

Complexity of the multicut problem, in its vanilla, partial and generalized versions, in graphs of bounded treewidth

Cédric Bentz, Pierre Le Bodic

▶ **To cite this version:**

Cédric Bentz, Pierre Le Bodic. Complexity of the multicut problem, in its vanilla, partial and generalized versions, in graphs of bounded treewidth. Theoretical Computer Science, 2020, 809, pp.239-249. 10.1016/j.tcs.2019.12.015 . hal-02444300

HAL Id: hal-02444300
https://cnam.hal.science/hal-02444300

Submitted on 7 Mar 2022

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

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



Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S030439751930790X
Manuscript_9a3513811eabd13ff34f74e9ef2e7069

Complexity of the Multicut problem, in its Vanilla, Partial and Generalized versions, in Graphs of Bounded Treewidth

Cédric Bentz[a,*], Pierre Le Bodic[b]

[a] *CNAM, CEDRIC, 292 rue Saint-Martin, 75141 Paris, France*
[b] *Faculty of Information Technology, Monash University, Melbourne, Australia*

Abstract

In the Multicut problem, the input consists of a graph and a set of pairs of terminal vertices that have to be disconnected by the removal of a set of edges with minimum total weight. The Partial Multicut problem generalizes Multicut by only requiring a given number of pairs of terminals to be disconnected by the removal of edges. It has been shown that Multicut remains **APX**-hard if the treewidth $tw(G)$ of its input graph G is bounded, but that it is FPT w.r.t. $tw(G+H)$, where H is the demand graph, whose vertices are the terminals and whose edges are the pairs of terminals. We prove that this also holds for Partial Multicut. Furthermore, it has been proved that Multicut also becomes polynomial-time solvable if $tw(G)$ is bounded and H is complete (this is the Multiterminal Cut problem). We extend this result in two directions, by proving that Multicut remains polynomial-time solvable if $tw(G+\bar{H})$ is bounded, and that this remains true for Partial Multicut. Finally, we show that if we further generalize the problem to allow non-unitary profits for pairs of terminals, then the problem is weakly **NP**-hard and has an FPTAS if $tw(G+H)$ is bounded, and becomes **APX**-hard if $tw(G+\bar{H})$ is bounded.

Keywords: partial multicut, graphs of bounded treewidth

1. Introduction

Cut problems in graphs are known to have strong links with many other combinatorial problems [31], and are a relevant framework for modeling various real-life problems. In particular, many reliability network problems can be reduced to several kinds of cut problems: the failure of a node or a link of the telecommunication network corresponds to the removal of the associated vertex or edge in the graph that represents it.

In the classical Multicut problem (MC), we are given a connected (un)directed graph $G = (V, E)$ with $n = |V|$ vertices, a set $S \subseteq V^2$ of pairs of *terminals*, a weight function $w : E \to \mathbb{Z}^+$, and we ask for a set of edges $E^* \subseteq E$ of minimum weight $\sum_{e \in E^*} w(e)$ that *disconnects* all pairs of terminals, i.e., such that in $(V, E \backslash E^*)$ there remains no path from the first terminal (source) to the second one (sink) of each pair in S.

MC is known to be equivalent to the Vertex Cover problem in stars with unit weights [16], and hence it is **APX**-hard even in trees. However, if $|S|$ is fixed, then MC is tractable in bounded-treewidth graphs [2, 20], and in planar graphs [4, 14]. In other recent articles, MC have been proved to be FPT w.r.t. the solution size [6, 26].

We also have to mention that MC with unit weights is **APX**-hard in unrestricted graphs, even if $|S| = 3$ [13], and that a few interesting special cases have been proved to be tractable: paths (and more generally directed trees) [11], rings [5], and undirected graphs when $|S| = 2$ [34]. In directed graphs, however, MC becomes **APX**-hard when $|S| = 2$, even in acyclic graphs where the maximum degree is bounded and weights are unitary [3]. In [19], the parameter $tw(G + H)$, which is the treewidth of the graph $G + H$ (the notion of treewidth will be defined properly at the end of this section), is introduced, where $H = (V_H, E_H)$ is the support graph of S (or *demand graph*), i.e. V_H is the set of terminals of G, and any two vertices $u, v \in V_H$ are connected by an edge iff $(u, v) \in S$. The authors prove that MC can be expressed using monadic second order logic, and, as such, using Courcelle's theorem [12] in an extended and constructive version [1], MC can be shown to be FPT w.r.t. parameter $tw(G + H)$. In [28], an explicit dynamic programming algorithm is given, which also proves this result. Moreover, MC admits a 2-approximation algorithm in trees [16], a $O(1)$-approximation algorithm in planar graphs [32],

*Corresponding author
Email address: cedric.bentz@cnam.fr (Cédric Bentz)

© 2019 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

a $O(\log |S|)$-approximation algorithm in undirected graphs [15], and a $O(\sqrt{n})$-approximation algorithm in digraphs [21]. On the negative side, assuming the *Unique Games Conjecture*, MC has no $(2-\epsilon)$-approximation algorithm in trees for any $\epsilon > 0$ (from [22] and the equivalence between MC in stars with weights 1 and the Vertex Cover problem), and no $O(1)$-approximation algorithm in general graphs [8].

In MC, we are interested in the minimum number of edges whose removal makes the network entirely collapse (no pair of terminals can communicate with each other). In this paper, we investigate partial and generalized versions of MC, which are at least as hard as MC. The motivation for considering this partial version comes from the fact that, in practice, we are not always interested in the minimum number of link or node failures that can make the entire network collapse, but in the minimum number of such failures that can make a given portion of the network collapse.

Formally, in the Partial Multicut problem (PMC), we are given an integer $P \geq 1$, and we are asking for the minimum weight of a set of edges that disconnects at least P pairs of terminals. Hence, when $P = |S|$, PMC is exactly MC. In the Generalized Partial Multicut problem (GPMC), we additionally define a profit function $p : S \to \mathbb{Z}^+$, and a feasible solution must disconnect a set of terminal pairs of total profit at least P.

PMC has been studied in [18, 25], where $(8/3+\epsilon)$ and $O(\log^2 n \log\log n)$-approximation algorithms are given for trees and unrestricted graphs, respectively. The approximation ratio in trees has been improved to $2+\epsilon$ in [27], and the $O(\log^2 n \log\log n)$-approximation algorithm has been extended to GPMC in [24]. To the best of our knowledge, the only known tractable case for PMC is the one of rooted trees [18] (which immediately implies the tractability of PMC in graphs of maximum degree two), and GPMC has only been studied in [25] as the "prize-collecting multicut problem", and in [24]. As MC is also tractable in rooted trees (which are directed trees), and as PMC is hard whenever MC is, this means in particular that we are not aware of any special case where MC and PMC behave differently (with respect to their computational complexity).

We provide new results for (G)(P)MC in particular in the case where the treewidth $tw(G)$ of the input graph G is bounded, together with additional assumptions on either the demand graph H or its complement $\bar{H}$. Table 1 provides a summary of state-of-the-art and new complexity results for MC and related problems depending on the treewidth of G, H, $\bar{H}$, $G+H$ and $G+\bar{H}$.

We end this section by defining properly the notion of *treewidth*. Given a graph $G = (V, E)$, a *tree decomposition* of G is a pair $\big(\{X_i | i \in I\}, T\big)$ where $X_i \subseteq V, \forall i \in I$, is a *bag* of vertices of G, $T = (I, F)$ is a tree, and:

(1) $\bigcup_{i \in I} X_i = V$,

(2) For every edge $uv \in E$, there is an $i \in I$ such that $u, v \in X_i$,

(3) For all $i, j, l \in I$, if j lies in the path between i and l, then $X_i \cap X_l \subseteq X_j$.

The *width* of a given tree decomposition of a graph G is equal to $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width of a tree decomposition of G, taken over all tree decompositions of G. Note that trees (and hence chains and stars) have treewidth 1. Without loss of generality, we can also assume that the tree decomposition is *nice* [23], i.e.:

- T is rooted at some node r,
- T is binary and has $O(|V|)$ nodes,
- If a node i has two children j and k then $X_i = X_j = X_k$ (join node)
- If a node i has one child j, then either
  - (a) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$ (forget node)
  - (b) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ (introduce node)

Such a nice tree decomposition of width $tw(G)$ can be computed in polynomial time when $tw(G) = O(1)$ [23].

2. (G)(P)MC with $tw(G) = 1$ when G is directed

We begin by studying MC, PMC and GPMC when G is directed and $tw(G) = 1$, meaning that G is a directed tree. Recall that, in this case, MC is tractable. Moreover, PMC is tractable in rooted trees. We shall show that, on the contrary, PMC (and hence GPMC with unit profits) is strongly **NP**-hard in directed stars, hence providing the first special case where MC and PMC do not have the same behavior with respect to their computational complexity.

First, we show that PMC in directed stars is polynomially equivalent to another partial covering problem, called the Partial Weighted Vertex Cover problem (PWVC), in bipartite graphs. In this problem, we are given a node-weighted graph, and the goal is to compute the minimum weight of a set of vertices that covers at least a given number of edges. It is clearly **NP**-hard in general graphs (as optimally covering *all* edges is), and in [7] it was shown to remain **NP**-hard in bipartite graphs as well, even when all weights are 1. Let us show the following proposition:

Proposition 1. *PMC in directed stars is equivalent to PWVC in bipartite graphs.*

Bounded Parameter(s)	*MC*	*PMC*	*GPMC*
$tw(G)$, G undirected	**APX**-hard even in unweighted stars [16]		
$tw(G)$, G directed	**P** if $tw(G) = 1$	Strongly **NP**-hard in directed stars (Sec. 2)	
$tw(G) + tw(H)$	**APX**-hard (Sec. 3.1)		
$tw(G + H)$	**FPT** [19, 28]	**FPT** (Sec. 3.3)	**NP**-hard & **FPTAS** (Sec. 3.2 and 3.4)
$tw(G + \bar{H})$	**P** (Sec. 4.1)		**APX**-hard (Sec. 4.2)

Table 1: Summary of complexity results for MC in graphs of bounded treewidth.

Proof. Take any PMC instance in a directed star. We can assume without loss of generality that all sources have out-degree 1 and in-degree 0, and that all sinks have out-degree 0 and in-degree 1. We associate an undirected bipartite graph to this directed star by taking as vertices the sources and sinks (the weight of each vertex being the weight of the arc incident to the corresponding source or sink), and by adding an edge between two vertices if the corresponding source and sink in the directed star belong to the same terminal pair. Then, in the PMC instance defined in a directed star, removing an arc is clearly equivalent, in the PWVC instance defined in a bipartite graph, to selecting the vertex (source or sink) this arc is incident to. Conversely, any PWVC instance in a bipartite graph can be polynomially reduced to a PMC instance in a directed star, using the same correspondence (edges become terminal pairs). □

Together with [7], Proposition 1 immediately implies:

Corollary 1. *PMC is **NP**-hard in directed stars, even when all weights are 1.*

The results in the rest of the paper are concerned with *undirected* graphs.

3. (G)(P)MC with bounded $tw(G + H)$

We now study MC, PMC and GPMC when the parameter $tw(G + H)$ is bounded. We already know that MC is FPT w.r.t. $tw(G+H)$ [19], and that it is **APX**-hard if only $tw(G)$ is bounded. First, in Section 3.1, we prove that MC remains **APX**-hard if both $tw(G)$ and $tw(H)$ are bounded. Second, in Section 3.2, we show that GPMC is weakly NP-hard even if $tw(G+H)$ is bounded. Third, in Section 3.3, we provide a pseudopolynomial-time algorithm for GPMC if $tw(G + H)$ is bounded. This algorithm is FPT w.r.t. $tw(G + H)$ when profits are unitary, thereby extending [19] to PMC. Fourth, in Section 3.4, we show how a simple rounding scheme can provide an FPTAS for GPMC when $tw(G+H)$ is bounded.

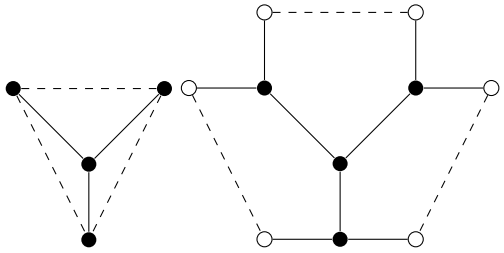


Figure 1: Example of instances $\mathcal{I}_1$ (left) and $\mathcal{I}_2$ (right).

3.1. *APX*-hardness of MC with bounded $tw(G)$ and $tw(H)$

We now prove that MC with bounded $tw(G)$ remains **APX**-hard even if we additionally suppose that $tw(H)$ is bounded.

Proposition 2. *MC in trees of height* 1 *is equivalent to MC in trees of height* 2 *with H a set of vertex-disjoint edges.*

Proof. Let $\mathcal{I}_i$ denote an instance of MC in a tree T_i of height i, with demand graph H_i, for $i \in \{1, 2\}$. First, let $\mathcal{I}_1$ be given with arbitrary H_1, and let r be the central vertex of T_1. We can suppose w.l.o.g. that r has no incident edge in H_1. We create an instance $\mathcal{I}_2$ by replacing each leaf v of T_1 by a star with as many leaves as there are edges incident to v in H_1. In $\mathcal{I}_2$, v is no longer a terminal, and each created leaf is a terminal that is incident to a single edge of H_2. The graph H_2 is thus a set of vertex-disjoint edges, implying that $tw(H_2) = 1$, and hence that $tw(T_2) + tw(H_2) = 2$. Figure 1 shows an example where the edges of T_i are solid, while the edges of H_i are dashed. Moreover, the vertices appearing both in T_1 and T_2 are the black ones, while the vertices appearing in T_2 only are the white ones. The equivalence directly comes from the fact that, from each solution to $\mathcal{I}_2$, we can always build an alternative solution that costs no more, and that contains only edges incident to r. □

Note that the equivalence naturally extends to the weighted case. This proposition, together with the fact that MC with unit weights is **APX**-hard in stars (Theorem 3.1 of [16]), directly leads to the following result:

Theorem 1. *MC with unit weights is **APX**-hard in trees of height 2, even if H is a set of vertex-disjoint edges.*

*3.2. Weak **NP**-hardness of GPMC with bounded $tw(G+H)$*

There is a natural reduction from the MINIMUM KNAPSACK problem (MinKP), which is weakly **NP**-hard (since it is equivalent to the classical MAXIMUM KNAPSACK problem), to GPMC where $G = H$ is a star, and thus $tw(G + H) = 1$. Given a MinKP instance, the associated GPMC instance consists of a central vertex with one leaf for each knapsack item. For each knapsack item, there is a terminal pair between the central vertex and a leaf, with corresponding profit and weight (on the edge). Hence the edges of H are the edges of G. In a solution to GPMC, an isolated terminal translates into a selected item in a MinKP solution. Conversely, given a non-trivial instance of GPMC where $G = H$ is a star, we can build an equivalent instance of MinKP in the same fashion: there is an item for each leaf, whose weight (resp. profit) is the one of the edge of G (resp. of H) incident to this leaf. This proves the following:

Theorem 2. *MinKP is equivalent to GPMC if $G = H$ is a star.*

Hence GPMC is weakly **NP**-hard when $tw(G + H) = 1$.

3.3. An algorithm for GPMC with bounded $tw(G + H)$

We design a dynamic programming algorithm which is FPT w.r.t. $tw(G+H)$ for PMC and runs in pseudopolynomial time for GPMC.

Suppose that the graph $G + H$ has bounded treewidth $tw(G + H)$, and that a nice tree decomposition $\big(\{X_i | i \in I\}, T\big)$ of $G + H$ of width $tw(G + H)$ is provided, such that the bag of each leaf of T contains a single vertex of G (given a nice tree decomposition where the bag of a leaf is of arbitrary cardinality, adding at most $tw(G+H)$ forget nodes is sufficient for each leaf). We add to the root r of T a gadget similar to the one for the leaves, that is to say a path in T, with r at one end, composed only of forget nodes, and which other end is an empty bag. For each $i \in I, E_i = E \cap X_i^2$, i.e., E_i denotes the set of edges of $G + H$ induced by X_i.

The output of the algorithm will be a partitioning of vertices into components, determined by a coloring of vertices. A coloring function on vertices is a function $c : V \to \mathcal{C}$, where $\mathcal{C} \subseteq \mathbb{Z}$ is a set of colors, such that two vertices with different colors are not in the same component. Two vertices with the same color are in the same component if and only if there exists a path in G using only vertices with the same color. Hence two different components may be represented with the same color if they are not adjacent, i.e. if there exists no edge between any vertex of the first component and any vertex of the second component. Such a graph coloring determines the components as well as the set of edges between them (i.e., the bicolored ones). In general, the number of components of a graph G is thus not bounded by the number of colors $|\mathcal{C}|$.

The weight (resp. profit) associated with a given coloring function c in a subtree T_i rooted at an arbitrary node $i \in I$ is defined as the sum of the weights (resp. profits) over all edges uv of G (resp. H) such that (1) both u and v lie in a bag at a node of T_i, (2) $uv \notin E_i$, and (3) $c(u) \neq c(v)$. At the root r, this definition corresponds to the total weight (resp. profit) associated with a coloring function c.

Given a partial coloring function c_i on a bag X_i, we therefore define $f(i, c_i, p_i)$ to return the minimum weight in the subtree T_i rooted at i over all coloring functions c which extend c_i and yield a profit at least p_i in T_i. If no such coloring exists, then $f(i, c_i, p_i)$ returns $+\infty$.

The function f is defined recursively via four node cases.

If i is a leaf. Then, by convention, $X_i = \{v\}$, and

$$f(i, c_i, p_i) = \begin{cases} 0 & \text{if } p_i = 0, \\ +\infty & \text{otherwise.} \end{cases}$$

If i is an introduce node. Let j be the child of i such that $X_i = X_j \cup \{v\}$. We then simply have

$$f(i, c_i, p_i) = f(j, c_j, p_i)$$

where c_j is the restriction of c_i to X_j.

If i is a join node. Let j and h be the children of i, such that $X_i = X_j = X_h$. We then have

$$f(i, c_i, p_i) = \min_{p_i = p_j + p_h} \big(f(j, c_i, p_j) + f(h, c_i, p_h)\big)$$

If i is a forget node. Let j be the child of i such that $X_i = X_j \setminus \{v\}$. Then,

$$f(i, c_i, p_i) = \min_{a \in \mathcal{C}} \Big(f(j, c_j, \max(0, p_i - \sum_{uv \in E_j,\ c_i(u) \neq a} p_{uv})) + \sum_{uv \in E_j,\ c_i(u) \neq a} w_{uv}\Big)$$

where c_j is the coloring function that extends c_i over X_j and $c_j(v) = a$, and where $p_{uv} = 0$ if $uv \notin S$.

At the root node r. Calling $f(r, c_r, P)$ returns the minimum cost of a partition of the vertices of G which yields a profit at least P.

Proof of correctness. The algorithm relies on the fact that the tree decomposition has exactly one forget node per vertex of G. The algorithm provides a coloring of G, which determines a partition of G, and therefore the set of edges of G and H that are cut. The weight or profit of an edge uv of $G+H$ is taken into account at the forget node where one of the two vertices is removed (both u and v belonging to its child). Such a node necessarily exists by property of the tree decomposition, and because the bag of r is empty. Furthermore, such a node is necessarily unique, as the vertex removed cannot be introduced again at a later point in the tree decomposition.

Number of colors. The following lemmas help us establish better running times:

Lemma 3. *Given a graph G, $\chi(G) \leq tw(G) + 1$*

where $\chi(G)$ is the chromatic number of G.

Proof. Given any graph G and a nice tree decomposition for G of minimum width $tw(G)$, rooted at r, we show how to define a coloring of G using $tw(G) + 1$ colors. Start at the root bag X_r, and give every vertex in X_r a different color. We process the tree in a top-to-bottom fashion: once a color has been assigned to a vertex, the color of this vertex does not change. At a forget node, the new vertex is given a color that is not used in the parent bag (it may have been used earlier). The number of colors used in each bag is its cardinality, hence the total number of colors needed to color the whole graph is $tw(G) + 1$. This coloring is proper since for every edge there exists a bag containing its two endpoints. □

Lemma 4. *$tw(G)+1$ colors are sufficient to color any feasible solution of a GPMC instance in a graph G.*

Proof. Suppose we have a feasible solution $E' \subseteq E$ to such a GPMC instance. Let G' be the minor of G obtained by contracting every edge in $E \setminus E'$. Since $tw(G') \leq tw(G)$ (see [30]), we have $\chi(G') \leq tw(G) + 1$ by Lemma 3. Thus, we can compute a proper coloring c of G' with size at most $tw(G)+1$, and, in G, give each vertex the color of the vertex of G' it has been contracted into. This way, every edge in E' has endpoints with different colors, or c would not be proper. □

In the following analysis, we can thus suppose that the set of colors $\mathcal{C}$ has size bounded by $tw(G+H)+1$. Throughout, two vertices u and v which have been assigned the same color lie in the same component if and only if there exists a path in G between u and v that only goes through vertices with the same color.

Types of Nodes	Asymptotic Running Times
Leaf Node	$O(1)$
Introduce Node	$O(1)$
Join Node	$O(P)$
Forget Node	$O(tw(G+H)^2)$
Root Node	$O(1)$

Table 2: Running time for computing f on each type of node.

Running time. The running time depending on the type of nodes is given in Table 2. There were $O(n)$ nodes initially, but we added some forget nodes. However, after this transformation, there are exactly n forget nodes, hence there are still $O(n)$ nodes in T. Moreover, there are $O((tw(G+H)+1)^{tw(G+H)+1})$ possible colorings for each bag and P possible profits, hence the overall running time of this dynamic programming algorithm is $O(n\,(tw(G+H)+1)^{tw(G+H)+1}P(P+tw(G+H)^2))$.

Theorem 5. *There is a pseudopolynomial-time algorithm that solves GPMC if $tw(G+H)$ is bounded.*

Furthermore, PMC corresponds to the case where all profits are equal to 1, and therefore $P \leq n^2$. This yields:

Theorem 6. *PMC is FPT w.r.t. $tw(G+H)$.*

3.4. An FPTAS for GPMC with bounded $tw(G+H)$

We present an FPTAS for GPMC in graphs with bounded $tw(G+H)$ based on an FPTAS by rounding for MinKP [29]. In the traditional FPTAS for the Maximum Knapsack problem [33], the sum of the objective coefficients provides a trivial upper bound on the optimal value, which can be readily used to determine an appropriate scaling factor. By contrast, there is no trivial lower bound for MinKP. One alternative consists in computing a lower bound using a separate algorithm (as hinted in [17]). We first show how an FPTAS can be obtained if these bounds can be guessed correctly.

Algorithm 1 A rounding FPTAS for GPMC with bounded $tw(G+H)$ and bounds

Let a precision $\epsilon > 0$ and an instance of GPMC in a graph with n vertices and bounded $tw(G+H)$ be given.
Also suppose that $0 < LB \leq UB$ are given.

1. Let $K = \frac{\epsilon LB}{m}$ and, for each edge uv, set $\tilde{w}(uv) = \lceil \frac{w(uv)}{K} \rceil$
2. Return the optimal solution $\tilde{E}$ found by an algorithm polynomial in n and $\tilde{W} = (1+\epsilon)UB/K$ on the modified instance

Theorem 7. *Given $LB \leq OPT \leq UB = g(n)LB$ as input, where g is a polynomial, Algorithm 1 is an FPTAS for GPMC with bounded $tw(G+H)$.*

Proof. First, we prove that Algorithm 1 achieves a $(1+\epsilon)$ ratio. For any edge $uv \in E$, we have

$$K\tilde{w}(uv) < w(uv) + K,$$

and thus, for an optimal solution E^*,

$$K\tilde{w}(E^*) < w(E^*) + mK. \tag{1}$$

Finally, we can prove:

$$\begin{aligned} w(\tilde{E}) &\leq K\tilde{w}(\tilde{E}) && \text{by rounding up,} \\ &\leq K\tilde{w}(E^*) && \text{by optimality of } \tilde{E} \text{ for } \tilde{w}, \\ &< w(E^*) + mK && \text{because of (1),} \\ &= OPT + \epsilon LB && \text{by definition of } K, \\ &\leq (1+\epsilon)OPT && \text{by definition of } LB. \end{aligned}$$

In establishing this result, we have not used the fact that we set $\tilde{W} = (1+\epsilon)UB/K$ in Algorithm 1. We have just proved that $K\tilde{w}(\tilde{E}) \leq (1+\epsilon)OPT$, and thus $\tilde{W} = (1+\epsilon)UB/K$ is a valid upper bound on $\tilde{w}(\tilde{E})$.

Second, we prove that the running time of Algorithm 1 is indeed polynomial in n and $1/\epsilon$. Step 1 is linear in the number of edges, hence in $O(n^2)$. Step 2 requires an algorithm polynomial in n and $\tilde{W}$. One such algorithm can be obtained by defining a function $h(i, c_i, w_i)$ similar to $f(i, c_i, p_i)$ (see Section 3.3), where h returns the maximum profit rather than the minimum weight (for f), and essentially exchanging the role of P and $\tilde{W}$. By an analysis similar to the one used for f, the computation of $h(r, c_r, \tilde{W})$ can be done in $O(n\,(tw(G+H)+1)^{tw(G+H)+1}\tilde{W}(\tilde{W}+tw(G+H)^2))$ time. In the modified instance, we have

$$\tilde{W} = \frac{(1+\epsilon)UB}{K} = \frac{ng(n)}{\epsilon} + ng(n),$$

which is polynomial in n and $1/\epsilon$. Since, when $tw(G+H) = O(1)$, Step 2 runs in $O(n(\tilde{W})^2)$ time, this enables us to conclude that the whole running time is $O(n^3 g^2(n)/\epsilon^2)$, which is polynomial in n and $1/\epsilon$. □

Theorem 8. *There exists an FPTAS for GPMC with bounded $tw(G+H)$.*

Proof. Simply call the approximation algorithm given in [24, Theorem 13] for general graphs to compute a valid upper bound UB with $g(n) = \log^2(n)\log(\log(n))$. □

Theorem 8 relies on an auxiliary algorithm to produce a valid upper bound UB. Complementary to this approach, we present a simple scaling scheme that allows to identify suitable LB and UB values in polynomial time by iteratively calling Algorithm 1. We first need a simple observation:

Lemma 9. *If $LB \leq OPT$, then the output SOL of Algorithm 1 has the following properties:*

1. *$SOL \leq (1+\epsilon)UB \Rightarrow SOL \leq (1+\epsilon)OPT$,*
2. *$SOL > (1+\epsilon)UB \Rightarrow OPT > UB$.*

Proof. 1. Suppose $SOL \leq (1+\epsilon)UB$. If $UB \leq OPT$, this directly implies $SOL \leq (1+\epsilon)OPT$. Otherwise, $LB \leq OPT < UB$, in which case Theorem 7 applies, hence $SOL \leq (1+\epsilon)OPT$.

2. Suppose $SOL > (1+\epsilon)UB$. If we had $LB \leq OPT \leq UB$, this would contradict Theorem 7. Hence $OPT > UB$.

□

Lemma 9 allows us to write an algorithm which scales up LB and UB geometrically until they define an interval on OPT.

Algorithm 2 A scaling and rounding FPTAS for GPMC with bounded $tw(G+H)$

Let a scaling factor $\alpha > 1$ be given.

1. Let $LB = 1$.
2. Let $UB = \alpha LB$
3. Call Algorithm 1 with values LB and UB. Let SOL denote the value of the solution ($+\infty$ if there is none).
4. If $SOL > (1+\epsilon)UB$, let $LB = UB$, and go to Step 2.
5. Return SOL.

Lemma 10. *At any point of Algorithm 2, $LB \leq OPT$.*

Proof. At the first iteration, $LB = 1$, hence the property holds. The value of LB is only changed at step 4, under the condition that $SOL > (1+\epsilon)UB$. From Lemma 9, this implies $OPT > UB$, hence in the next iteration we will have $OPT > LB$. □

Theorem 11. *Algorithm 2 is an FPTAS for GPMC with bounded $tw(G+H)$.*

Proof. At step 4, as $LB \leq OPT$ holds by Lemma 10, Lemma 9 guarantees that either (1) SOL is a $(1+\epsilon)$-approximation, and the algorithm stops, or (2) $OPT > UB$, and the algorithm iterates.

At iteration $i \geq 0$, we have $LB = \alpha^i$ and $UB = \alpha^{i+1}$, and thus $\tilde{W} = \frac{n\alpha}{\epsilon} + n\alpha$ for any iteration. The running time at each iteration is thus $O(n^3/\epsilon^2)$, since α is a constant. Since the algorithm stops at the first iteration k for which $OPT \leq SOL \leq (1+\epsilon)\alpha^{k+1}$, this means that $\log_\alpha(OPT) \leq \log_\alpha(1+\epsilon) + (k+1)$, which gives the stopping criterion $k \geq \log_\alpha(OPT) - \log_\alpha(1+\epsilon) - 1$, hence the total running time is $O(n^3 \log(OPT)/\epsilon^2)$, which is polynomial in n, the size of the binary encoding of w, and $1/\epsilon$. □

It is interesting to notice that the maximum value $\tilde{W}$ of an optimal solution considered in the modified problem with rounded weights in Algorithm 1 is fixed for all iterations of Algorithm 2. The modified weights of individual edges become increasingly smaller, and thus more edges can belong to an optimal solution of the modified problem. Indeed, in Algorithm 1, any edge uv with $\tilde{w}(uv) \geq \tilde{W}$ cannot be cut.

4. (G)(P)MC with bounded $tw(G+\bar{H})$

In this section we introduce the parameter $\bar{H}$, which is simply defined as the complement of H. It turns out that a well-structured *dense* H can help design efficient algorithms for MC, as a well-structured *sparse* H does. Indeed, the case where H is complete (and therefore $tw(\bar{H}) = 0$) is equivalent to the Multiterminal Cut problem, a well-studied special case of MC, which is polynomial-time solvable if $tw(G)$ only is bounded [13]. Furthermore, the problem becomes trivially linear-time solvable in trees [9, 10].

In Section 4.1, we show that PMC (and thus MC) is polynomial-time solvable in graphs with bounded $tw(G+\bar{H})$. Note that the algorithm we design is not FPT w.r.t. $tw(G+\bar{H})$, and that, to the best of our knowledge, no result more general than the tractability of the Multiterminal Cut problem in graphs of bounded treewidth was known so far, even for MC.

We then show in Section 4.2 that the generalized version of this problem is not even solvable in pseudopolynomial time in this case, unlike the case where $tw(G+H) = O(1)$. More precisely, we prove that GPMC is **APX**-hard when G is a star and H is complete, i.e., when $tw(G+\bar{H}) = 1$. This highlights the fact that $tw(G+\bar{H})$ is a parameter that does make the problem easier when bounded, and hence may seem similar to $tw(G+H)$ (which was introduced in [19]) in that regard, but actually behaves a bit differently.

4.1. A polynomial-time algorithm for PMC in graphs with bounded $tw(G+\bar{H})$

Suppose that the graph $G+\bar{H}$ has bounded treewidth $tw(G+\bar{H})$, and that a nice tree decomposition $\big(\{X_i | i \in I\}, T\big)$ of $G+\bar{H}$ of width $tw(G+\bar{H})$ is provided. As in Section 3.3, we let $E_i = E \cap X_i^2$, and assume that a gadget is used to set up an empty bag at the root node of T, such that there is exactly one forget node in T per vertex of G.

The profit of a solution to a PMC instance is the number of edges of H having their two endpoints in different components (obtained after the removal of the edges in the solution), and can thus be computed as:

- $|S|$, i.e. the total number of edges of H,
- minus the number of edges of H with both endpoints in the same component. This is itself accounted as the sum, over each component:
  - of the number of edges of a complete graph over the vertices of H in that component,
  - minus the number of edges of $\bar{H}$ with both endpoints in that component.

Hence, we can compute the profit of a solution by knowing the number of vertices of H in each of these components, and the number of edges of $\bar{H}$ with both endpoints in the same component. In particular, the parameters of the recursive function $\bar{f}$ we define must keep track of these quantities. Since we consider a tree decomposition of $G+\bar{H}$, for any pair of vertices of H that are non adjacent in H, and hence adjacent in $\bar{H}$, there must exist a bag of this tree decomposition that contains both vertices. This implies that the number of edges of $\bar{H}$ with both endpoints in a given component can be updated any time a vertex of H is "forgotten", while the number of vertices of H in a given component can be computed *only once the last vertex of this component has been "forgotten"*. The complete list of parameters of $\bar{f}$ is:

- a node i of T,
- a coloring function c_i defined as in Section 3.3.
- a counting function $n_i : \mathcal{C} \to \{0, \ldots, |V_H|\}$, which, for each color $a \in \mathcal{C}$, stores 0 if no vertex in X_i has been assigned color a, and the number of vertices of H (possibly 0) not in X_i that lie in the subtree of T rooted

at i and belong to that component (colored by a) otherwise. Hence $n_i(a)$ counts the number of "forgotten" vertices of H in the component colored by a at node i, if there is at least one vertex of H having color a in X_i.

- $p_i \in \{-|E_{\bar{H}}|, \dots, |S| - P\}$ keeps track of the *uncut* edges of H in the subtree rooted at node i. More precisely, it is computed as the sum of:

  1. *Plus* the number of edges of H for which both endpoints belong to a same "forgotten" component, i.e. such that these components lie in the subtree rooted at i and have no vertex in X_i.
  2. *Minus* the number of edges of $\bar{H}$ for which at least one endpoint does not belong to X_i (i.e., has been "forgotten"), both endpoints lie in the subtree rooted at node i and have the same color, say a, and this color appears in X_i as well as in each bag lying between X_i and any bag containing the first of the two endpoints that has been forgotten (meaning that both endpoints are in the component of color a having at least one vertex in X_i).

  Note that, when there remains in X_i only one vertex of H belonging to a component, all the edges of $\bar{H}$ whose endpoints belong to that component have been taken into account. Hence, when the last vertex of this component is forgotten, it suffices to add the number of edges of a complete graph over the vertices of H in that component in order to obtain exactly the number of edges of H whose endpoints belong to that component.

The returned value of $\bar{f}(i, c_i, n_i, p_i)$ is defined as in Section 3.3.

If i is a leaf. Then, by convention, $X_i = \{v\}$, and

$$\bar{f}(i, c_i, n_i, p_i) = \begin{cases} 0 & \text{if } n_i = 0 \text{ and } p_i = 0, \\ +\infty & \text{otherwise.} \end{cases}$$

If i is an introduce node. Let j be the child of i such that $X_i = X_j \cup \{v\}$. We then simply have

$$\bar{f}(i, c_i, n_i, p_i) = \bar{f}(j, c_j, n_i, p_i),$$

where c_j is the restriction of c_i to X_j.

If i is a join node. Let j and h be the children of i, such that $X_i = X_j = X_h$. We then have

$$\bar{f}(i, c_i, n_i, p_i) = \min_{\substack{p_i = p_j + p_h, \\ n_i = n_j + n_h}} \left(\bar{f}(j, c_i, n_j, p_j) + \bar{f}(h, c_i, n_h, p_h)\right)$$

where $n_i = n_j + n_h$ is an element-wise vector addition.

If i is a forget node. Let j be the child of i such that $X_i = X_j \setminus \{v\}$. Then,

$$\begin{aligned} \bar{f}(i, c_i, n_i, p_i) = \min_{(c_j, n_j) \in comp(c_i, n_i)} \Big(& \bar{f}(j, c_j, n_j, p_i \\ & - \begin{cases} \frac{n_j(a)(n_j(a)+1)}{2} & \text{if } c_j(v) = a,\ v \in H, \\ & \text{and } c_i(w) \neq a\ \forall w \in X_i, \\ \frac{n_j(a)(n_j(a)-1)}{2} & \text{if } c_j(v) = a,\ v \notin H, \\ & \text{and } c_i(w) \neq a\ \forall w \in X_i, \\ 0 & \text{otherwise.} \end{cases} \\ & + |\{uv \in \bar{H}, u \in X_i, c_i(u) = c_j(v)\}| \\ & + \sum_{uv \in E_j,\ c_i(u) \neq a} w_{uv} \Big), \end{aligned}$$

where c_i and n_i are such that, for each $a \in \mathcal{C}$, $c_i(w) \neq a\ \forall w \in X_i \Rightarrow n_i(a) = 0$, and where $comp(c_i, n_i)$ is the set of (c_j, n_j) compatible with (c_i, n_i). For all $c_i \in \mathcal{C}, n_i \in \{0, \dots, |V_H|\}^{|\mathcal{C}|}, (c_j, n_j) \in comp(c_i, n_i)$ if and only if

- c_j is a coloring function that extends c_i over X_j, such that $c_j(v) \in \mathcal{C} \setminus \{a \in \mathcal{C}, n_i(a) = 0, \exists u \in X_i : c_i(u) = a\}$ if $v \in H$.
- for all $a \in \mathcal{C}$,

$$n_j(a) \begin{cases} = n_i(a) - 1 & \text{if } c_j(v) = a,\ n_i(a) \geq 1, \\ & \text{and } v \in H, \\ \in \{0, \dots, |V_H|\} & \text{if } c_j(v) = a \text{ and} \\ & c_i(w) \neq a\ \forall w \in X_i, \\ = n_i(a) & \text{otherwise.} \end{cases}$$

At the root node r. We have added to r the same gadget as in Section 3.3. We call $\bar{f}$ with parameters $i = r, c_i = 0, n_i = 0, p_i = |S| - P$.

Proof of correctness. This algorithm relies on the fact that we can keep track of the number of vertices of H in the components of G associated with coloring c by using n_i. When the last vertex of a component with color a is "forgotten" at a forget node i, we know the number of vertices of H in this component, and thus we know how many edges of H would not be cut if H was complete (this is the term $\frac{n_j(a)(n_j(a)+1)}{2}$ or $\frac{n_j(a)(n_j(a)-1)}{2}$). Since H is in general not complete, we count the uncut edges of $\bar{H}$ as follows: each such edge is taken into account when its first endpoint v of color a is forgotten (this is the term $|\{uv \in \bar{H}, u \in X_i, c_i(u) = a\}|$). We know that, in this way, all uncut edges of $\bar{H}$ will be taken into account, as T is a tree decomposition of $G + \bar{H}$, and hence, for each edge in $\bar{H}$, there is a bag in T containing its two endpoints. Thus, we can retrieve the number of edges of H which were not cut. The other parts of $\bar{f}$ are similar to those of f.

Type of Nodes	Asymptotic Running Time
Leaf Node	$O(1)$
Introduce Node	$O(1)$
Join Node	$O(n^{tw(G+\bar{H})+3})$
Forget Node	$O(n\ tw(G+\bar{H})^2)$
Root Node	$O(1)$

Table 3: Running time for computing $\bar{f}$ on each type of node.

Running time. The running time depending on the type of nodes is given in Table 3. Since there are $O(n)$ nodes, $O((tw(G+H)+1)^{tw(G+H)+1})$ possible colorings for each bag, $|S| - P + |E_{\bar{H}}| = O(n^2)$ possible profits, and $O(n^{tw(G+\bar{H})+1})$ possible countings, the overall running time of the dynamic programming algorithm is $O(n^{2tw(G+\bar{H})+7})$.

Theorem 12. *PMC can be solved in polynomial time if $tw(G+\bar{H})$ is bounded.*

*4.2. **APX**-hardness of GPMC in stars with complete H*

We have proved in Section 4.1 that Partial Multicut is polynomial-time solvable in graphs with bounded $tw(G+\bar{H})$. We now prove that the generalized version of this problem, GPMC, is actually **APX**-hard in these graphs. This result may seem surprising considering the fact that the algorithm for PMC in graphs with bounded $tw(G+H)$ can be extended naturally to handle arbitrary profits in pseudopolynomial time, as shown in Section 3.3. However, the one given in Section 4.1 cannot be adapted to GPMC, as in this case the edges of $\bar{H}$ have no associated profit, and hence cannot be used to compute the total profit of the uncut edges of H.

Theorem 13. *The* Generalized Partial Multicut *problem with unit weights is **APX**-hard if $G = (V, E)$ is a star and H is complete.*

Proof. We reduce MC with arbitrary $H_{MC} = (V_{MC}, E_{MC})$ to GPMC with complete $H_{GPMC} = (V_{MC}, E_{GPMC})$, i.e., H_{GPMC} is the complete graph on V_{MC}, and in both problems $G = (V, E)$ is a star with unit weights. As in [16], we can suppose w.l.o.g. that the central vertex r of G is not a terminal in MC and thus not in GPMC, and that every leaf is a terminal, i.e., that $V_{MC} = V \setminus \{r\}$. We define the profits of pairs of terminals $(u, v) \in E_{GPMC}$ to be

$$p_{uv} = \begin{cases} |E_{GPMC}| & \text{if } (u,v) \in E_{MC}, \\ 1 & \text{otherwise.} \end{cases}$$

We set the target profit to be $P = |E_{MC}||E_{GPMC}|$.

For any $w \geq 0$, we prove that there exists a solution of size w to MC if and only if there exists a solution of size w and profit at least P for GPMC.

($\Rightarrow$ direction) Suppose that a solution to an instance of MC is a set of edges E^* of size w. Then, in the associated instance of GPMC, E^* disconnects each pair of terminals that has a profit $|E_{GPMC}|$, therefore the profit retrieved is at least $|E_{MC}||E_{GPMC}|$.

($\Leftarrow$ direction) Suppose that a set of edges E^* of size w is a solution to an instance of GPMC, i.e., that it disconnects pairs of terminals in H_{GPMC} with a total profit at least P. Suppose now that there exists a pair of terminals $(u, v) \in E_{MC}$ that is not disconnected in G by the removal of E^*. Then, even if every other pair of terminals in E_{GPMC} is disconnected, the associated profit is

$$(|E_{MC}| - 1)|E_{GPMC}| + (|E_{GPMC}| - |E_{MC}|) = P - |E_{MC}|,$$

which is a contradiction. Therefore, in any solution E^* to this GPMC instance, all pairs of terminals in E_{MC} must be disconnected. Since the set of edges E^* has size w, it is a solution to the associated MC instance.

The encoding size of the additional input in GPMC is polynomial in the size of the MC instance, as at most $|V|^2$ profits of value $|E_{GPMC}| \leq |V|^2$ must be encoded, and the reduction itself can be performed in a time polynomial in the size of the MC instance. Since MC is **APX**-hard in stars with unit weights [16], the result follows. □

Intuitively, the proof of Theorem 13 shows that we can emulate an arbitrary H_{MC} for MC with a complete H_{GPMC} in GPMC using hierarchical profits.

5. Conclusion and open problems

We summarize our main results, and highlight two open questions:

- We prove that PMC is strongly **NP**-hard in directed stars, while it is tractable in rooted trees, and while in addition MC is tractable in directed trees. This is the first such case we are aware of: are there other cases where MC and PMC behave differently, with respect to their computational complexity?
- We prove that MC remains **APX**-hard even if $tw(G)+tw(H) = 2$, while it was known to be FPT w.r.t. $tw(G+H)$.
- We prove that, like MC, PMC is FPT w.r.t. $tw(G+H)$, and that GPMC, which is weakly **NP**-hard when $tw(G+H) = O(1)$, admits both a pseudopolynomial-time algorithm and an FPTAS in this case.

- We prove that PMC (and hence MC) can be solved in polynomial time when $tw(G + \bar{H}) = O(1)$, generalizing the tractability of MC when $tw(G) = O(1)$ and $tw(\bar{H}) = 0$. Can we also prove that (P)MC is not FPT w.r.t. $tw(G + \bar{H})$?
- We finally prove that, on the contrary, GPMC remains **APX**-hard even if $tw(G + \bar{H}) = 1$ and all profits are polynomially bounded.

References

[1] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[2] Cédric Bentz. On the complexity of the multicut problem in bounded tree-width graphs and digraphs. *Discrete Applied Mathematics*, 156:1908–1917, 2008.

[3] Cédric Bentz. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theoretical Computer Science*, 412(39):5325–5332, 2011.

[4] Cédric Bentz. A polynomial-time algorithm for planar multicuts with few source-sink pairs. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *IPEC*, Lecture Notes in Computer Science, pages 109–119. Springer, 2012.

[5] Cédric Bentz, Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Multicuts and integral multiflows in rings. *European Journal of Operational Research*, 196(3):1251–1254, 2009.

[6] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 459–468. ACM, 2011.

[7] Bugra Caskurlu, Vahan Mkrtchyan, Ojas Parekh, and K. Subramani. Partial vertex cover and budgeted maximum coverage in bipartite graphs. *SIAM J. Discrete Math.*, 31(3):2172–2184, 2017.

[8] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.

[9] Sunil Chopra and Mendu R. Rao. On the multiway cut polyhedron. *Networks*, 21:51–89, 1991.

[10] Marie-Christine Costa and Alain Billionnet. Multiway cut and integer flow problems in trees. *Electronic Notes in Discrete Mathematics*, 17:105–109, 2004.

[11] Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162:55–69, 2005. (An erratum can be found in: C. Bentz, M.-C. Costa, L. Létocart, F. Roupin. Erratum to "Minimal multicut and maximal integer multiflow: A survey" [European Journal of Operational Research 162 (1) (2005) 55–69]. European Journal of Operational Research 177 (2007) 1312).

[12] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. Elsevier Science Publishers, 1990.

[13] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal of Computing*, 23(4):864–894, 1994.

[14] Éric Colin de Verdière. Multicuts in planar and bounded-genus graphs with bounded number of terminals. *Algorithmica*, 78(4):1206–1224, 2017.

[15] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal of Computing*, 25:235–251, 1996.

[16] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[17] Georgii Gens and Evgenii Levner. Complexity of approximation algorithms for combinatorial problems: a survey. *SIGACT News*, 12(3):52–65, 1980.

[18] Daniel Golovin, Viswanath Nagarajan, and Mohit Singh. Approximating the k-multicut problem. In *SODA*, pages 621–630. ACM Press, 2006.

[19] Georg Gottlob and Stephanie T. Lee. A logical approach to multicut problems. *Information Processing Letters*, 103(4):136–141, 2007.

[20] Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for vertex multicut in interval and bounded treewidth graphs. *European Journal of Operational Research*, 186:542–553, 2008.

[21] Anupam Gupta. Improved results for directed multicut. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 454–455, 2003.

[22] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[23] Ton Kloks. Treewidth: Computations and approximations. *Lecture Notes in Computer Science*, 842, 1994.

[24] Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59(4):489–509, 2011.

[25] Asaf Levin and Danny Segev. Partial multicuts in trees. *Theoretical Computer Science*, 369(1-3):384–395, 2006.

[26] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 469–478. ACM, 2011.

[27] Julián Mestre. Lagrangian relaxation and partial cover (extended abstract). In Susanne Albers and Pascal Weil, editors, *STACS '08*, volume 1 of *LIPIcs*, pages 539–550. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[28] Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut algorithms via tree decompositions. In Tiziana Calamoneri and Josep Diaz, editors, *Algorithms and Complexity*, volume 6078 of *Lecture Notes in Computer Science*, pages 167–179. Springer Berlin / Heidelberg, 2010.

[29] Kirk Pruhs and Gerhard. J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151 – 156, 2007.

[30] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[31] David B. Shmoys. *Cut problems and their application to divide-and-conquer*, chapter 5, pages 192–234. in D. Hochbaum (ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997.

[32] Éva Tardos and Vijay V. Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and $k_{r,r}$-free graphs. *Information Processing Letters*, 47:77–80, 1993.

[33] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[34] Mihalis Yannakakis, Paris Kanellakis, Stavros Cosmadakis, and Christos Papadimitriou. Cutting and partitioning a graph after a fixed pattern. In Josep Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 712–722. Springer Berlin / Heidelberg, 1983.