Sum-Max Graph Partitioning Problem

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

On The Sum-Max Graph Partitioning Problem[☆]

Rémi Watrigant[a], Marin Bougeret[a], Rodolphe Giroudeau[a], Jean-Claude König[a]

[a] *LIRMM*
Université de Montpellier II
UMR 5506 CNRS
161 rue Ada
34392 Montpellier Cedex 5, France

Abstract

This paper tackles the following problem: given a connected graph $G = (V, E)$ with a weight function on its edges and an integer $k \leq |V|$, find a partition of V into k clusters such that the sum (over all pairs of clusters) of the heaviest edges between the clusters is minimized. We first prove that this problem (and even the unweighted variant) cannot be approximated within a factor of $O(n^{1-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$, and cannot admit an FPT algorithm unless $FPT = W[1]$. Next, we develop several algorithms: we first present a greedy algorithm and prove that it achieves a tight ratio smaller than $\frac{k}{2}$. Concerning the unweighted version of the problem, we study how the diameter of the input graph can influence its complexity, by obtaining negative results for graphs of low diameter, positive results for graphs of high diameter, and a polynomial-time approximation algorithm with an approximation ratio smaller than $\frac{k}{2}$ and depending on the diameter. We also highlight a link between the k-SPARSEST SUBGRAPH problem and the unweighted version. This allows us to develop several constant ratio approximation algorithms running in exponential time for general graphs, and in polynomial time in some restricted graph classes. Finally, we discuss the exact resolution for fixed k and the connections to a graph homomorphism problem.

Keywords: graph partitioning problem, approximation, combinatorial optimization

1. Introduction

1.1. Definition of the Problem

Graph partitioning problems are classical combinatorial optimization problems, and are the core of many practical issues [1]. They commonly consist in

[☆] This work has been funded by grant ANR 2010 BLAN 021902

Email addresses: watrigant@lirmm.fr (Rémi Watrigant), bougeret@lirmm.fr (Marin Bougeret), rgirou@lirmm.fr (Rodolphe Giroudeau), konig@lirmm.fr (Jean-Claude König)

finding a partition of vertices of a given graph into subsets (or clusters) according to different constraints and objective functions depending on the application. In this paper, we focus on minimizing the sum, over all pair of clusters, of the heaviest edges between the clusters. More formally, we study the following optimization problem:

SUM-MAX GRAPH PARTITIONING (SM-GP)
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 $V_i \neq \emptyset$ for all $i \in \{1, ..., k\}$
Goal: Minimize $cost(V_1, ..., V_k) = \sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

We denote by U-SUM-MAX GRAPH PARTITIONING (U-SM-GP) the unweighted version of the problem, where $w(e) = 1$ $\forall e \in E$. The threshold value for the associated decision version of the problem will be denoted by C. All graphs studied here are supposed to be simple, non oriented and connected, unless otherwise stated. Thus, it is easy to remark that any k-partition has a cost of at least $(k-1)\min_{e \in E} w(e)$.

1.2. Applications and Related Work

Let us now point out some interesting links between SM-GP and several classical combinatorial problems:

- Firstly, one can mention all graph partitioning problems using different objective functions. Among them, one may want to minimize the sum of *all* edge weights between each pair of clusters (instead of the heaviest only), leading to the classical MIN-K-CUT problem which has been extensively studied on the side of complexity [2], exact algorithms [3], approximability [4] or parameterized complexity [5]. Another objective function is the optimization of the sum of the edge weights (or the heaviest one) inside each cluster [6]. Finally, one can combine these objectives and minimize the cut ratio of the partition [7]. 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, [9] 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 [10] defines the M-PARTITIONING problem where the objective is to find a partition of the vertices respecting some constraints defined by a matrix M.

- Secondly, the unweighted version of our graph partitioning problem can be viewed as a graph homomorphism problem[1] [11]. Indeed, we are looking

[1] Recall that an homomorphism from a graph $G = (V_G, E_G)$ to a graph $H = (V_H, E_H)$ is a function $h : V_G \to V_H$ such that $\{x, y\} \in E_G$ implies $\{h(x), h(y)\} \in E_H$.

for an homomorphism from G to a target graph with k vertices and as few edges as possible, but with the additional constraints that (a) the homomorphism must be edge-surjective and (b) the target graph must be reflexive, *i.e.* contains a loop on every vertex. Several papers of N. Vikas [12, 13, 14, 15] study the problem of deciding whether a graph G admits such an homomorphism (called *compaction*) to a fixed graph H, by restricting G or H to special graph classes in order to obtain polynomial or $\mathcal{NP}$-hardness results. In Section 6, we describe how these results can help in the design of exact algorithms for U-SM-GP.

1.3. Contributions and Organization of the Paper

In this paper we describe both negative and positive results for SUM-MAX GRAPH PARTITIONING and its unweighted variant:

- On the negative side, we show in Section 2 that when k is part of the input, the problem, and its unweighted variant are $\mathcal{O}(n^{1-\epsilon})$ inapproximable unless $\mathcal{P} = \mathcal{NP}$ for all $0 < \epsilon \leq 1$, and $W[1]$-hard for the parameter k.

Then, on the positive side, we present several algorithms together with their approximation analysis:

- First, we present in Section 3 a natural greedy algorithm and show that its approximation ratio for SM-GP is better than $k/2$. Moreover, we show that the analysis is tight.

- Then in Section 4, for the unweighted version U-SM-GP, we show how the diameter of the input graph can influence its complexity, by obtaining negative results for graphs of low diameter, positive results for graphs of high diameter, and a polynomial-time $(\frac{k}{2} - d + 2 + \frac{d^2-3d}{2(k-1)})$-approximation algorithm where d is the diameter of the input graph.

- Also for the unweighted version, we present in Section 5 a link with the k-SPARSEST SUBGRAPH problem and a general algorithm which provides $(2-2/k)$-approximated solutions and runs in $\mathcal{O}(k^2 n^{\omega \frac{k-1}{3}})$ time for general graphs (where $\omega < 2.376$ is the matrix multiplication exponent), in polynomial time for split graphs and in $\mathcal{FPT}$ time in interval graphs, and which provides $(2-2/k+\frac{1}{\epsilon})$-approximated solution in proper interval graphs in polynomial time for every fixed $\epsilon > 0$. Still using the link with k-SPARSEST SUBGRAPH, we provide a polynomial de-randomized approximation algorithm achieving a ratio of $(1+\frac{m(k-2)}{n(n-1)})$.

- Finally, in Section 6 we present a method to solve the weighted problem for $k = 3$ and the unweighted problem for $k = 4$, and we use a negative result of the compaction problem presented above to suggest that this method may be hard to generalize for all fixed k. We also discuss some open problems.

1.4. Notations

Recall that all graphs studied here are supposed to be simple, non oriented and connected, unless otherwise stated. Given a graph $G = (V, E)$ and $w : E \to \mathbb{N}$, we define as usually $n = |V|$ and $m = |E|$. A vertex $\omega \in V$ is called a *universal vertex* of G if $\{\omega, v\} \in E$ for all $v \neq \omega$. We denote by $\alpha(G)$ the size of a maximum independent set of G (*i.e.* a set of pairwise non-adjacent vertices). The eccentricity of a vertex is the greatest distance between this vertex and any other vertex (in terms of number of edges of a shortest path connecting these vertices). The diameter of G is the maximum eccentricity over all vertices, and will be denoted by $diam(G)$. Given a partition $P = \{V_1, ..., V_k\}$ of V, we define $cost(V_1, ..., V_k) = \sum_{i=1}^{k} \sum_{j=i+1}^{k} \max_{u \in V_i, v \in V_j} w(u, v)$. Moreover, we note $G_{/P}$ the quotient graph obtained by the partition, *i.e.* the graph with $\{V_1, ..., V_k\}$ as vertex set, and with an edge $\{V_i, V_j\}$ of cost $\max_{u \in V_i, v \in V_j} w(u, v)$ if there exists in G an edge between a vertex of V_i and a vertex of V_j. For other definitions of graph theory, approximation and parameterized complexity, we refer the reader to the classical literature.

2. Computational Lower Bounds

In this section, we provide a reduction which highlights the intractability of SUM-MAX GRAPH PARTITIONING (and even the unweighted version), both from approximation and parameterized complexity. Indeed, we show that the problem is $W[1]$-hard for the parameter k, and that unless $\mathcal{P} = \mathcal{NP}$, we cannot expect a $\mathcal{O}(n^{1-\epsilon})$-approximation algorithm for any fixed $0 < \epsilon \leq 1$.

Theorem 1. U-SUM-MAX GRAPH PARTITIONING *(and thus the weighted version) is* $W[1]$*-hard for the parameter* k*, and cannot be approximated within a* $\mathcal{O}(n^{1-\epsilon})$ *factor for any fixed* $0 < \epsilon \leq 1$*, unless* $\mathcal{P} = \mathcal{NP}$*.*

PROOF. We show a gap preserving reduction from INDEPENDENT SET to U-SM-GP. It is known [16] that for any constant $r \leq 1$, the following problem is $\mathcal{NP}$-complete:

GAP$_r$ INDEPENDENT SET
Input: a graph $G = (V, E)$, $k \in \mathbb{N}$
Output: Decide whether $\alpha(G) \geq k$ or $\alpha(G) \leq rk$.

Let $G = (V, E)$ be a graph, and $k \in \mathbb{N}$. We construct an instance Φ' of U-SUM-MAX GRAPH PARTITIONING, composed of a graph $G' = (V', E')$, together with $r' \geq 1$ and $k' \in \mathbb{N}$, such that:

- $\alpha(G) \geq k \Rightarrow \mathcal{OPT}(\Phi') \leq k'$
- $\alpha(G) \leq rk \Rightarrow \mathcal{OPT}(\Phi') \geq r'k'$
- $r' = \mathcal{O}(\frac{1}{r})$

Where $\mathcal{OPT}(\Phi')$ is the cost of an optimal solution of U-SM-GP for Φ'.

The construction is as follows: $G' = (V', E')$ is a copy of G, plus a universal vertex ω. We define $k' = k + 1$ and $r' = \frac{1}{2r}$. Notice that this reduction can clearly be performed in polynomial time.
Suppose that $\alpha(G) \geq k$ and 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 $\{V_1, ..., V_{k'}\}$ of V': $V_i = \{s_i\}$ for all $i \in \{1, ..., k\}$, and $V_{k+1} = V' \backslash S$. By construction, it is clear that $cost(V_1, ..., V_{k'}) = k$, and thus $\mathcal{OPT}(\Phi') \leq k'$.
Suppose now $\alpha(G) \leq rk$ and let $\{V_1, ..., V_{k'}\}$ be any k'-partition of G'. Because ω is an universal vertex, it is clear that the cost of this solution will be at least k, plus the minimum number of edges of any graph with k nodes that does not contain an independent set of size rk, *i.e.* $\mathcal{OPT}(\Phi') \geq k + x$, where $x = \min\{|E_H| : H = (V_H, E_H)$ such that $|V_H| = k$ and $\alpha(H) \leq rk\}$. Then, since every graph H on n_H vertices and m_H edges contains an independent set of size at least $\frac{n_H}{2m_H + n_H}$, we have:

$$rk \geq \alpha(H) \geq \frac{k}{\frac{2x}{k+1}}$$

Which gives the following bound (we can suppose that $rk \geq 1$):

$$\mathcal{OPT}(\Phi') \geq k + \frac{k^2}{2rk} - \frac{k}{2} \geq \frac{k+1}{2r} = r'k'$$

Thus, we have a gap of $r' = \frac{1}{2r}$, where r is arbitrary small.
Finally, notice that the parameter k from INDEPENDENT SET is preserved through the reduction ($k' = k + 1$). Since a gap preserving reduction is a special case of polynomial reduction, and since INDEPENDENT SET is a well known $W[1]$-hard problem, this reduction also proves the $W[1]$-hardness of our problem.

3. Approximation Algorithm for the General Case

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 $\mathcal{O}(|E| \log |E|)$. Actually, this algorithm corresponds to the SPLIT algorithm of [4], which gives $(2-2/k)$-approximated solutions for the MIN-K-CUT problem.

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
    $G \leftarrow G \backslash \{e_j\}$
    $j \leftarrow j + 1$
  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

3.2.1. 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 $cost(S) = \sum_{i=1}^{p_S} w(c_i)$.

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 1). 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.

By definition, we have

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

3.2.2. Preliminaries

Let us now state several properties of the algorithm that will be the base of the approximation result (Theorem 2). 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*

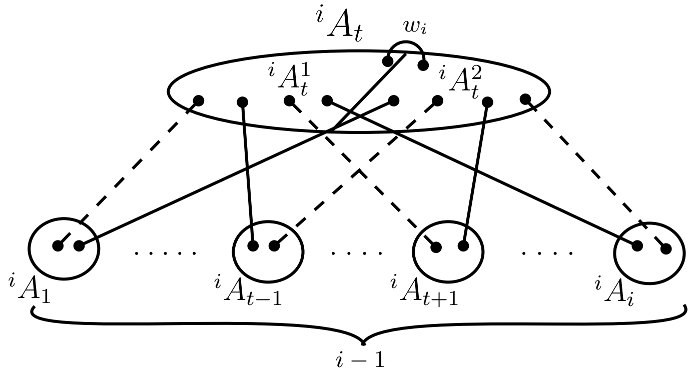


Figure 1: Dashed lines represent edges of maximum weight between ${}^iA_t$ and other clusters, already in C_A, solid lines represent the at most $(i-1)$ new edges added to C_A.

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 1. *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.

3.2.3. 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 2. *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:

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

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:

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

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 ($\sum_{i=1}^{t}(w_j)$) becomes tight.

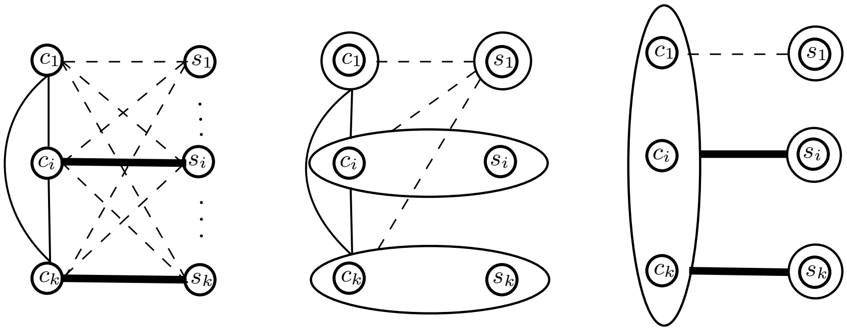
Proposition 1. *Approximation ratio of Algorithm 1 is tight.*

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 2.



(a) (b) (c)

Figure 2: (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 $j \in \{1, ..., k\}$, $V_j = \{s_j\}$, and $V_{k+1} = C$. 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 $|E|$.*

4. Using the Diameter of the Graph for the Unweighted Case

In this section we study how the diameter of the input graph influences the complexity of U-sum-max graph partitioning.

4.1. Analysis for Extremal Values of the Diameter

Here we analyse the complexity of the problem for extremal values of the diameter of the input graph. First, notice that the graph obtained in the reduction of Theorem 1 is of diameter 2 (because of the universal vertex). By slightly modifying the reduction, we can obtain the same result for unweighted graphs of diameter 3, and weighted graphs of all fixed diameter $4 \leq d \leq k$, as shown in the following proposition:

Proposition 2. U-SM-GP *is* $\mathcal{NP}$*-hard in graphs of diameter* 3*.* SM-GP *is* $\mathcal{NP}$*-hard in graphs of diameter* d *for all fixed* $d = 4, ..., k$ *(even if edge weights are defined over two distinct values).*

Proof. We only sketch the proof of the second statement, as it uses the same arguments of the proof of Theorem 1. This first statement will then be deduced as a special case.

Let $G = (V, E)$ be a graph, and $k \leq |V|$. Let $\delta \leq k$. We construct the graph G' as a disjoint union of δ copies of G: $(G_1, \ldots, G_\delta)$ where $G_i = (V^i, E^i)$, with $V^i = \{v_1^i, ..., v_n^i\}$ for all $i \in \{1, \ldots, \delta\}$, and we set $w(e) = 1$ for all $e \in \bigcup_{i=1}^{\delta} E^i$. Moreover, for all $i \in \{1, ..., (\delta - 1)\}$ and all $j \in \{1, ..., n\}$, we make v_j^i be adjacent to v_j^{i+1} with an edge of weight $M > k$. Then, for all $i \in \{1, \ldots, \delta\}$, we add a vertex α_i which is adjacent to every vertex of G_i, with edges of weight 1. Finally, we make α_i be adjacent to α_{i+1} for all $i \in \{1, \ldots, (\delta - 1)\}$ with edges of weight M. It is clear that this construction can be done in polynomial time (notice that $\delta \leq k \leq |V|$), and that the diameter of the output graph is $d = \delta + 1$.

Suppose that G contains an independent set S of size k, with $S = \{s_1, ..., s_k\}$. For $i \in \{1, \ldots, \delta\}$, we define the corresponding independent set $S^i = \{s_1^i, \ldots, s_k^i\}$ in G'. We build the the following $(k+1)$-partition $P = \{P_1, \ldots, P_{k+1}\}$ of G' as follows:

- $P_i = \{s_j^i : j \in \{1, ..., \delta\}\}$ for all $i \in \{1, ..., k\}$
- $P_{k+1} = V' \backslash \left(\bigcup_{i=1}^{k} P_i \right)$

By construction this is a $(k+1)$-partition of cost k. Conversely, because of the edges of weight M, for all $j \in \{1, ..., n\}$, all $\{v_j^i : i \in \{1, ..., (\delta - 1)\}\}$ must be in the same cluster, as well as all $\{\alpha_i : i \in \{1, ..., (\delta - 1)\}\}$. Thus, it is easily seen that that if G' contains a $(k+1)$-partition of cost k, then G admits an independent set of size k.

Moreover, for $\delta = 2$ (*i.e.* which leads to a graph of diameter 3), we can set $M = 1$ and obtain the result for the unweighted problem. Notice that the equivalence does not hold anymore for $\delta = 3$ when $M = 1$, since there is no edge between G_1 and G_3 for instance, and hence we can no longer control the structure of the solution (adding complete graphs between graphs G_i would decrease the diameter of G').

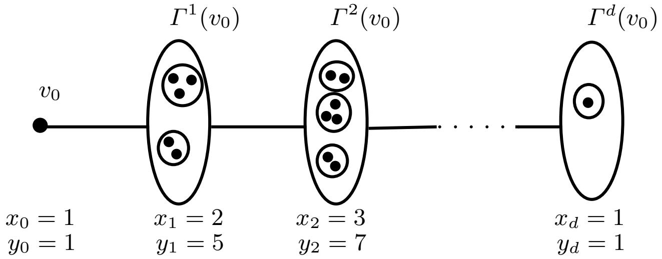


Figure 3: Illustration of Algorithm EXPAND-AND-SPLIT

Whereas the problem is hard for small values of diameter, we observe that it is polynomial solvable on graphs of high diameter:

Proposition 3. U-SM-GP *is polynomial solvable on graphs of diameter $d \geq k-1$.*

PROOF. Suppose that G has diameter $d \geq k-1$, and let v_0 be a vertex of maximum eccentricity in G. We define the following $(d+1)$-partition:

$$\Gamma^i(v_0) = \begin{cases} \{v_0\} & \text{if} \quad i = 0 \\ N(v_0) & \text{if} \quad i = 1 \\ \{N(v) : v \in \Gamma^{i-1}(v_0)\} \backslash \Gamma^{i-2}(v_0) & \text{if} \quad 1 < i \leq d \end{cases}$$

Then, we output the solution $P = \{\Gamma^0(v_0), ..., \Gamma^{k-2}(v_0), \bigcup_{i=k-1}^{d} \Gamma^i(v_0)\}$. By construction, $\Gamma^i(v_0) \cap \Gamma^j(v_0) = \emptyset$ for all $i \neq j$, and we thus have a k-partition of V of minimum cost $(k-1)$, since its quotient graph is a path of length $(k-1)$.

We now turn to the definition and analysis of an algorithm using the diameter the graph.

4.2. Presentation of Algorithm EXPAND-AND-SPLIT

Let $G = (V, E)$ be a graph and $k \leq |V|$. The idea of the following algorithm is to construct the same partition as in the case of $d \geq k-1$, using a vertex of maximum eccentricity, and then to modify it in order to obtain exactly k clusters.

Let v_0 be a vertex of maximum eccentricity in G, and let us consider the $(d+1)$-partition $\{\Gamma^0(v_0), ..., \Gamma^d(v_0)\}$ as defined in Proposition 3. For all $i \in \{0, ..., d\}$, let $y_i = |\Gamma^i(v_0)|$. Since the graph has diameter $d < k-1$, the EXPAND-AND-SPLIT algorithm arbitrarily splits each $\Gamma^i(v_0)$ into x_i clusters, with $\sum_{i=0}^{d} x_i = k$. An example of such a partition is depicted in Figure 3.

4.3. Analysis of the algorithm

Theorem 3. EXPAND-AND-SPLIT *is a polynomial* $(\frac{k}{2}-d+2+\frac{d^2-3d}{2(k-1)})$*-approximation algorithm, where* d *denotes the diameter of the graph.*

PROOF. It is clear that the algorithm runs in polynomial time. Let $\mathcal{A}$ denote the value of the solution returned by EXPAND-AND-SPLIT. We have:

$$\begin{aligned}
2\mathcal{A} &\leq \overbrace{\sum_{i=0}^{d} x_i(x_i-1)}^{\text{edges between clusters in } \Gamma^i(v_0)} + \overbrace{\sum_{i=0}^{d-1} 2x_i x_{i+1}}^{\text{edges between clusters of } \Gamma^i(v_0) \text{ and } \Gamma^{i+1}(v_0)} \\
&= \sum_{i=0}^{d} x_i^2 - \sum_{i=0}^{d} x_i + \sum_{i=0}^{d-1} 2x_i x_{i+1} \\
&= \sum_{i=0}^{d} x_i^2 - k + \sum_{i=0}^{d} 2x_i x_{i+1} - 2x_0 x_d \\
&= \left(\sum_{i=0}^{d} x_i\right)^2 - \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j - k - 2x_d \\
&\leq \left(\sum_{i=0}^{d} x_i\right)^2 - \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j - k \\
&= k^2 - k - \Delta_d \qquad \text{with } \Delta_d = \sum_{\substack{0\leq i,j\leq d \\ |i-j|\geq 2}} x_i x_j \qquad (3)
\end{aligned}$$

Lemma 3. *Let* $\{x_0, ..., x_d\} \in \mathbb{N}^{d+1}$ *such that* $\sum_{i=0}^{d} = k$ *and* $x_i > 0$ *for* $i \in \{0, ..., d\}$*,* Δ_d *is minimum if there exists* $i_0 \in \{1, ..., d-1\}$ *such that* $x_{i_0} = k - d$*, and* $x_i = 1$ *for all* $i \neq i_0$*.*

PROOF. Given $d > 0$, the goal is to find $\{x_0, ..., x_d\}$ with $x_i \in \mathbb{N}$ for all $0 \leq i \leq d$ such that

$$z = \sum_{\substack{0\leq i,j\leq d \\ i-j\geq 2}} x_i x_j$$

is minimized, under the the constraints:

- $\sum_{i=0}^{d} x_i = k$
- $x_i > 0 \ \forall i \in \{0, ..., d\}$

Notice that $2z = \Delta_d$ (because of the sum over i and j such that $j - i \geq 2$ instead of $|j - i| \geq 2$). Let us denote by z^* the value of an optimal solution. If $d = 1$ then the result is obvious ($z^* = 0$). If $d = 1$, then $z^* = 1$ for $x_0 = 0$,

$x_1 = k-1$ and $x_2 = 1$. Thus, in the following, we consider the case $d \geq 2$. For sake of readability, we define $x_{-1} = x_{d+1} = 0$. First, let us remark that $z = \sum_{j-i\geq 2} x_i x_j = \frac{1}{2}\sum_{i=0}^{d}(k-(x_{i-1}+x_i+x_{j+1}))$. Thus, minimizing z is the same as maximizing

$$t = \sum_{i=0}^{d} x_i(x_{i-1}+x_i+x_{i+1})$$

under the same constraints. Here again we denote by t^* the value of an optimal solution. For all $i \in \{1, ..., d\}$, we define $\alpha_i = x_{i-1}+x_i+x_{i+1}$. Thus, we have $t = \sum_{i=0}^{d} x_i\alpha_i$. The proof is divided in three steps:

- Step 1: There exists an optimal solution such that $x_0 = x_d = 1$
- Step 2: There exists an optimal solution such that:
  - $\exists j \in \{1, ..., d-1\}$ such that $\alpha_j = k-d+2$
  - $\forall i \in \{1, ..., d\}$ with $|j-i| \geq 2$ we have $x_i = 1$
- Step 3: There exists an optimal solution such that:
  - $\exists j \in \{1, ..., d-1\}$ such that $x_j = k-d$
  - $\forall i \neq j$ we have $x_i = 1$

Proof of Step 1:
Let $\{x_0, ..., x_d\}$ be an optimal solution. Without loss of generality, suppose by contradiction that $x_0 > 1$, and let us define the following solution $\{x'_0, ..., x'_d\}$:

- $x'_0 = x_0 - 1$
- $x'_1 = x_1 + 1$
- $x'_i = x_i$ for all $1 \leq i \leq d-1$

With the associated $\alpha'_i = x'_{i-1}+x'_i+x'_{i+1}$ for all $i \in \{0, ..., d\}$. We have:

$$\begin{aligned}\sum_{i=0}^{d} x'_i\alpha'_i &= (x_0-1)\alpha_0 + (x_1+1)\alpha_1 + x_2\alpha_2 + \sum_{i=3}^{d} x_i\alpha_i \\ &= \sum_{i=0}^{d} x_i\alpha_i + x_2\end{aligned}$$

Since $x_2 > 0$, it implies that $\{x_0, ..., x_d\}$ is not an optimal solution. Contradiction.

Proof of Step 2:
Let $\{x_0, ..., x_d\}$ be an optimal solution, with $x_0 = x_d = 1$, and let $j \in \{0, ..., d\}$ such that $\alpha_j = \max_{k=2...d} \alpha_k$ (remark that $j \neq 0$ and $j \neq d$). By contradiction, suppose that there exists $i \notin \{j-1, j, j+1\}$ such that $x_i > 1$. Let us define the following solution $\{x'_0, ..., x'_d\}$:

- $x'_i = x_i - 1$
- $x'_j = x_j + 1$
- $\forall k \in \{i, j\}\ x'_k = x_k$

With the associated $\alpha'_i = x'_{i-1} + x'_i + x'_{i+1}$ for all $i \in \{0, ..., d\}$. Here we distinguish 2 cases:
First case: $i + 1 = j - 1$ (the case $i - 1 = j + 1$ is symmetrical). We have:

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k + 2$$

Second case: $i \notin \{j-2, j-1, j, j+1, j+2\}$. We have:

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k + 2\alpha_j - 2\alpha_i + 2$$

With $2\alpha_j - 2\alpha_i + 2 > 0$. In both cases, it implies that $\{x_0, ..., x_d\}$ is not an optimal solution. Contradiction.

Proof of Step 3:
Let $\{x_0, ..., x_d\}$ be an optimal solution with:

- $x_{j-1} + x_j + x_{j+1} = k - d + 2$ for some $j \in \{1, ..., d-1\}$
- $x_i = 1$ for all $i \neq \{j-1, j, j+1\}$

And suppose that $x_{j-1} > 1$. Let us define the following solution $\{x'_0, ..., x'_d\}$:

- $x'_{j-1} = x_{j-1} - 1$
- $x_j = x_j + 1$
- for all $k \notin \{j-1, j\}\ x'_k = x_k$

With the associated $\alpha'_i = x'_{i-1} + x'_i + x'_{i+1}$ for all $i \in \{0, ..., d\}$. We have:

$$\sum_{k=0}^{d} x'_k \alpha'_k = \sum_{k=0}^{d} x_k \alpha_k - \alpha_{j-1} + \alpha_j + x_{j+1} - x_{j-2}$$

And since $-\alpha_{j-1} + \alpha_j + x_{j+1} - x_{j-2} \geq 0$, $\{x'_0, ..., x'_d\}$ is also an optimal solution, which proves the result.

By Lemma 3, we have:

$$\Delta_d \geq 2\left(k(d-2) - \frac{d(d-1)}{2} + 2\right) \tag{4}$$

Using (3) and (4), we obtain:

$$\begin{aligned} 2\mathcal{A} &\leq k^2 - k - 2k(d-2) + d(d-1) - 4 \\ &\leq k^2 - 2kd + 3k + d^2 - d - 4 \end{aligned}$$

Let $\mathcal{OPT}$ denote the value of an optimal solution for the given instance. Using $\mathcal{OPT} \geq k - 1$ (since G is connected), we get:

$$\begin{aligned} \rho = \frac{2\mathcal{A}}{2\mathcal{OPT}} &\leq \frac{k^2 - 2kd + 3k + d^2 - d - 4}{2(k-1)} \\ &\leq \frac{k}{2} - d + 2 + \frac{d^2 - 3d}{2(k-1)} \end{aligned}$$

Which proves the result.

5. Unbalanced Partitions

5.1. Definition and link with the k-SPARSEST SUBGRAPH problem

In this section, we study how structural properties of some very simple partitions, called *unbalanced partitions*, may help in the design of approximation algorithms for the unweighted version of our problem. More precisely, we first show that any optimal solution P of U-SM-GP can be restructured into an unbalanced solution P' such that $cost(P') \leq (2 - \frac{2}{k})cost(P)$. Then, we link the simple structure of these solutions to the problem of finding in a graph k vertices which induce as few edges as possible (namely the k-SPARSEST SUBGRAPH problem). Combining these two remarks, we show that a ρ-approximated solution for k-SPARSEST SUBGRAPH gives a $(\rho + 1 - \frac{2\rho}{k})$-approximated solution for U-SM-GP. To apply this result, we provide for k-SPARSEST SUBGRAPH an $\mathcal{O}(k^2 n^{\omega\frac{k-1}{3}})$ algorithm on general graphs (where $\omega < 2.376$ is the matrix multiplication exponent), and use three known results on some restricted graph classes. Finally, we present a natural algorithm which consists in building an unbalanced solution at random. We show that we can de-randomize the algorithm to obtain a polynomial-time $(1 + \frac{m(k-2)}{n(n-1)})$-approximation algorithm. As we could expect, this algorithm gives good solutions when the input graph has low density.

Definition 1. *Given $G = (V, E)$ and $k \in \mathbb{N}$, a k-partition of V is called an unbalanced partition (or unbalanced solution) if $(k-1)$ parts are reduced to a singleton.*

Proposition 4. *Let $G = (V, E)$ and $k \leq |V|$. Let P^* be a solution of* U-SM-GP *for (G, k). Then we can construct in polynomial time an unbalanced solution P such that $cost(P) \leq (2 - \frac{2}{k})cost(P^*)$.*

PROOF. Let G, k and P^* as in the statement, and note $P^* = \{V_1^*, ..., V_k^*\}$. Without loss of generality, suppose that V_k^* is a vertex of maximum degree in $G_{/P^*}$. For all $1 \leq i \leq (k-1)$, let x_i be any vertex of V_i^*, and let us build the following unbalanced k-partition $P = \{V_1, ..., V_k\}$:

- $V_i = \{x_i\}$ for all $1 \leq i \leq (k-1)$
- $V_k = V \setminus \left(\bigcup_{i=1}^{k-1} V_i \right)$

Let q be the number of edges in $G_{/P}$ between V_k and other vertices, and q^* be the number of edges in $G_{/P^*}$ between V_k^* and other vertices. We have the following:

$$cost(P) \leq cost(P^*) + q - q^*$$

And by definition, $q \leq k-1 \leq cost(P^*)$ and $kq^* \geq 2cost(P^*)$ (since q^* is equal to the maximum degree of $G_{/P^*}$ which is a graph with k vertices and $cost(P^*)$ edges), hence:

$$\begin{aligned} cost(P) &\leq cost(P^*) + cost(P^*) - \frac{2cost(P^*)}{k} \\ &= (2 - \frac{2}{k})cost(P^*) \end{aligned}$$

Which concludes the proof.

Thus, by the previous result, enumerating all $(k-1)$-tuples of vertices allows us to build a $(2-\frac{2}{k})$-approximated solution. In fact, one can observe that we can focus on a subset of $(k-1)$ vertices which induce as few edges as possible. This problem is more generally called the k-SPARSEST SUBGRAPH problem. In the following, we show a any approximation algorithm for k-SPARSEST SUBGRAPH can be transferred to U-SM-GP.

Proposition 5. *Any ρ-approximated solution for $(k-1)$-SPARSEST SUBGRAPH gives a $(\rho + 1 - \frac{2\rho}{k})$-approximated solution for U-SM-GP.*

PROOF. Let $G = (V, E)$ and $k \leq |V|$. Let $S = \{x_1, ..., x_{k-1}\}$ be a ρ-approximated solution for $(k-1)$-SPARSEST SUBGRAPH, and $P^* = \{V_1^*, ..., V_k^*\}$ be an optimal solution for U-SM-GP on (G, k) (as previously we suppose w.l.o.g. that V_k^* is a vertex of maximum degree in $G_{/P^*}$). We construct the following k-partition $P = \{V_1, ..., V_k\}$:

- $V_i = \{x_i\}$ for all $1 \leq i \leq (k-1)$
- $V_k = V \setminus \left(\bigcup_{i=1}^{k-1} V_i \right)$

Let q (resp. q^*) be the number of edges in $G_{/P}$ (resp. in $G_{/P^*}$) between V_k (resp. V_k^*) and other vertices, and let α (resp. α^*) be the number of edges in $G_{/P}$ (resp. in $G_{/P^*}$) of the subgraph induced by $\{V_1, ..., V_{k-1}\}$ (resp. $\{V_1^*, ..., V_{k-1}^*\}$).

By definition, we have $cost(P) = q + \alpha$ and $cost(P^*) = q^* + \alpha^*$. As S is a ρ-approximated solution, we have $\alpha \leq \rho\alpha^*$. Thus, we have

$$\begin{aligned} cost(P) = q + \alpha + q^* - q^* &\leq q + \rho\alpha^* + q^* - q^* \\ &= q + \rho(\alpha^* + q^*) - \rho q^* \\ &= q + \rho cost(P^*) - \rho q^* \end{aligned}$$

Then, by definition we have $q \leq k - 1 \leq cost(P^*)$ and $kq^* \geq 2cost(P^*)$ (as previously). Hence:

$$cost(P) \leq cost(P^*)(\rho + 1 - \frac{2\rho}{k})$$

Which concludes the proof.

5.2. Applications to U-SUM-MAX GRAPH PARTITIONING

In the following, we show how to solve k-SPARSEST SUBGRAPH in $\mathcal{O}(k^2 n^{\omega\frac{k-1}{3}})$ time, where $\omega < 2.376$ [17] is the matrix multiplication exponent. In addition, we present three results for the same problem on some special graph classes.

Proposition 6. *k-SPARSEST SUBGRAPH can be solved in $\mathcal{O}(k^2 n^{\omega\frac{k}{3}})$ time, where $\omega < 2.376$ is the matrix multiplication exponent.*

PROOF. We only sketch the proof, as it is similar to the approach of [18] for the k-CLIQUE problem. Let $G = (V, E)$ and $k \in \mathbb{N}$. The algorithm consists in creating a graph whose vertices correspond to all $\frac{k}{3}$-tuples of vertices of G, and linking two vertices if the two corresponding tuples do not share any vertex. Additionally, each vertex receives a weight corresponding to the number of edges in the subgraph induced by vertices of the corresponding tuple in G, and each edge receives a weight corresponding to the number of edges in G between vertices of the two corresponding tuples. Finally, the algorithm consists in finding a triangle of minimum weight in the constructed graph (the weight of a triangle is defined as the sum of the edge and vertex weights in the triangle). This can be done in time $\mathcal{O}(Mn^{\omega})$ for any graph on n vertices and integer weights in $[0, M]$, where $\omega < 2.376$ is the matrix multiplication exponent [18]. Thus, the running time of our algorithm is $\mathcal{O}(k^2 n^{\omega\frac{k}{3}})$.

Let us now mention some results of [19] for k-SPARSEST SUBGRAPH that we will use in the following.k-SPARSEST SUBGRAPH admits:

- A polynomial-time algorithm in split graphs.

- A $PTAS$ in proper interval graphs.

- An $\mathcal{FPT}$ algorithm in interval graphs (parameterized by the cost of the solution).

Thus, by Propositions 5 and 6, we have the following result:

Proposition 7. *The following results hold for* U-SM-GP*:*

- *in general graphs, a* $(2-2/k)$*-approximated solution can be found in* $\mathcal{O}(k^2 n^{\omega\frac{k-1}{3}})$ *time.*
- *in split graphs, a* $(2-2/k)$*-approximated solution can be found in polynomial time.*
- *in proper interval graphs, a* $(2-2/k+1/\epsilon)$*-approximated solution can be found in polynomial time for any fixed* $\epsilon > 0$*.*
- *in interval graphs, a* $(2-2/k)$*-approximated solution can be found in* $\mathcal{FPT}$ *time (parameterized by the cost of the solution).*

5.3. On Graphs on Low Density

Let us now consider the algorithm that picks randomly an unbalanced solution, *i.e.* that picks uniformly $(k-1)$ different vertices[2] $\{X_1, \ldots, X_{k-1}\}$. Let $\mathcal{A}_{rand}$ denotes the random variable corresponding to the value of the corresponding solution. As we can expect, the expected value of $\mathcal{A}_{rand}$ decreases in graphs of low density.

Proposition 8. $\mathbb{E}(\mathcal{A}_{rand}) \leq (k-1)(1+\frac{m(k-2)}{n(n-1)})$.

Proof. We can upper bound the expected value of $\mathcal{A}_{rand}$ as follows (assuming that the k^{th} cluster is connected to the $(k-1)$ singletons):

$$\mathbb{E}(\mathcal{A}_{rand}) \leq k-1+\frac{1}{2}\sum_{i,j\in\{X_1,\ldots,X_{k-1}\},i\neq j}\mathbb{E}(\mathbb{1}_{i,j})$$

where $\mathbb{1}_{i,j}$ equals 1 if there is an edge between vertex i and j, and 0 otherwise.

Moreover, for any i and j we have $\mathbb{E}(\mathbb{1}_{i,j}) = m\frac{2}{n(n-1)}$. Thus, we get

$$\mathbb{E}(\mathcal{A}_{rand}) \leq k-1+(k-1)(k-2)\frac{m}{n(n-1)}$$

Now, we prove that the $(1+\frac{m(k-2)}{n(n-1)})$ ratio can be achieved by a deterministic polynomial-time algorithm using classical de-randomized arguments.

Proposition 9. *There is a deterministic polynomial* $(1+\frac{m(k-2)}{n(n-1)})$*-approximation algorithm for* U-SM-GP*.*

[2] In this section, picking uniformly x different elements corresponds to pick uniformly an element among the set of x-subsets.

Proof. We will de-randomize $\mathcal{A}_{rand}$ using the conditional expectation method. Roughly speaking, at each of the $(k-1)$ steps we find in polynomial time the next most "promising" vertex, *i.e.* the vertex that minimizes our upper bound on the conditional expectation.

Let us consider a fixed instance composed of $G = (V, E)$ and $k \leq |V|$. For any set of $k-1$ different vertices $\{x_1, \dots, x_{k-1}\}$, let us define $f(\{x_1, \dots, x_{k-1}\}) = k - 1 + \frac{1}{2}\sum_{i,j \in \{x_1,\dots,x_{k-1}\}, i \neq j} \mathbb{1}_{i,j}$. Notice that $f(\{x_1\ , \dots,\ x_{k-1}\})$ corresponds to the upper bound used in Proposition 8 on the cost of the unbalanced solution where $\{x_1, ..., x_{k-1}\}$ are singletons. Thus, Property 8 states that when choosing uniformly the $\{X_t, 1 \leq t \leq k-1\}$ we get $\mathbb{E}(f(\{X_1, \dots, X_{k-1}\})) \leq b(n, m, k)$, with $b(n, m, k) = (k-1)(1 + \frac{m(k-2)}{n(n-1)})$.

Let us now come back to the de-randomization. For any i, $0 \leq i \leq k-2$ and any set of i different vertices $\{x_1, \dots, x_i\}$, let $X(\{x_1, \dots, x_i\})$ be the following random variable:

- pick uniformly (among $V \setminus \{x_1, \dots, x_i\}$) $(k-1-i)$ different vertices $(X_t)_{1 \leq t \leq k-1-i}$
- return $f(\{x_1, \dots, x_i, X_1, \dots, X_{k-1-i}\})$.

Thus, following the conditional expectation principle it is now sufficient to prove by induction on i $(0 \leq i \leq k-2)$ that we can find in polynomial time $(x_1, \dots, x_i)$ such that $\mathbb{E}(X(x_1, \dots, x_i)) \leq b(n, m, k)$. Statement is clearly true for $i = 0$ as $\mathbb{E}(X(\emptyset)) = \mathbb{E}(f(\{X_1, \dots, X_{k-1}\})) \leq b(n, m, k)$. Let us now consider any i, and see how to find the $(i+1)^{th}$ vertex. We have:

$$\mathbb{E}(X(x_1, \dots, x_i)) = \frac{1}{n-i} \sum_{x_t \notin \{x_1,\dots,x_i\}} \mathbb{E}(X(x_1, \dots, x_i, x_t))$$

As $\mathbb{E}(X(x_1, \dots, x_i)) \leq b(n, m, k)$, there must exists $x_t \notin \{x_1, \dots, x_i\}$ such that $\mathbb{E}(X(x_1, \dots, x_i, x_t)) \leq b(n, m, k)$, and thus we chose x_{i+1} as:

$$x_{i+1} = \underset{x_t \notin \{x_1,\dots,x_i\}}{\text{argmin}} \left(\mathbb{E}(X(x_1, \dots, x_i, x_t))\right)$$

Moreover, notice that for any x_t, $\mathbb{E}(X(x_1, \dots, x_i, x_t))$ can be computed in polynomial time using directly the definition of f and the linearity of expectation (even if the X_t are not independent).

6. Exact Algorithms for Fixed k and Open Problems

In terms of exact algorithms, the $W[1]$-hardness presented in Section 2 shows that we cannot expect to solve the problem in $\mathcal{O}(f(k).p(n, k))$ for any function f and any polynomial p. Hence, a natural question is to ask whether the problem is in the complexity class $\mathcal{XP}$, *i.e.* if there exists an exact algorithm running in $\mathcal{O}^*(n^{f(k)})$ time for some function f. A natural approach would be to enumerate all possible quotient graphs, and then to try to match them one by one. Here we show that this strategy leads to a polynomial algorithm for the weighted problem for $k = 3$. Unfortunately, generalizing this simple method is hopeless even in the unweighted case, as it corresponds to the H-compaction problem of Vikas et al. [12], who proved, among others, that matching a non-chordal graph (and thus the cycle of length four in the case of $k = 4$) is $\mathcal{NP}$-hard. Nevertheless, we bypass the previous strategy to show how we still can solve the problem for $k = 4$ in the unweighted case.

6.1. A Polynomial Algorithm for $k = 3$

Here we present a polynomial algorithm for SM-GP when $k = 3$:

Theorem 4. SM-GP *is polynomial-time solvable 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 largest 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 4). Let a be the value of the edge $e_a^* = \{a_1, a_2\}$ between V_1^* and V_2^*, and b be the value 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).

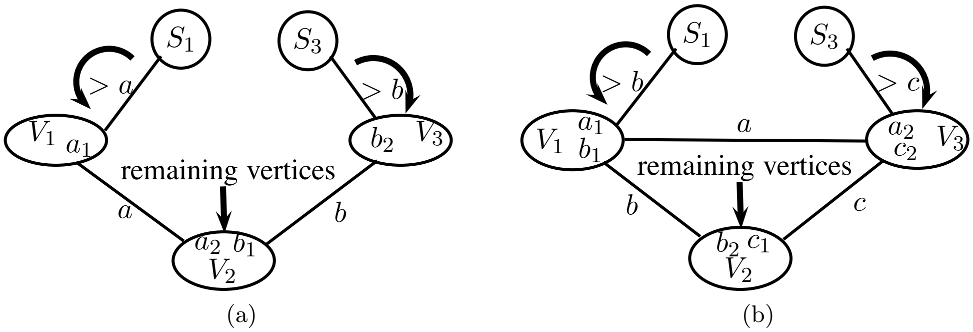


Figure 4: Illustration of the polynomial algorithm for $k = 3$. Bold arrows represent assignments to clusters. (a): One optimal solution contains 2 edges (b): Any optimal solution contains 3 edges

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 4). 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 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.

6.2. Higher k Values for the Unweighted Problem

Let us try to generalize the previous algorithm for higher fixed k values, in the unweighted case. Thus, we are given an input graph $G = (V, E)$, and we enumerate all quotient graphs with k vertices in increasing order of number of edges. Let H be a given target graph with k vertices, and let $\bar{H}$ be the graph obtained from H by adding a loop on every vertex. It is easily seen that:

Remark 2. *G admits a k-partition P such that $G_{/P}$ is isomorphic to H if and only if there exists an edge-surjective homomorphism (also called a compaction in [12]) from G to $\bar{H}$.*

Unfortunately, given a fixed graph H, testing whether a graph G admits a compaction to $\bar{H}$ is $\mathcal{NP}$-hard if H is not a chordal graph [12]. Thus, it is hopeless to extend the previous algorithm for all fixed k, as even for $k = 4$ this would lead to test whether G admits a compaction to the cycle of length four, which is a non-chordal graph.

Nevertheless, we can slightly modify the algorithm to obtain the following result for the unweighted version of our problem:

Proposition 10. U-SM-GP *is polynomial-time solvable for $k = 4$.*

Proof. Let $G = (V, E)$ be our input graph, with $|V| \geq 4$. We will successively try to build a 4-partition of cost $C = 3, 4, 5, 6$.
By Proposition 3, if $diam(G) \geq 3$, then we can build a 4-partition of cost 3 (where the quotient graph is a path).
Otherwise, G admits a 4-partition of cost 3 if and only if G contains an independent set of size 3. Indeed, if such a set $\{x, y, z\}$ exists, then $\{x\}$, $\{y\}$, $\{z\}$, $V\backslash\{x, y, z\}$ is a partition of cost 3. Conversely, if $diam(G) < 3$ and G admits a 4-partition of size 3, then the quotient graph must be isomorphic to the star $K_{1,3}$, and thus contains an independent set of size 3.
Suppose now that G does not admit a 4-partition of cost 3. If there exists $x, y, z \in V$ such that $\{x, y\}, \{x, z\} \notin E$, then $\{x\}$, $\{y\}$, $\{z\}$, $V\backslash\{x, y, z\}$ is a 4-partition of cost at most 4. Otherwise, then every vertex of G is adjacent to at least $|V| - 2$ vertices. In this case it is easily seen that unless G is isomorphic to C_4, it cannot admit a 4-partition of cost 4.
Thus, if now G is not complete, then there exists $x, y \in V$ such that $\{x, y\} \notin E$. In this case we can clearly build a 4-partition of cost 5. Otherwise G is complete and the problem is obvious.
Finally, notice that the running time of the algorithm is $\mathcal{O}(n^3)$, since we only enumerate pairs or triplets of vertices.

7. Conclusion and Future Work

The strategy used in the previous result is a bypass of the general algorithm presented in the beginning of Section 6.2, in the sense that if we are not able to test in polynomial time if there exists a partition (or equivalently a compaction) to a given pattern, then we try with an "easier" pattern which uses the same number of edges (or less). In the previous algorithm for instance, instead of testing to match the cycle of length four, we first try to match the 3-pan (*i.e.* the cycle of length 3 with a pending vertex). Thus, an idea for solving the unweighted problem for all fixed k would be to solve the following problem in polynomial time: given an input graph G and a number of edges C as objective, such that G does not admit a compaction to a chordal graph with C edges, does G admit a compaction to a graph with C edges ?

On the other side, because U-SM-GP is harder than the INDEPENDENT SET problem (see Theorem 1), it would be interesting to investigate the complexity of our problem on restricted graph classes, such as graph classes where INDEPENDENT SET is polynomial, as for instance perfect graphs and their subclasses.

[1] C.-E. Bichot, P. Siarry, Graph Partitioning, Wiley-ISTE, 2011.

[2] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[3] O. Goldschmidt, D. S. Hochbaum, Polynomial algorithm for the k-cut problem, in: Proceedings of the 29th annual symposium on Foundations Of Computer Science, 1988, pp. 444–451.

[4] H. Saran, V. V. Vazirani, Finding k cuts within twice the optimal, SIAM Journal of Computing 24 (1) (1995) 101–108.

[5] R. G. Downey, V. Estivill-Castro, M. R. Fellows, E. Prieto, F. A. Rosamund, Cutting up is hard to do: The parameterised complexity of k-cut and related problems, Electronic Notes in Theoretical Computer Science 78 (0) (2003) 209 – 222.

[6] M. Hansen, P.and Delattre, Complete-link cluster analysis by graph coloring, Journal of the American Statistical Association 73 (362) (1978) pp. 397–403.

[7] S. B. Patkar, H. Narayanan, An efficient practical heuristic for good ratio-cut partitioning, in: Proceedings of the 16th International Conference on VLSI Design, 2003, pp. 64–69.

[8] T. Gonzalez, On the computational complexity of clustering and related problems, in: System Modeling and Optimization, Vol. 38, 1982, pp. 174–182.

[9] M. Koivisto, An O(2^n) algorithm for graph coloring and other partitioning problems via inclusion-exclusion, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 583–590.

[10] T. Feder, P. Hell, S. Klein, R. Motwani, Complexity of graph partition problems, in: Proceedings of the 31st annual Symposium on Theory Of Computing, 1999, pp. 464–472.

[11] P. Hell, Graphs and Homomorphisms, Oxford University Press, 2004.

[12] N. Vikas, Computational complexity of compaction to reflexive cycles, SIAM Journal of Computing 32 (1) (2002) 253–280.

[13] N. Vikas, Computational complexity classification of partition under compaction and retraction, in: Proceedings of the 10th international Computing and Combinatorics Conference, 2004, pp. 380–391.

[14] N. Vikas, A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results, Journal of Computer and System Sciences 71 (4) (2005) 406–439.

[15] N. Vikas, Algorithms for partition of some class of graphs under compaction, in: Proceedings of the 17th international Computing and Combinatorics Conference, 2011, pp. 319–330.

[16] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, in: Proceedings of the 37th annual symposium on Foundations Of Computer Science, 1996, pp. 627–636.

[17] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, in: Proceedings of the 9th annual Symposium on Theory Of Computing, 1987, pp. 1–6.

[18] J. Nesetril, S. Poljak, on the complexity of the subgraph problem, Commentationes Mathematicae Universitatis Carolinae 26 (1985) 415–419.

[19] R. Watrigant, M. Bougeret, R. Giroudeau, The k-sparsest subgraph problem, Tech. Rep. RR-12019, LIRMM-CNRS-UMR 5506 (2012).