}$, a set $\{v_1, ..., v_k\} \subseteq V$
Question: Is there a k-partition $(V_1, ..., V_k)$ 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 $v_i \in V_i \ \forall i \in \{1, ..., k\}$?

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

Proof. Membership in $\mathcal{NP}$ is clear. We reduce from the well known $\mathcal{NP}$-hard problem 3-COLOURING:

3-COLOURING
Input: a graph $G = (V, E)$
Question: Is there a 3-colouring $c : V \longrightarrow \{1, 2, 3\}$ such that for all $\{u, v\} \in E$ we have $c(u) \neq c(v)$?

Given $G = (V, E)$, we construct the following instance of SUM-MAX GRAPH PARTITIONING WITH FIXED VERTICES with $k \geq 4$ fixed. Let G' containing all vertices of G (but not its edges), for which we add k vertices $X = \{x_1, x_2, x_3, x_4, ..., x_k\}$ which are the vertices that must belong to a different partition, *i.e.* $x_i \in V_i \ \forall i \in \{1, ..., k\}$. We connect x_i to $x_{i+1} \ \forall i \in \{1, 2, 3\}$, x_4 to x_1, and then x_i to x_{i+1} for all $i = 4...(k-1)$. For each edge $\{u, v\} \in E$, we connect u and v to x_2, and add a gadget composed of 6 vertices $\{a_1, a_2, b_1, b_2, c_1, c_2\}$ with the following adjacencies:

- a_1 is adjacent to u, a_2, x_4 and x_1.
- a_2 is adjacent to v, a_1, x_1 and x_2.
- b_1 is adjacent to u, b_2, x_2 and x_3.
- b_2 is adjacent to v, b_1, x_3 and x_4.
- c_1 is adjacent to u, c_2, x_3 and x_4.
- c_2 is adjacent to v, c_1, x_4 and x_1.

The gadget is illustrated in Figure 2. Notice that all edge weights are equal to 1 in G'. We claim that G has a proper 3-colouring if and only if G' has a k-partition of value k, or more precisely if and only if one can assign each vertex of G' to $\{V_1, V_2, V_3, V_4\}$ without creating any additional edge to the existing ones (between vertices of X). Any additional edge will be called a *forbidden edge*. Notice that the solution cannot be less that k because of the adjacencies of X. We actually show that if a vertex v of G is coloured with the color 1, 2 or 3, then this same vertex in G' will be respectively in V_1, V_2 or V_3, and conversely.

The central idea of our gadget is the following: if a vertex v is adjacent to V_r, V_s and V_t, with $r, s, t \in \{1, 2, 3, 4\}$ and $r \neq s \neq t$, then this vertex must be in the set which is "in the middle" of the other two in order not to create any forbidden edge.

- Let us show that if such a partition exists, then the two vertices of every edge cannot be in the same cluster (it is clear that none of them can be in V_j for all $j \geq 4$ because of the adjacencies with x_2 which would create a forbidden edge). Suppose for sake of a contradiction that there exists two adjacent vertices u and v such that $\{u, v\} \subseteq V_1$ or $\{u, v\} \subseteq V_2$ or $\{u, v\} \in V_3$.

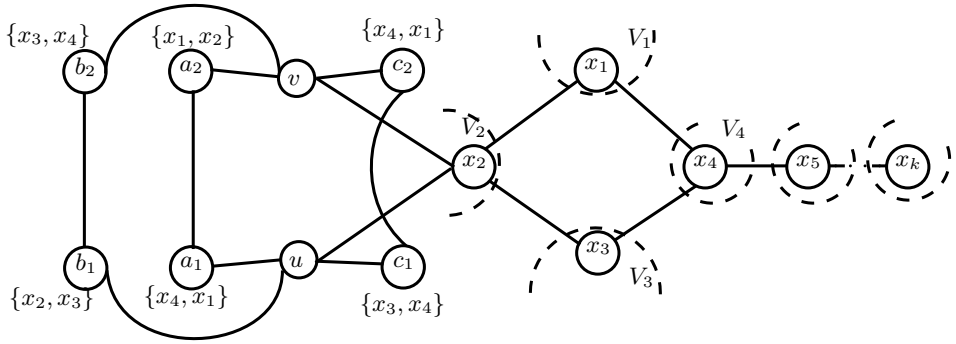

Fig. 2: Gadget used to represent an edge $\{u, v\}$ of G. Additional adjacencies are represented by vertices in braces

  - if u and v belong to V_1, then b_1 must belong to V_2 (because otherwise it would create an edge between V_1 and V_3 which is forbidden) and b_2 must belong to V_4, which would create an edge between V_2 and V_4, which is forbidden.
  - if u and v belong to V_2, then c_1 must belong to V_3, and c_2 must belong to V_1, which would create an edge between V_1 and V_3.
  - if u and v belong to V_3, then a_1 must belong to V_4, and a_2 must belong to V_2, which would create an edge between V_2 and V_4.
- Suppose now that we have a proper 3-colouring of G, and let us show that we can assign every vertex of G' to V_1, V_2, V_3 or V_4 without creating any adjacency between V_1 and V_3 nor between V_2 and V_4 (one cannot create any forbidden edge with V_j for $j > 4$ because none of the vertices of G' is adjacent to it). Let u and v be two adjacent vertices in G, and let i and j be their respective colors. We firstly assign u to V_i and v to V_j. It is easy to see that these assignments do not create any forbidden edge, and will let every vertex of the gadget to be adjacent to 2 or 3 consecutive clusters (consecutive means that if a vertex is adjacent to 2 clusters, then these two clusters cannot be (V_1, V_3) nor (V_2, V_4)). Notice that among (V_1, V_2, V_3, V_4), if a node r of the gadget is adjacent to 2 consecutive clusters then r must be assigned to one of them, if r is adjacent to 2 non-adjacent cluster, r must not be assigned to one of them (but can be assigned to one of the other two), and if r is adjacent to 3 clusters, then r must be assigned to the one which is adjacent to the other two (the one "in the middle"). This proves that if the assignments create a forbidden edge, then this edge must be between some r_1 and r_2, and not between u and r_1 nor between v and r_2, with $r \in \{a, b, c\}$. Let i and j be the respective colors of u and v (u and v are respectively in V_i and V_j, according to the previous assignments), we have the following:
  - a_1 is adjacent to V_i, V_4 and V_1 and a_2 is adjacent to V_1, V_2 and V_j. The only way to create a forbidden edge is to have $i = j = 3$, which is impossible.
  - b_1 is adjacent to V_i, V_2 and V_3 and b_2 is adjacent to V_3, V_4 and V_j. The only way to create a forbidden edge is to have $i = j = 1$, which is impossible.
  - c_1 is adjacent to V_i, V_3 and V_4 and c_2 is adjacent to V_4, V_1 and V_j. The only way to create a forbidden edge is to have $i = j = 2$, which is impossible.

  We are thus able to assign each vertex of G' to one of the k clusters without creating any forbidden edge.

□

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 [3], given a fixed pattern graph $H = (V_H, E_H)$:

> LIST-GRAPH HOMOMORPHISM TO H
> **Input:** a graph $G = (V_G, E_G)$ and for all $v \in V_G$, a list $L(v) \subseteq V_H$
> **Question:** Is there a graph homomorphism $h : V_G \to V_H$ such that for all $v \in V_G$ $h(v) \subseteq L(v)$?

Thus, U-SUM-MAX GRAPH PARTITIONING is related to a special case of L-HOMH, where all lists are equal to V_H.

In [3], 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 $O(n^{f(k)})$ 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 $\mathcal{NP}$-hard. More formally, they show that OAL-HOMH is $\mathcal{NP}$-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 [3] 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 [14], which gives a $(2 - 2/k)$-approximation algorithm for MIN-K-CUT.

[1] recall that $G = (V_G, E_G)$ is homomorphic to $H = (V_H, E_H)$ iff there is a function $h : V_G \to V_H$ such that for all $\{u, v\} \in E_G$, $\{f(u), f(v)\} \in E_H$

Algorithm 1 a greedy algorithm for SUM-MAX GRAPH PARTITIONING

```
Sort E in non decreasing order of weights (ties are broken arbitrarily)
j ← 0
for i = 1 to k − 1 do
  while G has i connected components do
    G ← G\{e_j}
    j ← 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 $\theta = \max\{\frac{w(e)}{w(e')} : e, e' \in E, e \neq e', w(e') \geq w(e)\}$. For a solution $S = \{S_1, ..., S_k\}$ of the problem, we associate the set $C_S = \{c_1, ..., c_{p_s}\}$ 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 $val(S) = \sum_{i=1}^{p_S} w(c_i)$.

Let $A = \{A_1, ..., A_k\}$ be the solution returned by Algorithm 1, and $\{{}^iA_1, ..., {}^iA_i\}$ the partial solution at the beginning of step i. The while loop consists in separating a cluster ${}^iA_t$ (for some $t \in \{1, ..., i\}$) into two clusters ${}^iA_t^1$ and ${}^iA_t^2$. Thus, when separating ${}^iA_t$, we add to C_A the edge of maximum weight between ${}^iA_t^1$ and ${}^iA_t^2$, and at most $(i-1)$ edges (called the *unexpected edges*) between ${}^iA_t^1$ or ${}^iA_t^2$ and the other clusters (cf Figure 3). We thereby add to the solution value one term w_i (between ${}^iA_t^1$ and ${}^iA_t^2$) and $(i-1)$ terms $(\alpha_i^j)_{j=1..(i-1)}$. For $j \in \{1, ..., (i-1)\}$, if the edge of maximum weight between ${}^iA_t$ and ${}^iA_j$ has one endpoint in ${}^iA_t^1$ (resp. ${}^iA_t^2$), then α_i^j is equal to the edge of maximum weight between ${}^iA_t^2$ (resp. ${}^iA_t^1$) and ${}^iA_j$, or 0 if the two clusters are not adjacent.

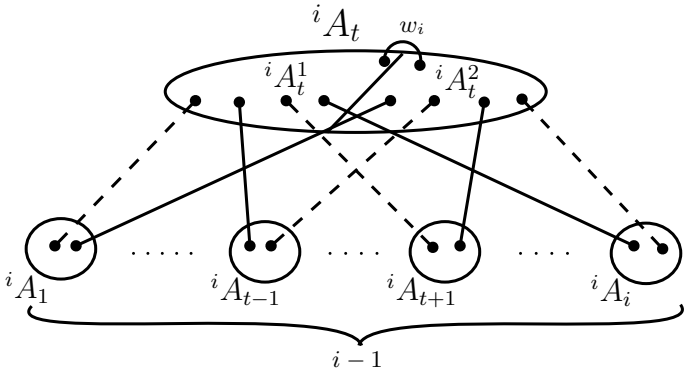

Fig. 3: Dashed lines represent edges of maximum weight between ${}^iA_t$ and other clusters, already in C_A, solid lines represent the at most $(i-1)$ new edges added to C_A.

By definition, we have

$$val(A) = \sum_{i=1}^{k-1}(w_i + \sum_{j=1}^{i-1}\alpha_i^j) \tag{1}$$

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 $\{{}^iA_1, ..., {}^iA_i\}$. Then, for any $t \in \{1, \ldots, i\}$ we can upper bound the total weights of the heaviest edges outcoming from ${}^iA_t$ in the following way*

$$\sum_{\substack{j=1 \\ j\neq t}}^{i} w(e_{t,j}) \leq \sum_{j=1}^{i-1} w_j,$$

where $e_{t,j}$ denotes the edge of maximum weight between ${}^iA_t$ and ${}^iA_j$.

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, ${}^iA_t$ has been separated into ${}^iA_t^1$ and ${}^iA_t^2$, thus incurring an additional weight of w_i.

For $j_0 \neq t$, notice that edge $e_{j_0,t}$ (edge between ${}^iA_{j_0}$ and ${}^iA_t$, before the split) is now replaced by two edges e_{j_0,t_1} and e_{j_0,t_2}, with $\max(w(e_{j_0,t_1}), w(e_{j_0,t_2})) = w(e_{j_0,t})$. Let us now bound the weight of edges out-coming from ${}^iA_{j_0}$. W.l.o.g., suppose that $w(e_{j_0,t_1}) = w(e_{j_0,t})$, and let ${}^iS_{j_0}$ be the sum of all heaviest edges linking ${}^iA_{j_0}$ to each one of the other clusters (including ${}^iA_t^1$ and ${}^iA_t^2$. Thus, we have

$$\begin{aligned} {}^iS_{j_0} &= \sum_{\substack{j=1 \\ j\neq j_0, j\neq t}}^{i} w(e_{j_0,j}) + \underbrace{w(e_{j_0,t_1})}_{w(e_{j_0,t})} + \underbrace{w(e_{j_0,t_2})}_{\leq w_i} \\ &\leq \sum_{j=1}^{i-1} w_j + w_i \quad \text{using the induction hypothesis} \end{aligned}$$

Same arguments hold for sets ${}^iA_t^1$ and ${}^iA_t^2$, which completes the proof. □

Corollary 2. *Let us consider the beginning of step i, and the corresponding i partition $\{{}^iA_1, ..., {}^iA_i\}$. When splitting ${}^iA_t$, 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 ${}^iA_j$ and ${}^iA_t$. For example, if $e_{j,t}$ was in fact an edge between ${}^iA_j$ and ${}^iA_t^1$, $\tilde{e}_{j,t}$ is the edge between ${}^iA_j$ and ${}^iA_t^2$. By definition of θ, we have $w(\tilde{e}_{j,t}) \leq \theta w(e_{j,t})$, and thus $\sum_{j=1}^{i-1} \alpha_i^j = \sum_{j=1, j\neq t}^{i} w(\tilde{e}_{j,t}) \leq \theta \sum_{j=1}^{i-1} w_j$ (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 $C_S = \{c_1, ..., c_{p_S}\}$. We have:*

$$\sum_{j=1}^{p_S} w(c_j) \geq \sum_{j=1}^{i} w_j$$

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 $C_S = \{c_1, ..., c_{p_S}\}$. Let $w_M = \max_{j=1...p_S} w(c_j)$, and let (S_{i_1}, S_{i_2}) 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:

$$\sum_{j=1}^{p_S} w(c_j) - w_M \geq \sum_{j=1}^{i-1} w_j$$

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 $(1 + (\frac{k}{2} - 1)\theta)$-approximation algorithm.*

Proof. Using Lemma 2 with an optimal solution, it is sufficient to show the following inequality:

$$val(A) \leq (1 + (\frac{k}{2} - 1)\theta) \sum_{i=1}^{k-1} w_i \tag{2}$$

Let us prove it by induction over k. Statement is clear for $k = 2$. Suppose now that the result is true for all $k = 1, 2, ..., t$ and let us show that it remains true for $k = t + 1$. By Equation (1) and the induction hypothesis, we have:

$$
\begin{aligned}
val(A) &\leq (1+(\frac{t}{2}-1)\theta)\sum_{i=1}^{t-1} w_i + w_t + \sum_{j=1}^{t-1}\alpha_t^j \\
&= (1+(\frac{t}{2}-1)\theta)\sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2}\sum_{j=1}^{t-1}\alpha_t^j + \frac{1}{2}\sum_{j=1}^{t-1}\alpha_t^j \\
&\leq (1+(\frac{t}{2}-1)\theta)\sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2}\theta\sum_{j=1}^{t-1} w_j + \frac{1}{2}\sum_{j=1}^{t-1}\alpha_t^j \quad \text{using Lemma 1} \\
&\leq (1+(\frac{t}{2}-1)\theta)\sum_{i=1}^{t-1} w_i + w_t + \frac{1}{2}\theta\sum_{j=1}^{t-1} w_j + \frac{1}{2}(t-1)\theta w_t \quad \text{as } \alpha_t^j \leq \theta w_t \\
&\leq (1+(\frac{t+1}{2}-1)\theta)\sum_{i=1}^{t-1} w_i + w_t + (\frac{t+1}{2}-1)\theta w_t \\
&\leq (1+(\frac{t+1}{2}-1)\theta)\sum_{i=1}^{t} w_i
\end{aligned}
$$

Which gives the desired inequality. □

Thus, Algorithm 1 becomes arbitrarily good as θ 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 $\mathcal{NP}$-hard even if all edge weights are different (and thus even when θ 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 ($\sum(w_j)$) 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 $C = \{c_1, ..., c_k\}$ and $S = \{s_1, ..., s_k\}$. Finally, $w(e) = 1$ for all $e \in E$. Let us now define three categories of edges:

- first category: $X = \{\{c_i, s_j\}$ such that $i \neq j$ or $j = 1\}$,
- second category: $Y = \{\{c_i, c_j\}$ such that $i \neq j\}$,
- third category: $Z = \{\{c_i, s_j\}$ such that $i = j$ and $j \neq 1\}$.

An example of such a graph is presented in Figure 4.

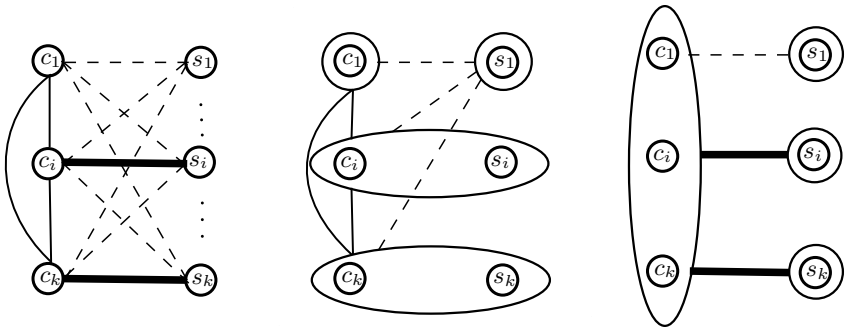

Fig. 4: (a): Example of a graph that reaches the ratio. First category of edges is represented with dashed lines, second category with solid lines, third category with bold lines (b): Solution given by Algorithm 1 (c): Optimal solution

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 $(V_1, ..., V_k)$:

- for all $j \in \{1, ..., k\}$, $V_j = \{s_j\}$
- $V_{k+1} = C$

The value of this solution is k, (it is thus an optimal one). Then, notice that $\theta = \max\{\frac{w(e)}{w(e')} : e, e' \in E, e \neq e', w(e') \geq w(e)\} = 1$. Let $\mathcal{A}(I_k)$ and $\mathcal{OPT}(I_k)$ denote respectively the value of the solution given by Algorithm 1 and the value of an optimal solution for I_k. We have $\frac{\mathcal{A}(I_k)}{\mathcal{OPT}(I_k)} = \frac{k+1}{2}$, which proves the result (we are looking for a $(k+1)$-partition).

□

Remark 1. 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 $|E|$.

Remark 2. It is easy to see that the differential ratio [1] of the instances built previously is 0. Thus, the differential ratio of Algorithm 1 is 0.

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 $\mathcal{NP}$-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 $O(n^{f(k)})$ algorithm (like for MIN-K-CUT [7]), or getting an $\mathcal{NP}$-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. M. Demange and V. Th. Paschos. On an approximation measure founded on the links between optimization and polynomial approximation theory. *Theoretical Computer Science*, 158(1–2):117 – 141, 1996.
2. 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 205–218. Elsevier Science Publishers, 2003.
3. T. Feder and P. Hell. List homomorphisms to reflexive graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236 – 250, 1998.
4. 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.
5. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
7. 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.
8. 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.
9. M. Hansen, P.and Delattre. Complete-link cluster analysis by graph coloring. *Journal of the American Statistical Association*, 73(362):pp. 397–403, 1978.
10. P. Hell. *Graphs and Homomorphisms*. Oxford University Press, 2004.
11. M. Koivisto. An $O(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In *in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), IEEE*, pages 583–590, 2006.
12. 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.
13. 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.
14. H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, February 1995.
15. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
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.