Multicuts in Unweighted Graphs and Digraphs with Bounded Degree and Bounded Tree-Width

Gruia Călinescu * Cristina G. Fernandes † Bruce Reed ‡

Abstract

The Multicut problem can be defined as: given a graph G and a collection of pairs of distinct vertices (s_i, t_i) of G, find a minimum set of edges of G whose removal disconnects each s_i from the corresponding t_i. Multicut is known to be NP-hard and Max SNP-hard even when the input graph is restricted to being a tree. The main result of the paper is a polynomial-time approximation scheme (PTAS) for Multicut in unweighted graphs with bounded degree and bounded tree-width. That is, for any $\epsilon > 0$, we present a polynomial-time $(1+\epsilon)$-approximation algorithm. In the particular case when the input is a bounded-degree tree, we have a linear-time implementation of the algorithm. We also provide some hardness results: we prove that Multicut is still NP-hard for binary trees and that it is Max SNP-hard if we drop any of the three conditions (unweighted, bounded-degree, bounded tree-width). Finally we show that some of these results extend to the vertex version of Multicut and to a directed version of Multicut.

1 Introduction

Multicommodity Flow problems have been intensely studied for decades [FF58, H63, GVY96, KARR90, LR88, PT93] because of their practical applications and also of the appealing hardness of several of their versions. The fractional version of a Multicut problem is the dual of a Multicommodity Flow problem and, therefore, Multicut is of similar interest [DJPSY94, GVY96, GVY97, KARR90, TV93].

The **Weighted Multicut** is the following problem: given an undirected graph G, a weight function w on the edges of G, and a collection of k pairs of distinct vertices (s_i, t_i) of G, find a minimum weight set of edges of G whose removal disconnects each s_i from the corresponding t_i.

The particular case in which $k = 1$ is characterized by the famous *Max-Flow Min-Cut Theorem* [FF56], and is solvable in strongly polynomial time [D70]. For $k = 2$, a variant of the Max-Flow Min-Cut Theorem holds [H63, I78] and Multicut is of Multicut. solvable in polynomial time. For $k \geq 3$, the problem is NP-hard [DJPSY94].

Since many variants of the Weighted Multicut are known to be NP-hard, we search for efficient approximation algorithms. The *performance ratio of an approximation algorithm A* for a minimization problem is the supremum, over all possible instances I, of the ratio between the weight of the output of A when running on I and the weight of an optimal solution for I. We say A is an *α-approximation algorithm* if its performance ratio is at most α. The smaller the performance ratio, the better.

The best known performance ratio for Weighted Multicut in general graphs is $O(\log k)$ [GVY97]. Im-

*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106. E-mail: calinesc@cs.iit.edu.

†Department of Computer Science, University of São Paulo, Brazil. E-mail: cris@ime.usp.br. Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106, FAPESP (Proc. 96/04505–2) and PRONEX (MCT/FINEP)(Proj. 107/97).

‡CNRS - Paris, France and Department of Computer Science. E-mail: reed@cs.toronto.edu. Research partially done at University of São Paulo (Brazil) and supported in part by FAPESP (Proc. 96/12111–4) and PRONEX (MCT/FINEP)(Proj. 107/97).

portant research has been done for improving the performance ratio when the input graph G belongs to special classes of graphs. For planar graphs, Tardos and Vazirani [TV93], see also [KARR90], give an approximate Max-Flow Min-Cut theorem and an algorithm with a constant performance ratio.

The case when the input graph is restricted to a tree has been studied in [GVY96]. Unweighted Multicut problem (in which $w(e) = 1$ for all edges e of G) restricted to stars (trees of height one) is equivalent (including performance ratio) to Minimum Vertex Cover, by Proposition 1 in [GVY96]. It follows that Unweighted Multicut restricted to stars is NP-hard and Max SNP-hard. In fact, getting a performance ratio better than two seems very hard, since getting a performance ratio better than two for Minimum Vertex Cover remains a challenging open problem [S95]. Garg, Vazirani and Yannakakis give an algorithm in [GVY96] with a performance ratio of two for the Weighted Multicut problem in trees. Note that the integral unweighted Multicommodity Flow problem in trees is solvable in polynomial time [GVY96].

We find useful two variations of the Multicut problem. The **Vertex Multicut** problem is: given an undirected graph G and a collection of k pairs of distinct nonadjacent vertices (s_i, t_i) of G called *terminals*, find a minimum set of nonterminal vertices whose removal disconnects each s_i from the corresponding t_i. The **Unrestricted Vertex Multicut** problem is: given an undirected graph G and a collection of k pairs of vertices (s_i, t_i) of G called *terminals*, find a minimum set of vertices whose removal disconnects each s_i from the corresponding t_i. (Note that in this variation, terminals might be removed.) Observe that Vertex Multicut is at least as hard (including performance ratio) as Unrestricted Vertex Multicut. From an instance of Unrestricted Vertex Multicut we can obtain an instance of Vertex Multicut by adding, for each s_i, a new vertex s'_i adjacent only to s_i, and, for each t_i, a new vertex t'_i adjacent only to t_i. Each pair (s_i, t_i) is substituted by the new pair (s'_i, t'_i). Solving Vertex Multicut in this instance is equivalent to solving Unrestricted Vertex Multicut in the original instance.

Both Vertex Multicut and Unrestricted Vertex Multicut might be of interest on their own. Garg, Vazirani and Yannakakis considered the weighted version of Vertex Multicut, and proved that their algorithm in [GVY97] achieves a performance ratio of $O(\log k)$ for the weighted version of Vertex Multicut in general graphs.

From now on, we refer to Multicut as **Edge Multicut**, to avoid confusion. Let us mention some results we obtained for Vertex Multicut and Unrestricted Vertex Multicut. We have a proof that Vertex Multicut is NP-hard in bounded-degree trees. Unrestricted Vertex Multicut is easier: it is polynomially solvable in trees, but it becomes NP-hard in bounded-degree series-parallel graphs.

The tree-width notion (first introduced by Robertson and Seymour [RS84, RS86]), seems to often capture a property of the input graph which makes hard problems easy. Various NP-hard problems, like Clique or Coloring, have a polynomial-time algorithm (linear time in fact) if the input graph has bounded tree-width (see for example [AL91]). We will present the formal definition of tree-width in Section 2.

Bounded tree-width can also be used to obtain good approximation algorithms for those problems that remain NP-hard even if restricted to graphs of bounded tree-width. In our case, Unrestricted Vertex Multicut is NP-hard in graphs of tree-width at most two, since this class of graphs coincides with the series-parallel graphs (see for example [vL90]). A PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1 + \epsilon$. We give a straightforward PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width.

We present an approximation-ratio preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. If the Edge Multicut instance graph has bounded degree and bounded tree-width, the Unrestricted Vertex Multicut instance obtained by the reduction has bounded tree-width. Combining the reduction with the PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width, we obtain a PTAS for Unweighted Edge Multicut in graphs with bounded degree and bounded tree-width. This is the main result of the paper. Note that, according to [GJ79, page 140, Theorem 6.8], a FPTAS cannot exist for this problem, unless P=NP. (A FPTAS consists of, for each $\epsilon > 0$, an algorithm for the problem with a performance ratio of at most $1+\epsilon$ and running-time polynomial in the size of the input and $1/\epsilon$.)

We also have a linear-time implementation of our PTAS for Edge Multicut in bounded-degree trees. The running time of our implementation is $O((n+k)\lceil 1/\epsilon\rceil d^{d^{\lceil 1/\epsilon\rceil+2}})$, where n is the number of vertices of the tree, k is the number of (s_i, t_i) pairs, d is the maximum degree of the tree and $1+\epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n+k)$.

Most of our results hold also for the following version of multicut in directed graphs (here called *digraphs*), introduced by Klein, Plotkin, Rao and Tardos [KPRT97]. We call **Directed Edge Multicut** the following problem: given a digraph D and a collection of k pairs of distinct pairs of vertices (s_i, t_i) of D, find a minimum set of edges whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices. In other words, for all i, either all paths from s_i to t_i *or* all paths from t_i to s_i must be broken.

Klein et al. [KPRT97] presents a $O(\log^2 k)$-approximation algorithm for edge-weighted Directed Edge Multicut in general digraphs. We give a PTAS for Directed Edge Multicut in unweighted digraphs with bounded degree and bounded tree-width. It is worth mentioning that we mean here the concept of tree-width in digraphs, introduced by Johnson, Robertson, Seymour, and Thomas [JRST99], and which differs from the one in undirected graphs. For example, the class of digraphs of tree-width zero consists of all acyclic digraphs.

From the hardness point of view, we show that Edge Multicut is still NP-hard for binary (degree bounded by three) trees. Thus on the class of graphs of bounded-degree and bounded-tree-width, which contains binary trees, Edge Multicut is easier (there is a PTAS) than on general graphs, yet still NP-hard. Identifying this class is the main theoretical result of this paper.

Also, we show that Directed Edge Multicut is NP-hard in digraphs with tree-width one and maximum in and out degree three. Other hardness results indicate why we cannot eliminate any of the three restrictions—unweighted, bounded degree and bounded tree-width—on the input graph and still obtain a PTAS. It is known [ALMSS92] that for a Max SNP-hard problem, unless P=NP, no PTAS exists. We have already seen that Unweighted Edge Multicut is Max SNP-hard in stars [GVY96], so letting the input graph have unbounded degree makes the problem harder. We show that Weighted Edge Multicut is Max SNP-hard in binary trees, therefore letting the input graph be weighted makes the problem harder. Finally, we show that Unweighted Edge Multicut is Max SNP-hard if the input graphs are walls. Walls, to be formally defined in Section 6, have degree at most three. We conclude that letting the input graph have unbounded tree-width makes the problem significantly harder.

In Section 2 we present the polynomial-time algorithm for Unrestricted Vertex Multicut in trees and the polynomial-time approximation scheme for Unrestricted Vertex Multicut in bounded-tree-width graphs. In Section 3, we show the approximation-preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. In Section 4 we present the linear-time implementation of a polynomial-time approximation scheme for bounded-degree trees. Section 5 treats the directed case. Finally, in Section 6 we present our hardness results.

2 Algorithms for Unrestricted Vertex Multicut

In this section we concentrate on Unrestricted Vertex Multicut. We present a polynomial-time algorithm for trees and a PTAS for graphs with bounded tree-width. Let us start defining tree-width.

Let G be a graph and Θ be a pair $(T, (X_w)_{w\in V(T)})$, which consists of a tree T and a multiset whose elements X_w, indexed by the vertices of T, are subsets of $V(G)$. For a vertex v of G, we denote by F_v the subgraph of T induced by those vertices w of T for which X_w contains v. Then Θ is called a *tree decomposition of* G if it satisfies the two conditions below:

(1) For every edge $e = xy$ of G, there is a vertex w of T such that $\{x, y\} \subseteq X_w$;

(2) For every vertex v of G, the subgraph F_v of T is a tree.

The *width of* Θ is the maximum, over all vertices w of T, of $|X_w| - 1$, and the *tree-width of* G, denoted by $tw(G)$, is the minimum of the widths of all tree decompositions of G.

Consider an instance of Unrestricted Vertex Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs (s_i, t_i) of vertices of G. We say a pair (s_i, t_i) in $\mathcal{C}$ is *disconnected by a set* $S \subseteq V$ if s_i or t_i is in S or s_i is disconnected from t_i in the subgraph of G induced by $V - S$. A set S is a *solution for* G if S disconnects all pairs (s_i, t_i) in $\mathcal{C}$. If S has minimum size (i.e., minimum number of vertices), then S is an *optimal solution for* G.

Now, let us describe the polynomial-time algorithm for trees. The input of the algorithm is a tree T and a set $\mathcal{C}$ of pairs (s_i, t_i) of vertices of T.

Consider the tree T rooted at an arbitrary vertex and consider also an arbitrary ordering of the children of each vertex (so that we can talk about postorder).

Algorithm:

Input: a tree T.

Start with $S = \emptyset$.

Call a pair (s_i, t_i) in $\mathcal{C}$ *active* if it is not disconnected by S.

Traverse the tree in postorder.

When visiting vertex v, if v is the least common ancestor of some active pair (s_i, t_i) in $\mathcal{C}$, then insert v into S.

Output S.

Clearly the following invariant holds: all non-active pairs in $\mathcal{C}$ are disconnected by S. A pair in $\mathcal{C}$ that becomes non-active does not go back to active since we never remove vertices from S. At the end of the algorithm, no pair in $\mathcal{C}$ is active, meaning that S is a solution for the problem. Besides, it is not hard to see that the algorithm can be implemented in polynomial time. The solution produced is optimum. Indeed, we associate to each vertex v included in S an active pair (s_i, t_i) having v as least common ancestor, and the paths connecting these pairs in T are vertex disjoint. So any solution must contain a vertex in each of these vertex-disjoint paths.

2.1 Bounded-Tree-Width Graphs

Next we present a PTAS for Unrestricted Vertex Multicut in graphs with bounded tree-width. Recall that a PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1 + \epsilon$. Let us describe such an algorithm.

The input of our algorithm is a graph $G = (V, E)$, a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G, and a set $\mathcal{C}$ of pairs of vertices of G.

Given a subgraph G' of G, denote by $\mathcal{C}(G')$ the set of pairs in $\mathcal{C}$ whose two vertices are in G', and by $G \setminus G'$ the subgraph of G induced by $V(G) \setminus V(G')$. For the description of the algorithm, all the instances we mention are on a subgraph G' of G and the set of pairs to be disconnected is $\mathcal{C}(G')$. So we will drop $\mathcal{C}(G')$ of the notation and refer to an instance only by the graph G'. Denote by $opt(G')$ the size (i.e., the number of vertices) of an optimal solution for G'.

Root the tree T (of the given tree decomposition) at a vertex r and consider an arbitrary ordering of the children of each vertex of T. For a vertex u of T, let $T(u)$ be the subtree of T rooted at u. Let $G(u)$ be the subgraph of G induced by the union of all X_w, $w \in V(T(u))$. Let $t = \lceil (tw(G) + 1)/\epsilon \rceil$.

Here is a general description of the algorithm: label the vertices of T in postorder. Find the lowest labeled vertex u such that an optimal solution for $G(u)$ has at least t vertices. If there is no such vertex, let

u be the root. Find an approximate solution S_u for $G(u)$ such that $|S_u| \leq (1+\epsilon)\,opt(G(u))$ and $X_u \subseteq S_u$. If u is the root of T, then output S_u. Otherwise, let $G' = G \setminus G(u)$ and let $\Theta' = (T', (X'_w)_{w \in V(T')})$ be the tree decomposition of G' where $T' = T \setminus T(u)$ and $X'_w = X_w \setminus V(G(u))$, for all $w \in V(T')$. Recursively get a solution S' for G'. Output $S = S' \cup S_u$.

Next we present a detailed description of the algorithm. It works in iterations. Iteration i starts with a subgraph G^{i-1} of G, a tree decomposition $\Theta^{i-1} = (T^{i-1}, (X_w^{i-1})_{w \in V(T^{i-1})})$ of G^{i-1} with T^{i-1} rooted at r, and a set S^{i-1} of vertices of G. Initially, $G^0 = G$, $\Theta^0 = \Theta$, $S^0 = \emptyset$ and $i = 1$. The algorithm halts when $G^{i-1} = \emptyset$. When G^{i-1} is nonempty, the algorithm starts calling a procedure *Get* (u, A), which returns a vertex u of T^{i-1} and a solution A for $G^{i-1}(u)$ such that $|A| \leq (1+\epsilon)\,opt(G^{i-1}(u))$ and $X_u^{i-1} \subseteq A$. Then the algorithm starts a new iteration with $G^i = G^{i-1} \setminus G^{i-1}(u)$, $\Theta^i = (T^i, (X_w^i)_{w \in V(T^i)})$, where $T^i = T^{i-1} \setminus T^{i-1}(u)$ and $X_w^i = X_w^{i-1} \setminus V(G^{i-1}(u))$, for all $w \in V(T^i)$, and $S^i = S^{i-1} \cup A$.

Algorithm

$G^0 \leftarrow G$;
$\Theta^0 \leftarrow \Theta$;
$S^0 \leftarrow \emptyset$;
$i \leftarrow 1$;
while $G^{i-1} \neq \emptyset$ do
    Get (u_i, A^i);   /* $|A^i| \leq (1+\epsilon)\,opt(G^{i-1}(u_i))$ and $X_{u_i}^{i-1} \subseteq A^i$ */
    $G^i = G^{i-1} \setminus G^{i-1}(u_i)$;
    $T^i = T^{i-1} \setminus T^{i-1}(u_i)$;
    $X_w^i = X_w^{i-1} \setminus V(G^{i-1}(u_i))$, for each $w \in V(T^i)$;
    $S^i = S^{i-1} \cup A^i$;
    $i \leftarrow i + 1$;
endwhile;
$f = i - 1$;
output S^f.

We will postpone the description of *Get* (u, A) and, for now, assume that it works correctly and in polynomial time. The next lemma [RS84] states a property of tree decompositions that we will use later.

Lemma 2.1 *Consider a graph G and a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G. Let u, t and v be vertices of T with t on the path from u to v in T. Let x be a vertex in X_u and y be a vertex in X_v. Then any path in G from x to y contains a vertex of X_t.*

Next we prove that the output of the algorithm is in fact a solution.

Lemma 2.2 *S^f is a solution for G.*

Proof. Let (s, t) be a pair in $\mathcal{C}$ and P be a path in G from s to t. We need to show that there is a vertex of P in S^f. Note that the vertex sets $V(G^{i-1}(u_i))$ define a partition of $V(G)$.

Let i be such that s is in $G^{i-1}(u_i)$. If all vertices of P lie in $G^{i-1}(u_i)$ then, in particular, both s and t are in $G^{i-1}(u_i)$, which means $(s, t) \in \mathcal{C}(G^{i-1}(u_i))$. Since S^f contains a solution for $G^{i-1}(u_i)$, S^f must contain a vertex of P.

If not all vertices of P lie in $G^{i-1}(u_i)$, let y be the first vertex of P that does not lie in $G^{i-1}(u_i)$. If y is in $G^{i-1} \setminus G^{i-1}(u_i)$ then, by Lemma 2.1, there is a vertex of $X_{u_i}^{i-1}$ in the segment of P from s to y. Since $X_{u_i}^{i-1} \subseteq S^f$, there is a vertex of P in S^f. If y is not in $G^{i-1} \setminus G^{i-1}(u_i)$, then y is not in G^{i-1}. This means y is in $G^{j-1}(u_j)$, for some $j < i$. Moreover, s is in $G^{j-1} \setminus G^{j-1}(u_j)$ (because this is a supergraph of G^{i-1}). Again by Lemma 2.1, there is a vertex of $X_{u_j}^{j-1} \subseteq S^f$ in P, concluding the proof of the lemma. ■

The next lemma proves that the performance ratio of the algorithm is at most $1 + \epsilon$.

Lemma 2.3 $|S^f| \leq (1+\epsilon)opt(G)$.

Proof. We have that

$$|S^f| \;=\; \sum_{i=1}^{f} |A^i| \;\leq\; \sum_{i=1}^{f} (1+\epsilon)\,opt(G^{i-1}(u_i)) \;=\; (1+\epsilon)\sum_{i=1}^{f} opt(G^{i-1}(u_i)) \;\leq\; (1+\epsilon)\,opt(G),$$

because the subgraphs $G^{i-1}(u_i)$ are vertex disjoint. ■

Now we proceed with the description of a straightforward polynomial-time implementation of *Get* (u, A).

Search the vertices of the tree T^{i-1} in postorder. Stop if the vertex u being visited is either the root or is such that $opt(G^{i-1}(u)) \geq t$. Let us show how we check whether $opt(G^{i-1}(u)) \geq t$ in polynomial time.

If we are searching vertex u, it is because all children of u have been searched and have an optimal solution with less than t vertices. Compute an optimal solution for each child v of u. This can be done in $O(n^{t+1})$ time by brute force: check all subsets of $G(v)$ of size at most t. The time is polynomial - since $t = \lceil (tw(G)+1)/\epsilon \rceil$ is fixed. Let s be the sum of the sizes of the solutions for the children of u.

Let us show that the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$. We do this by presenting a solution B for $G^{i-1}(u)$ of size at most $s + tw(G) + 1$. The set B is the union of X_u and an optimal solution for $G^{i-1}(v)$, for each child v of u. Thus $|B| \leq |X_u| + s \leq tw(G) + 1 + s$. Now we must prove that B is in fact a solution for $G^{i-1}(u)$. Let (s, t) be a pair in $\mathcal{C}(G^{i-1}(u))$ and P be a path in $G^{i-1}(u)$ from s to t. We need to show that there is a vertex of P in B. If there is a vertex of P in X_u, then clearly B contains a vertex of P. If, on the other hand, P contains no vertex of X_u, we must have all vertices of P in the same $G^{i-1}(v)$, for some child v of u, by Lemma 2.1. But B contains a solution for $G^{i-1}(v)$. Therefore, B contains a vertex of P. This completes the proof that B is a solution for $G^{i-1}(u)$, and so the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$.

Now, let us proceed with the description of *Get* (u, A). If $s < t$, then $opt(G(u)) \leq s + tw(G) + 1 < t + tw(G) + 1$, and we can compute in polynomial time an optimal solution A' for $G(u)$ (by brute force: in $O(n^{t+tw(G)+2})$ time, which is polynomial since $t = \lceil (tw(G)+1)/\epsilon \rceil$). If $|A'| < t$ then proceed to the next vertex in postorder. If $|A'| \geq t$, then we output u and the set $A = A' \cup X_u$. Note that in fact $opt(G^{i-1}(u)) = |A'| \geq t$, $X_u \subseteq A$ and $|A| \leq opt(G^{i-1}(u)) + (tw(G)+1) \leq opt(G^{i-1}(u)) + t\epsilon \leq (1+\epsilon)opt(G^{i-1}(u))$, as desired. On the other hand, if $s \geq t$, then $t \leq s \leq opt(G^{i-1}(u)) \leq s + tw(G) + 1 \leq s + t\epsilon \leq opt(G^{i-1}(u)) + opt(G^{i-1}(u))\epsilon = (1+\epsilon)opt(G^{i-1}(u))$. Thus B is a solution for $G^{i-1}(u)$ of size at most $s + tw(G) + 1 \leq (1+\epsilon)opt(G^{i-1}(u))$ that can be computed in polynomial time. Moreover, $X_u \subseteq B$. So in this case we output u and $A = B$. This finishes the description of *Get* (u, A).

3 Edge Multicut

In this section we show that Edge Multicut can be reduced to Unrestricted Vertex Multicut by a reduction that preserves approximability.

The reduction has the following property. If the instance of Edge Multicut is a graph with bounded degree and bounded tree-width, then the corresponding instance of Unrestricted Vertex Multicut has bounded tree-width.

Given a graph $G = (V, E)$, the *line graph of* G is the graph whose vertex set is E and such that two of its vertices (edges of G) are adjacent if they share an endpoint in G. In other words, the line graph of G is the graph (E, L), where $L = \{ef : e, f \in E$ and e and f have a common endpoint$\}$.

Consider an instance of Edge Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of G. Let us describe the corresponding instance of Unrestricted Vertex Multicut. The input graph for Unrestricted Vertex Multicut is the line graph of G, denoted by G'. Now let us describe the set of pairs of vertices of G'. For each pair (s,t) in $\mathcal{C}$, we have in $\mathcal{C}'$ all pairs (e,f) such that e has s as endpoint and f has t as endpoint.

Clearly G' can be obtained from G in polynomial time. Note that $\mathcal{C}'$ has at most $k\Delta^2$ pairs, where $k = |\mathcal{C}|$ and Δ is the maximum degree of G. Also $\mathcal{C}'$ can be obtained from G and $\mathcal{C}$ in polynomial time.

The following theorem completes the reduction.

Theorem 3.1 *S is a solution for Edge Multicut in G if and only if S is a solution for Unrestricted Vertex Multicut in G'.*

Proof. Consider a solution S of Edge Multicut in G, that is, a set S of edges of G such that any pair in $\mathcal{C}$ is disconnected in $(V, E-S)$. Note that $S \subseteq E(G) = V(G')$. Let us verify that the removal of S from G' disconnects all pairs in $\mathcal{C}'$. For any pair (e,f) in $\mathcal{C}'$, there are s and t in $V(G)$ such that s is an endpoint of e, t is an endpoint of f and the pair (s,t) is in $\mathcal{C}$. Moreover, a path P' in G' from e to f corresponds to a path P in G from s to t whose edges are a subset of the vertices in P' (which are edges of G). Since S is a solution of Edge Multicut in G, there must be an edge of P in S, which means that there is a vertex of P' in S. Hence S is a solution for Unrestricted Vertex Multicut in G'.

Conversely, let S be a solution for Unrestricted Vertex Multicut in G', that is, S is a set of edges of G whose removal from G' disconnects all pairs of vertices of G' in $\mathcal{C}'$. Let (s,t) be a pair in $\mathcal{C}$, and P a path in G from s to t. (Recall that, by the description of Edge Multicut, $s \neq t$.) Let e be the first edge of P and f the last one. Clearly s is incident to e, and t to f. Thus (e,f) is a pair in $\mathcal{C}'$. Corresponding to P, there is a path P' in G' from e to f containing as vertices all edges of P. Since S is a solution for Unrestricted Vertex Multicut in G' and (e,f) is in $\mathcal{C}'$, S must contain a vertex of P'. Therefore there is an edge of P in S, which implies that S is a solution of Edge Multicut in G. ■

The next lemma shows the previously mentioned property of this reduction.

Lemma 3.2 *If G has bounded degree and bounded tree-width, then the line graph of G has bounded tree-width.*

Proof. Denote by G' the line graph of G. Let us present a tree decomposition of G' whose tree-width is at most $(tw(G)+1)\Delta$, where Δ is the maximum degree of G.

Let $\Theta = (T, (X_u)_{u \in V(T)})$ be a tree decomposition of G of width $tw(G)$. For each $u \in V(T)$, let Y_u be the set of edges of G incident to some vertex in X_u. First let us prove that $\Theta' = (T, (Y_u)_{u \in V(T)})$ is a tree decomposition of G'. For this, given an edge e of G, denote by T_e the subgraph of T induced by those vertices in T for which Y_u contains e. We shall prove that (1) any edge h of G' has both endpoints in Y_u, for some u in $V(T)$; and (2) that T_e is a tree for any edge e of G'.

The endpoints of an edge h of G' are two edges e and f of G with a common endpoint, say, v. But $v \in X_u$ for some $u \in V(T)$. This implies that both e and f belong to Y_u, proving (1). For (2), let e be a vertex of G', that is, an edge $e = xy$ of G. For any u such that $e \in Y_u$, we must have that either $x \in X_u$ or $y \in X_u$. Therefore $T_e = T_x \cup T_y$. We know that the subgraphs T_x and T_y of T are subtrees of T. Moreover, T_x and T_y have a vertex in common, because both x and y belong to the same X_u, for some $u \in V(T)$. Hence T_e is a subtree of T. This completes the proof that Θ' is a tree decomposition of G'.

To verify that the width of Θ' is at most $(tw(G)+1)\Delta$, just note that $|Y_u| \leq |X_u|\Delta$, for all $u \in V(T)$. ■

The next corollary is a consequence of the previous reduction and the PTAS given in Section 2.1.

Corollary 3.3 *There is a PTAS for Edge Multicut in bounded-degree graphs with bounded tree-width.*

4 Linear-Time PTAS for Edge Multicut in Bounded-Degree Trees

The PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs can be modified into a PTAS for Edge Multicut in bounded-degree trees in a straightforward way. In this section we describe this modified PTAS and present a linear-time implementation for it.

Let $T = (V, E)$ be a tree and $\mathcal{C}$ be a set of pairs (s_i, t_i) of distinct vertices of T. We say a pair (s_i, t_i) in $\mathcal{C}$ is *disconnected by a set* $S \subseteq E$ if s_i is disconnected from t_i in $(V, E - S)$. A set S is a *solution for* T if S disconnects all pairs (s_i, t_i) in $\mathcal{C}$. If S has minimum size (i.e., minimum number of edges), then S is an *optimal solution for* T.

Now, for each fixed $\epsilon > 0$, let us describe the polynomial-time $(1 + \epsilon)$-approximation algorithm for bounded-degree trees. The input of the algorithm is a bounded-degree tree T and a set $\mathcal{C}$ of pairs (s_i, t_i) of vertices of T.

Consider the tree T rooted at an arbitrary vertex r and consider also an arbitrary ordering of the children of each vertex (so that we can talk about postorder and preorder).

As before, given a subtree T' of T, denote by $\mathcal{C}(T')$ the set of pairs in $\mathcal{C}$ whose two vertices are in T'. For the description of the algorithm, all the instances we mention are on a tree T' and the set of pairs to be disconnected is $\mathcal{C}(T')$, where T' is a subtree of T. So we will drop $\mathcal{C}(T')$ of the notation and refer to an instance only by the tree T'. Denote by $opt(T')$ the size (i.e., the number of edges) of an optimal solution for T'.

For a vertex u of T, let $T(u)$ be the subtree of T rooted at u. Denote by $T \setminus T(u)$ the subtree of T obtained from T after removing all vertices in $T(u)$. Let $t = \lceil 1/\epsilon \rceil$.

The algorithm is similar to the one for Unrestricted Vertex Multicut. It works in iterations. Iteration i starts with a subtree T^{i-1} of T also rooted at r, and a set S^{i-1} of edges of T. Initially, $T^0 = T$, $S^0 = \emptyset$ and $i = 1$. The algorithm halts when $T^{i-1} = \emptyset$. When T^{i-1} is nonempty, the algorithm starts calling a procedure *Get* (u, A), which returns a vertex u and an optimal solution A for $T^{i-1}(u)$ where either $u = r$ or $opt(T^{i-1}(u)) \geq t$. Then the algorithm starts a new iteration with $T^i = T^{i-1} \setminus T^{i-1}(u)$ and $S^i = S^{i-1} \cup A \cup \{uv\}$, where v is the parent of u.

The proof that this algorithm is a polynomial-time $(1 + \epsilon)$-approximation is analogous to the proof presented for the PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs.

4.1 Linear-Time Implementation

In this subsection we describe a $O((n+k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$ implementation of the algorithm described above. Recall that n is the number of vertices of the tree, k is the number of (s_i, t_i) pairs, d is the maximum degree of the tree and $1 + \epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n + k)$.

As in the straightforward implementation of *Get* (u, A) for Unrestricted Vertex Multicut, the procedure *Get* (u, A) finds the first qualified u in postorder. By qualified we mean that either $u = r$ or $opt(T^{i-1}(u)) \geq t = \lceil 1/\epsilon \rceil$. Also, while traversing T^{i-1} in postorder, we compute $opt(T^{i-1}(u))$ for each encountered vertex u.

The main idea to obtain this linear-time implementation is to compute $opt(T^{i-1}(u))$ faster. Since we compute these values in postorder, it is natural to use the values computed for the children of u. In fact, we compute a set of "relevant" optimal solutions for $T(u)$ (the technical definition of relevant appears in the next paragraph) using the sets of relevant optimal solutions for the children of u. Since *Get* (u, A) can return any optimal solution A, our algorithm will return only relevant optimal solutions. We will prove that the set of relevant optimal solutions is nonempty and show how to compute it for $T(u)$ using the sets of relevant optimal solutions for the children of u. Also we show that the number of relevant optimal solutions is small. One can use this idea alone to obtain a $O(kn^3 d^{d\lceil 1/\epsilon \rceil + 1})$-time implementation. This time is better than the time for the straightforward implementation since the degree of the polynomial in $n + k$ is fixed, that is, does not depend on ϵ or d. To further decrease the time dependence on $n + k$ we will show how to avoid recomputing optimum values and relevant optimal solutions when we switch from T^{i-1} to T^i and how to check faster if a set of edges is a solution.

We start with some definitions. Let T and $\mathcal{C}$ be an instance of the Multicut problem. Let v be a vertex of T and A a subset of edges of $T(v)$. We say that *a pair* $(s,t) \in \mathcal{C}$ *is open with respect to* v *and* A if exactly one of s and t is in $T(v)$ and the unique path in T from s to t does not contain any edge of A. We denote by $O_v(A)$ (T and $\mathcal{C}$ will be understood from the context) the set of open pairs with respect to v and A. Let v be a vertex of T. We say that an edge set B is a *relevant optimal solution for* $T(v)$ if, for any vertex u of $T(v)$, $B \cap E(T(u))$ is an optimal solution for $T(u)$. It is not obvious that a relevant optimal solution exists. In fact, this will follow from the following lemma.

Lemma 4.1 *Let* B *be an optimal solution for* $T(v)$*. Then there is a relevant optimal solution* A *for* $T(v)$ *such that* $O_v(A) \subseteq O_v(B)$*.*

Proof. By induction on the number of vertices in $T(v)$. If $T(v)$ has only one vertex (vertex v) then $B = \emptyset$ and we can take $A = \emptyset$.

Suppose now that $T(v)$ has more than one vertex and let $u_1, u_2, \ldots, u_r$ be the children of v. Let $B_j = B \cap E(T(u_j))$, for $j = 1, 2, \ldots, r$. We construct $A = \cup_{j=1}^{r} A_j$ in the following way: if $|B_j| = opt(T(u_j))$, by induction, we let A'_j be a relevant optimal solution for $T(u_j)$ such that $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$. If also $u_j v \in B$, we let $A_j = A'_j \cup \{u_j v\}$, otherwise $A_j = A'_j$. If $|B_j| > opt(T(u_j))$, let A'_j be any relevant optimal solution for $T(u_j)$. Such a relevant optimal solution exists by induction. Let $A_j = A'_j \cup \{u_j v\}$.

We now show that A is a solution for $T(v)$. Let (s,t) be a pair with both s and t in $T(v)$. A first case is when there is $j \in \{1, 2, \ldots, r\}$ such that both s and t are in $T(u_j)$. Then, since A'_j is a solution for $T(u_j)$, s and t are disconnected by A (meaning that s and t are not connected in $(V(T), E(T) - A)$).

A second case is when s is in $T(u_j)$ and $t = v$. The case when t is in $T(u_j)$ and $s = v$ is symmetric. If $u_j v \in A_j$, then s and t are disconnected by A. The only case in which $u_j v \notin A_j$ is when $u_j v \notin B$ and $|B_j| = opt(T(u_j))$. Since B is a solution for $T(v)$ and $u_j v \notin B$, there is an edge in B_j disconnecting s from u_j. This means that the pair (s,t) is not in $O_{u_j}(B_j)$, and since we selected A'_j such that $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$, (s,t) is not in $O_{u_j}(A'_j)$. Thus there is an edge in A'_j disconnecting s from u_j.

The third and last case is when s is in $T(u_j)$ and t is in $T(u_l)$, for some $l \neq j$. In this case there is an edge in B disconnecting s from v or there is an edge in B disconnecting t from v. We may assume without loss of generality that there is an edge in B disconnecting s from v. Using the same reasoning as in the second case, it follows that, because there is an edge in B disconnecting s from v, there is an edge in A disconnecting s from v. In conclusion, there is an edge in A disconnecting s from t.

We note that $|A| \leq |B|$. Since B is an optimal solution, we have $|A| = |B|$. We now show that A is a relevant optimal solution. Let u be a vertex in $T(v)$. If $u = v$, $A \cap E(T(v)) = A$ is an optimal solution for $T(v)$. If $u \neq v$, let $j \in \{1, 2, \ldots, r\}$ be such that u is a vertex in $T(u_j)$. Then $A \cap E(T(u)) = A'_j \cap E(T(u))$ is an optimal solution for $T(u)$ since A'_j is a relevant optimal solution for $T(u_j)$.

Finally we show that $O_v(A) \subseteq O_v(B)$. Let (s,t) be a pair in $O_v(A)$ and assume without loss of generality that s is a vertex in $T(v)$ and t is not a vertex in $T(v)$. If $s = v$, then, since $B \subseteq E(T(v))$, the pair (s,t) is in $O_v(B)$. If $s \neq v$, let $j \in \{1, 2, \ldots, r\}$ be such that s is a vertex in $T(u_j)$. Then (s,t) is a pair in $O_{u_j}(A'_j)$ (because $A'_j = A \cap E(T(u_j))$). Since (s,t) is a pair in $O_v(A)$, $u_j v \notin A$ and therefore $u_j v \notin A_j$. The only case in which $u_j v \notin A_j$ is when $u_j v \notin B$ and $|B_j| = opt(T(u_j))$. In this case, by the construction of A, $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$ and therefore the pair (s,t) is in $O_{u_j}(B_j)$. Using the fact that $u_j v \notin B$, we obtain that the pair (s,t) is in $O_v(B)$. ∎

In the following we describe a *generating procedure* which computes all relevant optimal solutions for $T(v)$. If v is a leaf, then we have exactly one relevant optimal solution, namely the empty set of edges. Otherwise, let $u_1, u_2, \ldots, u_r$ be the children of v. We assume that for all j, we have all the relevant optimal solutions for $T(u_j)$. The procedure is as follows: start with $l = 0$. Repeat the following process: try the union of all combinations of l edges incident to v and relevant optimal solutions for $T(u_1), T(u_2), \ldots, T(u_r)$. If some combination is a solution for $T(v)$, then write it down as a relevant optimal solution for $T(v)$. Obtain all such combinations for this same value of l and then halt. If none of the combinations leads to a solution for $T(v)$, we increment l and resume the process.

We now show that the above procedure correctly computes all relevant optimal solutions for $T(v)$. Any relevant optimal solutions for $T(v)$, if restricted to $T(u_j)$, is a relevant optimal solution for $T(u_j)$. Let A be any relevant optimal solution for $T(v)$. So A consists of a combination of relevant optimal solutions for $T(u_j)$, $j = 1, 2, \ldots, r$, and a set of q edges incident to v. All relevant optimal solutions have the same number of edges, and since any two have exactly the same number of edges in $E(T(u_j))$, for $j = 1, 2, \ldots, r$, it follows that all relevant optimal solutions have the same number of edges, q, incident to v. Also, for $l < q$, there cannot be any edge set B, which is the union of a combination of relevant optimal solutions for $T(u_1), T(u_2), \ldots, T(u_r)$ and l edges incident to v, and such that B is a solution for $T(v)$, since in this case $|B| < |A|$ and therefore A would not be an optimal solution. The correctness of the procedure above is now clear, as for $l < q$ the procedure will try all combinations but will not find any solution, and then, for $l = q$, it will output all relevant optimal solutions with q edges incident to v, and since all relevant optimal solutions have exactly q edges incident to v, it will output all relevant optimal solutions for $T(v)$.

From the procedure above, it follows by induction on the number of vertices in $T(v)$ that there are at most $d^{opt(T(v))}$ relevant optimal solutions for $T(v)$. Indeed, if v is a leaf, there is only one relevant optimal solution for $T(v)$. If v is not a leaf, using the notation from the paragraph above, $opt(T(v)) = q + opt(T(u_1)) + opt(T(u_2)) + \ldots + opt(T(u_r))$. By induction, there are at most $d^{opt(T(u_j))}$ relevant optimal solutions for $T(u_j)$. In conclusion, we need to try at most $\binom{r}{q} \prod_{j=1}^{r} opt(T(u_j))$ combinations and this number is at most $d^q \prod_{j=1}^{r} d^{opt(T(u_j))} = d^{opt(T(v))}$.

We are left with the implementation details on how to avoid recomputing values and relevant optimal solutions when we switch from T^{i-1} to T^i and how to check fast if a combination is a solution.

We represent the current tree T^i in the following way: we mark all vertices of T not in T^i. So when executing $T^i \leftarrow T^{i-1} \setminus T^{i-1}(u_i)$ in the algorithm, in fact we traverse $T^{i-1}(u_i)$ in preorder and mark each vertex. To traverse in preorder $T^{i-1}(u_i)$, we in fact traverse in preorder $T(u_i)$, but we stop and return immediately each time we meet an already marked vertex. The time for the traversal (and marking) of $T^{i-1}(u_i)$ is proportional to the number of vertices we mark plus the number of already marked vertices we encounter. Note that whenever we encounter an already marked vertex v during the traversal of $T^{i-1}(u_i)$, its parent u was not marked before the traversal of $T^{i-1}(u_i)$, but u is marked at the end of the traversal of $T^{i-1}(u_i)$. So the total number of times, over all i, we encounter an already marked vertex is $O(n)$. Since we mark a vertex exactly once, the total time, over all i, for these marking traversals is $O(n)$.

To compute the relevant optimal solutions for vertex u in T^{i-1}, we simply ignore the children of u which are marked. We will use the fact that, if for vertex v in T^{i-1}, the set of relevant optimal solutions is computed, and v is in T^i, then, in T^i, v has exactly the same set of relevant optimal solutions. Indeed, if v's set of relevant optimal solutions is computed in T^{i-1}, none of its descendant in T^{i-1} has optimum value at least t. In conclusion all descendants of v in T^{i-1}, including v itself, appear in T^i and therefore $T^{i-1}(v) = T^i(v)$. Since the set of relevant optimal solutions depends only on $T^{i-1}(v) = T^i(v)$, it follows that in T^i, v has exactly the same set of relevant optimal solutions. Also note that $opt(T^{i-1}(v)) < t$, since v appears in T^i. In conclusion we never have to recompute a set of relevant optimal solutions.

Note that, since T^i is a subtree of T, postorder in T^i equals postorder in T restricted to the vertices of T^i. It remains to show how to find quickly the next vertex of T^i in postorder whose set of relevant optimal solutions is not computed. The first such vertex of T^i (unless of course $T^i = \emptyset$) is the successor in postorder in T of u_i, and in general the next such vertex of T^i is actually the next vertex of T in postorder. This fact—that the next vertex of T^i in postorder, whose set of relevant optimal solutions is not computed, is simply the next vertex of T in postorder—follows from and is used to maintain the following invariant: the vertices of T for which the set of relevant optimal solutions (in some T^i) is computed is always the set of first j vertices of T in postorder, for some $0 \leq j \leq n$.

Finally, we describe how to check if a combination is a solution. In a preprocessing phase, in $O(n)$

[HT84], construct a data structure which allows us to find in constant time the lowest common ancestor (lca) of any two vertices u and v in T. Using this data structure, in time $O(n+k)$, construct for each vertex v, the list of (s,t) pairs such that $v = lca(s,t)$. Let k_v be the size of this list.

So, as above, let v be the vertex whose set of relevant optimal solutions we are now computing in T^i. Let $u_1, u_2, \ldots, u_r$ be the children of v in T^i. To obtain these children of v in T^i, we go through the list of the children of v in T and ignore the marked ones. We need to check whether an edge set B is a solution for $T^i(v)$, where B consists of some edges incident to v and $B \cap E(T^i(u_j))$ is a relevant optimal solution for $T^i(u_j)$. In conclusion, it is enough to check if B disconnects those pairs (s,t) with both s and t in T^i and having v as lowest common ancestor (all the other pairs are disconnected by $B \cap E(T^i(u_j))$, for $j = 1, 2, \ldots, r$). To do this, we go through the list of pairs having v as lowest common ancestor. For each pair (s,t) in the list, we first check if both s and t are unmarked. If one of them is marked we ignore this pair. Then we check for each edge of B if that edge is on the unique path from s to v. This can be accomplished in constant time per edge of B in the following way: for an edge of B with a deeper (in T) endpoint u, we check if $lca(u,s) = u$, and if so, we conclude that the edge disconnects s from v and therefore s from t. Similarly, we check for each edge of B if that edge is on the unique path from t to v. If none such edge is found, we conclude that B does not disconnect s from t. The total time spent for edge set B is $O(|B|(1+k_v))$.

Now let us prove that if the search reaches a vertex u, then $opt(T^{i-1}(u)) \leq dt$. We prove this by presenting a solution Y for $T^{i-1}(u)$ of size at most dt. The set Y is the union of the set of edges incident to u in T^{i-1} and an optimal solution for $T^{i-1}(v)$, for each child v of u. More specifically, let Y_0 be the set of edges incident to u in T^{i-1}, and let Y_j be an optimal solution for $T^{i-1}(v_j)$, where v_j is the j^{th} child of u. Because T^{i-1} has maximum degree at most d, $|Y_0| \leq d$ and u has at most d children. All of the at most d children of u were visited before u (because of the postorder). So for any child v_j of u, $|Y_j| = opt(T^{i-1}(v_j)) \leq t-1$ (otherwise the search would have stopped before it reaches u). Thus the set $Y = \cup_{i=0}^{|Y_0|} Y_j$ has size

$$|Y| = \sum_{i=0}^{|Y_0|} |Y_j| = |Y_0| + \sum_{i=1}^{|Y_0|} |Y_j| \leq d + d(t-1) = dt.$$

Therefore $|B| \leq dt$, and there are at most d^{dt} combinations we need to check for each l (from the generating procedure described on page 9). Thus, the total time spent computing the set of relevant optimal solutions for v in T^i is $O((1+k_v)dtd^{dt+1})$. Computing these sets of relevant optimal solutions dominates the time, so the total time of the algorithm is $O((n+k)dtd^{dt+1})$, since $\sum_v (1+k_v) = n+k$. Since $t = \lceil 1/\epsilon \rceil$, we obtain that the total time of the algorithm is $O((n+k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$, which is linear in the size of the problem for any fixed $\epsilon > 0$.

5 Multicuts in Digraphs

Using the same techniques we applied for undirected graphs, we can get a PTAS for Directed Edge Multicut in digraphs with bounded degree and bounded tree-width. Johnson et al. [JRST99] presented the following definition of tree-width in digraphs.

An *arborescence* is a digraph R such that R has a vertex r_0, called the root of R, with the property that for every vertex $r \in V(R)$ there is a unique directed walk from r_0 to r. Thus every arborescence arises from a tree by selecting a root, and directing all edges away from the root. If $r, r' \in V(R)$, we write $r' > r$ if $r' \neq r$ and there exists a directed walk in R with initial vertex r and terminal vertex r'. If $e \in E(R)$, we write $r' > e$ if either $r' = r$ or $r' > r$, where r is the head of e. We also write $e \sim r$ when e is incident with r.

Let D be a digraph and let $Z \subseteq V(D)$. The digraph obtained from D by deleting Z will be denoted by $D \setminus Z$. We say that a set $S \subseteq V(D)$ is *Z-normal* if there is no directed walk in $D \setminus Z$ with first and last vertex in S that uses a vertex of $D \setminus (Z \cup S)$.

An *arboreal decomposition of a digraph* D is a triple (R, X, W), where R is an arborescence, and $X = (X_e \subseteq V(D) : e \in E(R))$ and $W = (W_r \subseteq V(D) : r \in V(R))$ satisfy

(1) $(W_r : r \in V(R))$ is a partition of $V(D)$, and

(2) if $e \in E(R)$, then $\cup\{W_r : r \in V(R), r > e\}$ is X_e-normal.

The *width of* (R, X, W) is the least integer w such that $|\cup_{e\sim r} X_e \cup W_r| \leq w + 1$ for all $r \in V(R)$. The *tree-width of* D, denoted by $dtw(D)$, is the least integer w such that D has an arboreal decomposition of width w.

Consider the **Directed Unrestricted Vertex Multicut** problem: given a digraph D and a collection of k pairs of vertices (s_i, t_i) of D, called *terminals*, find a minimum set of vertices whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices.

As in the undirected case, we have an approximation-ratio preserving reduction from Directed Edge Multicut to Directed Unrestricted Vertex Multicut. If the digraph of the Directed Edge Multicut instance has bounded degree and bounded tree-width, the digraph of the Directed Unrestricted Vertex Multicut instance obtained by the reduction has bounded tree-width. Combining the reduction with a PTAS for Directed Unrestricted Vertex Multicut in digraphs of bounded tree-width, we obtain a PTAS for Directed Unweighted Edge Multicut in digraphs with bounded degree and bounded tree-width.

The PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs can be modified into a PTAS for Directed Unrestricted Vertex Multicut in bounded-tree-width digraphs in a straightforward way. Using the same notation modified to the directed setup, the following is a short description of the modification.

Label the vertices of R in postorder. Find the lowest labeled vertex r such that an optimal solution for $D(r)$—the subdigraph of D induced by the set $Z = \cup\{W_{r'} : r' \in V(R), r' \geq r\}$—has at least t vertices. If there is no such vertex, let r be the root of R. Let e be the edge of R entering r (if r is the root, let e undefined). As before, using the already computed optimal solutions for the children of r, find an approximate solution S_r for $D(r)$ such that $|S_r| \leq (1 + \epsilon)\mathit{opt}(D(r))$ and $X_e \cup W_r \subseteq S_r$. If r is the root of R, then output S_r. Otherwise, let $D' = D \setminus (D(r) \cup X_e)$ and let $\Theta' = (R', X', W')$ be the arboreal decomposition of D' where $R' = R \setminus R(r)$ and $X'_f = X_f \setminus (X_e \cup V(D(r)))$, for all $f \in E(R')$, and $W'_s = W_s \setminus X_e$, for all $s \in V(R')$. Recursively get a solution S' for D'. Output $S = S' \cup S_r$.

The set S produced is a solution for the Directed Unrestricted Vertex Multicut in D. Indeed if C is a directed circuit either entirely in $D(r) \cup X_e$ or entirely in D' and containing s and t for some $(s, t) \in \mathcal{C}$, then $V(C) \cap S \neq \emptyset$ (C either intersects S_r or S'). If C is a directed circuit with vertices in $D(r)$ (that is, in Z) and in D', containing s and t for some $(s, t) \in \mathcal{C}$, then C has a vertex in $X_e \subseteq S_r \subseteq S$ because Z is X_e-normal.

Now we proceed to the reduction, which is very different from the undirected reduction. Consider an instance of Directed Edge Multicut, that is, a digraph $D = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of D. Let us describe the corresponding instance of Directed Unrestricted Vertex Multicut.

The input graph for Directed Unrestricted Vertex Multicut is the following digraph D'. (To avoid confusion, we will refer to vertices of D' as nodes and to edges of D' as arcs.) The set of nodes of D' is E plus m copies $u_1, \ldots, u_m$ of each vertex $u \in V$, where m is the number of edges in D. There is an arc in D' from a copy of $u \in V$ to $e \in E$ if u is the tail of e, and there is an arc in D' from $e \in E$ to a copy of $u \in V$ if u is the head of e. These are all the arcs in D'. Note that D' is a bipartite digraph (edges of D in one side and copies of vertices of D in the other).

Now let us describe the set of pairs of vertices of D'. For each pair (s, t) in $\mathcal{C}$, we have in $\mathcal{C}'$ pairs (s_i, t_j) for all $1 \leq i, j \leq m$. Clearly D' can be obtained from D in polynomial time. Note that $\mathcal{C}'$ has km^2 pairs. Also $\mathcal{C}'$ can be obtained from D and $\mathcal{C}$ in polynomial time.

Lemma 5.1 *For any solution S' for the Directed Unrestricted Vertex Multicut in D' there is a solution*

$S \subseteq E(D)$ *such that* $|S| \leq |S'|$.

Proof. We may assume S' is a minimal solution for the Directed Unrestricted Vertex Multicut in D'. If $S' \subseteq E(D)$ then we take $S := S'$. If not, let $u \in V(D)$ and $1 \leq i \leq m$ be such that $u_i \in S'$. Then $S' - \{u_i\}$ is not a solution in D', which implies that there is a pair $(s,t) \in \mathcal{C}'$ and a directed circuit Q in $D' \setminus (S' - \{u_i\})$ containing s and t (because s and t are in the same strongly connected component). If $u_j \notin S'$ for some $1 \leq j \leq m$, the circuit obtained from Q by substituting u_i by u_j is a directed circuit in $D' \setminus S'$ and contains a pair in $\mathcal{C}'$, a contradiction. Thus $u_j \in S'$ for all $1 \leq j \leq m$. In this case, $|S'| \geq m = |E(D)|$ and we take $S := E(D)$. In both cases S is a solution in D' and $|S| \leq |S'|$, which completes the proof of the lemma. ■

Lemma 5.1 together with the following theorem, whose proof is similar to the proof of Theorem 3.1, completes the reduction.

Theorem 5.2 *$S \subseteq E(D)$ is a solution for Directed Edge Multicut in D if and only if S is a solution for Directed Unrestricted Vertex Multicut in D'.*

The next lemma shows that, if the instance of Directed Edge Multicut is a digraph with bounded degree and bounded tree-width, then the corresponding instance of Directed Unrestricted Vertex Multicut has bounded tree-width. For this, we need some notation. For a vertex v in a digraph D, we denote by $\delta^+_D(v)$ the set of edges in D with v as tail. For a set S of vertices of D, $\delta^+_D(S)$ is the set of edges with tail in S and head not in S.

Lemma 5.3 *If D has bounded degree and bounded tree-width, then D' has bounded tree-width.*

Proof. Let D be a digraph with bounded degree and bounded tree-width and D' be obtained from the reduction described above. Let us present an arboreal decomposition of D' whose tree-width is at most $(dtw(D)+1)(\Delta+1)$, where Δ is the maximum outdegree of D.

Let $\Theta = (R, X, W)$ be an arboreal decomposition of D of width $dtw(D)$. Let R' be an arborescence obtained from R by substituting each edge $e \in E(R)$ from a vertex r' to a vertex r in R by a directed path $P_e = \langle u_0, \ldots, u_m \rangle$ of length m, starting at $u_0 = r'$ and ending at $u_m = r$, where $u_1, \ldots, u_{m-1}$ are new vertices. Let f be the edge in $E(P_e)$ from u_{i-1} to u_i, $1 \leq i \leq m$. If $i = 1$, then let $Y_f := \delta^+_D(X_e)$, else let $Y_f := \delta^+_D(X_e \cup W_r)$. For $i = 1, \ldots, m$, let $W^i_r \subseteq V(D')$ be the set containing the i^{th} copy in D' of each vertex in W_r. Set $Z_{u_1} := W^1_r \cup \delta^+_D(W_r)$ and, for $i = 2, \ldots, m$, set $Z_{u_i} := W^i_r$. Let us prove that $\Theta' = (R', Y, Z)$ is an arboreal decomposition of D'.

First note that $(Z_u : u \in V(R'))$ is a partition of $V(D')$. Now let f be an edge of R'. We need to show that $Z' := \cup\{Z_u : u \in V(R'), u > f\}$ is Y_f-normal. Let $e \in E(R)$ be such that $f \in E(P_e)$. Let $x, y \in Z'$ be such that there is a directed path Q' in D' from x to y which uses a node in $V(D') - Z'$. Choose $x, y \in Z'$ so that Q' has minimum length. We need to show that $V(Q') \cap Y_f \neq \emptyset$. We analyze two cases.

If $x \notin E(D)$ then let $x' \in V(D')$ be the node following x in Q'. Note that $x' \in E(D) - Z'$, because of the bipartition of D' and the choice of x and y. This implies that f is not the first edge of the path P_e. Thus x is a copy of a vertex in W_r and $x' \in \delta^+_D(W_r) \subseteq Y_f$.

If $x \in E(D)$ then let Q be the path in D which starts at the tail of x and visits the vertices of D in the same order in which they (their copies in fact) appear in Q'. Let $Z \subseteq V(D)$ be the set of vertices with at least one copy in Z'. Note that $Z = \cup\{W_r : r \in V(R), r > e\}$. Both endpoints of Q are in Z. Indeed, the tail of any edge of D in Z' is also in Z' (because, by construction, if $a \in E(D) \cap Z_u$ then one copy of its tail is also in Z_u) so the first vertex of Q is in Z. Also the last vertex of Q' is not in $E(D)$ (for the same reason), and this implies that the last vertex of Q is in Z. Thus Q is a path in D between two vertices of Z. Since Θ is an arboreal decomposition of D, there is a vertex of X_e in Q. But this implies Q' has a node in Y_f, because $Y_f \supseteq \delta^+_D(X_e)$. This completes the proof that Θ' is an arboreal decomposition of D'.

Now we just need to verify that the width of Θ' is at most $(dtw(D)+1)(\Delta+1)$. For that, let $r' \in V(R')$. If r' is a vertex of R, then let $r = r'$, otherwise, r' was added when an edge e of R was

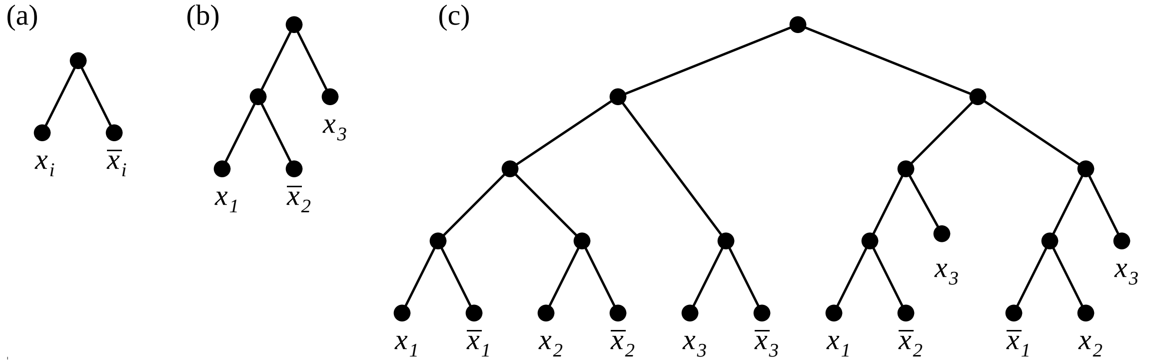

Figure 1: (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Tree T built for the instance $\Phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \overline{x}_2, x_3\}$ and $C_2 = \{\overline{x}_1, x_2, x_3\}$.

substituted by a path between its endpoints. Let r be the tail of e. Note that $| \cup_{f\sim r'} Y_f \cup Z_{r'}| \leq |(\cup_{e\sim r} X_e \cup W_r) \cup \delta^+_D(\cup_{e\sim r} X_e \cup W_r)| \leq (dtw(D)+1)(\Delta+1)$. ■

6 Complexity Results

In this section, we examine the complexity of Edge, Vertex and Unrestricted Vertex Multicut in graphs and digraphs. First we prove that Edge and Vertex Multicut are NP-hard in bounded-degree trees, while Unrestricted Vertex Multicut is NP-hard in series-parallel graphs of bounded degree. Second, we show similar results for digraphs. Then we show that the Weighted Edge Multicut is Max SNP-hard in binary tree and finally we prove that Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls (walls are graphs defined in Section 6).

Theorem 6.1 *Edge Multicut in binary trees is NP-hard.*

Proof. The reduction is from 3-SAT, a well-known NP-complete problem [GJ79].

Consider an instance Φ of 3-SAT, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, each clause with exactly three literals.

Let us construct an instance of Edge Multicut: a binary tree T and a set of pairs of distinct vertices of T. The tree T is built as follows. For each variable x_i, there is a gadget as depicted in Figure 1 (a). The gadget consists of a binary tree with three vertices: the root and two leaves, one labeled x_i and the other labeled $\overline{x}_i$. For each clause C_j, there is a gadget as depicted in Figure 1 (b). The gadget consists of a binary tree with five vertices: the root, one internal vertex and three leaves, each one labeled by one of the literals in C_j.

The tree T is built from these $n+m$ gadgets by arbitrarily connecting them using new vertices to get a binary tree. See Figure 1 (c) for an example.

Next, we give the set of pairs of vertices of T in our instance. For each variable x_i, there is a pair with the vertices labeled x_i and $\overline{x}_i$ in its gadget. For each clause C_j, there are two pairs in its gadget: one formed by the two leaves that are siblings and the other formed by the last leaf and the internal vertex. Finally, each vertex labeled $\tilde{x}_i$ in the gadget for a clause is paired to the vertex labeled $\tilde{x}_i$ in the gadget for the variable x_i, where $\tilde{x}_i \in \{x_i, \overline{x}_i\}$. This ends the construction of the instance for Edge Multicut. Note that all this can be done in polynomial time in the size of Φ.

Lemma 6.2 Φ *is satisfiable if an only if there is a solution for T of size exactly $n + 2m$.*

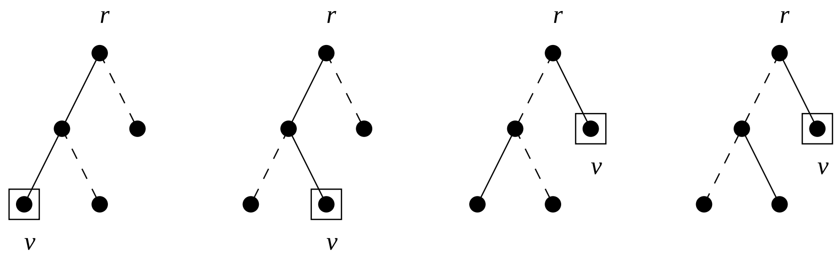

Figure 2: Possible choices of two edges, the dashed edges, in the gadget for a clause that leave exactly one leaf (the marked leaf v) connected to the root r.

Proof. Assume Φ is satisfiable. Let us present a solution S for T of size exactly $n+2m$. The edge set S consists of two types of edges:

1. For each variable x_i, S contains the edge in the gadget for x_i incident to the leaf labeled x_i if $x_i =$*TRUE* or to the leaf labeled $\overline{x}_i$ if $x_i =$*FALSE*.

2. For each clause C_j, S contains two distinct edges in the gadget for C_j. These edges are such that (1) they disconnect the two pairs in the gadget, and (2) the only leaf that is still connected to the root of the gadget is a leaf with a label $\tilde{x}_i \in C_j$ such that $\tilde{x}_i =$*TRUE*. (The four possible choices for the two edges are shown in Figure 2.)

Clearly such set S has exactly $n+2m$ edges. Let us prove that S is in fact a solution for T. It is easy to see that S disconnects the pairs inside the gadget of every clause and of every variable. The remaining pairs consist of two vertices labeled by a literal $\tilde{x}_i$, one in the variable gadget for x_i and the other in a clause gadget. If $\tilde{x}_i =$*TRUE*, then the edge in the variable gadget incident to the vertex labeled $\tilde{x}_i$ is in S, guaranteeing that the pair is disconnected. If $\tilde{x}_i =$*FALSE*, then the vertex labeled $\tilde{x}_i$ in the clause gadget is disconnected from the root of this gadget, and therefore, from the gadget for x_i. Thus S is a solution for T, and it has exactly $n+2m$ edges.

Let us prove the inverse implication. Assume there is a solution S for T with exactly $n+2m$ edges: one per variable and two per clause (one for each of the "disjoint" pairs). More specifically, S has exactly one edge in each variable gadget, and exactly two edges in each clause gadget in one of the configurations of Figure 2. Set $x_i =$*TRUE* if the edge of S in the gadget for x_i is incident to the vertex labeled x_i; set $x_i =$*FALSE* otherwise.

For each clause C_j, there is exactly one leaf v in the gadget for C_j that is connected to the root r of the gadget. Let $\tilde{x}_i \in \{x_i, \overline{x}_i\}$ be the label for this leaf. There is a pair formed by this leaf v and the leaf in the gadget for x_i whose label is $\tilde{x}_i$. In S, there must be an edge e in the path between these two leaves. Since leaf v is connected to the root r of the gadget for C_j and all edges in S are either in a variable gadget or in a clause gadget, this edge e has to be in the variable gadget. This means e is the edge incident to the leaf labeled $\tilde{x}_i$ in the gadget for x_i. Hence $\tilde{x}_i =$*TRUE*, and the clause is satisfied. Since this holds for all the clauses, the given assignment makes Φ *TRUE*, implying that Φ is satisfiable. ■

Note that any solution for T must have at least $n+2m$ edges (there are $n+2m$ "disjoint" pairs: one in each variable gadget and two in each clause gadget). Therefore Φ is satisfiable if and only if the optimal solution for T has at most $n+2m$ edges. This completes the proof of Theorem 6.1. ■

Theorem 6.3 *Vertex Multicut in tree with maximum degree at most four is NP-hard.*

Proof. Recall that in Vertex Multicut we are not allowed to remove terminals. The reduction is also from 3-SAT, and very similar to the previous one. We just have to modify the variable and clause gadgets as shown in Figure 3 (a) and (b). Edges incident to leaves are subdivided and the new vertex takes the

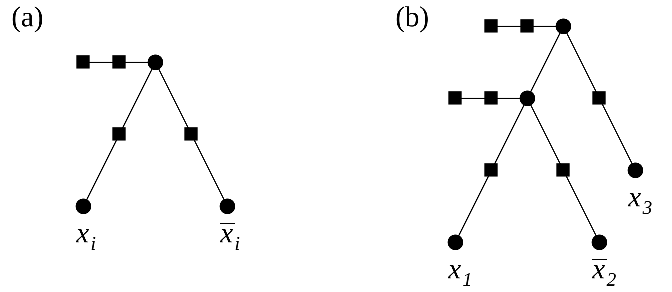

Figure 3: (a) The new gadget for variable x_i. (b) The new gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. Squares indicate new vertices.

role of the subdivided edge. For each root or internal vertex v in the gadgets we add two new vertices u and w, the edges uw and wv and the pair (u, v).

By similar arguments, the formula is satisfiable if and only if there is a solution for the constructed instance of Vertex Multicut of size at most $2n + 4m$. ■

Theorem 6.4 *Unrestricted Vertex Multicut in series-parallel graphs with maximum degree at most three is NP-hard.*

Proof. We present a reduction from 3-SAT to Unrestricted Vertex Multicut. The reduction is a modification of the previous reduction.

Consider an instance Φ of 3-SAT, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, each clause with exactly three literals.

Let us construct an instance of the Unrestricted Vertex Multicut problem: a series-parallel graph G and a set of pairs of vertices of G. The graph G is built as follows. For each variable x_i, there is a gadget which consists of an edge with endpoints one labeled x_i, the other $\overline{x}_i$ (see Figure 4 (a)). For each clause C_j, there is a gadget as depicted in Figure 4 (b), with three vertices labeled by the literals in C_j, as shown in the picture.

The graph G is obtained from the gadgets by connecting them in parallel, adding new vertices as necessary, to get a series parallel graph with maximum degree at most three. See Figure 4 (c) for an example. Vertices in a gadget are called *internal vertices.*

Next, we give the set of pairs of vertices of G in our instance. For each variable x_i, there is a pair with the vertices labeled x_i and $\overline{x}_i$ in its gadget. For each clause C_j, there are three pairs, one for each pair of literals in C_j, formed by the two vertices in the clause gadget labeled by these two literals. Finally, each vertex labeled $\tilde{x}_i$ in a clause gadget is paired to the vertex labeled $\tilde{x}_i$ in the gadget for the variable x_i, where $\tilde{x}_i \in \{x_i, \overline{x}_i\}$. This ends the construction of the instance for Unrestricted Vertex Multicut. Note that all this can be done in polynomial time in the size of Φ.

Lemma 6.5 Φ *is satisfiable if an only if there is a solution for* G *of size exactly* $n + 2m$*.*

Proof. Assume Φ is satisfiable. Let us present a solution S for T of size exactly $n + 2m$. The vertex set S consists of two types of vertices:

1. For each variable x_i, S contains the vertex in the gadget for x_i labeled x_i if $x_i =$*TRUE* or the vertex labeled $\overline{x}_i$ if $x_i =$*FALSE.*

2. For each clause C_j, S contains two distinct vertices in the gadget for C_j. These vertices are two of the three labeled vertices in the gadget, chosen so that the one left out has a label $\tilde{x}_i$ such that $\tilde{x}_i =$*TRUE.*

Clearly such set S has exactly $n+2m$ vertices. Let us prove that S is in fact a solution for T. It is easy to see that S disconnected the pairs for the variables, and the pairs for the clauses. The remaining pairs consist of two vertices labeled by a literal $\tilde{x}_i$, one in the variable gadget for x_i and the other in a clause gadget. If $\tilde{x}_i =$*TRUE*, then the vertex labeled $\tilde{x}_i$ in the variable gadget is in S, guaranteeing that the pair is disconnected. If $\tilde{x}_i =$*FALSE*, then the vertex labeled $\tilde{x}_i$ in the clause gadget is disconnected from s and t, and therefore, from the gadget for x_i. Thus S is a solution for T, and it has exactly $n+2m$ vertices.

Let us prove the inverse implication. Assume there is a solution S for T with exactly $n+2m$ vertices: one per variable and two per clause (one for each of the "disjoint" pairs). More specifically, S has exactly one vertex in each variable gadget, and exactly two vertices in each clause gadget in one of the configurations of Figure 5. Set $x_i =$*TRUE* if the vertex of S in the gadget for x_i is labeled x_i; set $x_i =$*FALSE* otherwise.

Let s and t be the extremes of the series-parallel graph G (see Figure 4 (c)). For each clause C_j, there is exactly one internal labeled vertex $v \notin S$ which is connected in $G \setminus S$ to both s and t. Let $\tilde{x}_i \in \{x_i, \overline{x}_i\}$ be the label of v, and let u be the vertex in the gadget for x_i such that the label of u is also $\tilde{x}_i$. In $G \setminus (S \setminus \{u\})$, there is a path between u and either s or t. In conclusion, in $G \setminus (S \setminus \{u\})$, there is a path between u and v. As (u, v) is a pair which must be disconnected by S, it follows that $u \in S$. Hence $\tilde{x}_i =$*TRUE*, and the clause is satisfied. Since this holds for all the clauses, the given assignment makes Φ *TRUE*, implying that Φ is satisfiable. ■

Again note that any solution for G must have at least $n + 2m$ vertices (in each variable gadget, at least one internal vertex must be removed, and in each clause gadget, at least two internal vertices must be removed, for a total of at least $n + 2m$ vertices). Therefore Φ is satisfiable if and only if the optimal solution for G has at most $n + 2m$ vertices. This completes the proof of Theorem 6.4. ■

A *bidirected tree* is a digraph which has as underlying graph a tree and two opposite directed edges for each edge in its underlying graph. Similarly, a *bidirected series-parallel digraph* has as underlying graph a series-parallel graph, and two opposite directed edges for each edge in its underlying graph. Bidirected trees have tree-width one and bidirected series-parallel digraphs have tree-width two, by Proposition 2.1 in [JRST99]. By a simple modification of the proofs shown above we can prove the following.

Theorem 6.6 *Directed Edge, Vertex and Unrestricted Vertex Multicut are NP-hard respectively in bidirected trees with maximum in and out degree three, bidirected trees with maximum in and out degree four and bidirected series-parallel digraphs with maximum in and out degree three.*

The modification consists of using, in each of the reductions, a digraph whose underlying graph is exactly the one described in the corresponding undirected case. Besides, the digraph has two opposite directed edges for each edge in its underlying graph. Note that any directed path in this digraph corresponds to a path in its underlying graph and *vice-versa*. For the edge version, we also note that any feasible solution can be converted in a feasible solution with edges directed away from the root, without increasing the size of the solution, by simply replacing any edge directed towards the root by the parallel edge directed away from the root. Thus the reductions work exactly as in the undirected case.

The next theorem, combined with the celebrated result of Arora et al. [ALMSS92], shows that a PTAS cannot be obtained for Weighted Edge Multicut in binary trees.

Theorem 6.7 *Weighted Edge Multicut is Max SNP-hard in binary trees.*

Proof. Recall that Unweighted Edge Multicut in stars is equivalent, including performance ratio, to the Max SNP-hard problem Vertex Cover. Let us reduce Edge Multicut in stars to Weighted Edge Multicut in binary trees. From an instance of the Unweighted Edge Multicut restricted to stars, we construct an instance of the Weighted Edge Multicut restricted to binary trees in the following way: for each leaf of the star S, there is a corresponding leaf in the binary tree T. The pairs are the same (we may assume there is no pair involving the root of the star). We connect the leaves of T arbitrarily into a binary tree, giving the edges in T incident to the leaves weight one and all other edges of T weight $2n+1$, where n is the number of leaves in the star S (which is the same as the number of leaves in the tree T we construct). Any solution within twice the optimum for the Weighted Edge Multicut instance we constructed will contain only edges of T incident to the leaves, since any other edge is too heavy. Then it is easy to see that any optimal solution for the Weighted Edge Multicut instance we constructed corresponds to an optimal solution for the original Unweighted Multicut star instance, and *vice versa*. Also approximability is preserved by this reduction. ■

A *wall of height* h consists of $h+1$ vertex disjoint paths $R_0, \ldots, R_h$, which we call *rows*, and $h+1$ vertex disjoint paths $L_0, \ldots, L_h$, which we call *columns*. A wall of height six is depicted in Figure 6. The formal definition is as follows. Each row is a path of $2h+2$ vertices. Each column, a path with $2h+1$ vertices. Column r contains the $(2r+1^{st})$ and the $(2r+2^{nd})$ vertices of all rows, as well as the edge between them. For $i < h$ and even, each L_r contains an edge between the $(2r+2^{nd})$ vertex of R_i and the $(2r+2^{nd})$ vertex of R_{i+1}. For $i < h$ and odd, each L_r contains an edge between the $(2r+1^{st})$ vertex of R_i and the $(2r+1^{st})$ vertex of R_{i+1}. These are all the edges of the wall.

We prove that Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls. This means, by Arora et al. [ALMSS92], that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for any of the three versions of Multicut with performance ratio at most $1+\epsilon$ implies that $P = NP$.

As in [PY91], we use the concept of *L-reduction*, which is a special kind of reduction that preserves approximability.

Let A and B be two optimization problems. We say A *L-reduces to* B if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $Opt_A(I)$ and $Opt_B(I')$ respectively, satisfy $Opt_B(I') \leq \alpha \cdot Opt_A(I)$, and

2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - Opt_A(I)| \leq \beta \cdot |c' - Opt_B(I')|$.

Theorem 6.8 *Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls.*

Proof. The reduction is from the well-known Max SNP-hard problem MAX 3-SAT [PY91]. We only show the reduction for Unrestricted Vertex Multicut. The other two reductions are similar.

The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance Φ of MAX 3-SAT, f produces an instance $W, \mathcal{C}$ of Unrestricted Vertex Multicut such that W is a wall. Also, the cost of the optimum of $W, \mathcal{C}$ in Unrestricted Vertex Multicut, denoted by $Opt_{MC}(W, \mathcal{C})$, is at most α times the cost of the optimum of Φ in MAX 3-SAT, denoted by $Opt_{SAT}(\Phi)$, i.e., $Opt_{MC}(W, \mathcal{C}) \leq \alpha \cdot Opt_{SAT}(\Phi)$.

Consider an instance Φ of MAX 3-SAT, that is, a collection of m clauses on n variables $x_1, \ldots, x_n$, each consisting of exactly three literals. Let us describe the corresponding instance for Unrestricted Vertex Multicut. The wall W is a wall of height $3m$. To describe the collection $\mathcal{C}$ of pairs of vertices of W, consider the last row of W partitioned into m same length paths, each one associated to one of the clauses of Φ. Each path has length 6. Label the $2^{nd}, 4^{th}$ and 6^{th} vertices in the j^{th} path each with one of the literals in the j^{th} clause. See Figure 6 for an example. For each pair of vertices u, v in W, u labeled x_i and v labeled $\overline{x}_i$, include into $\mathcal{C}$ the pair u, v. For each clause, include three pairs: the three pairs formed by each two of the vertices labeled by its three literals. This ends the description of the instance of Unrestricted Vertex Multicut.

First note that W and $\mathcal{C}$ can be obtained in polynomial time in the size of Φ.

Lemma 6.9 $Opt_{MC}(W, \mathcal{C}) \leq 6 \cdot Opt_{SAT}(\Phi)$.

Proof. The set S of all labeled vertices of W is clearly a solution of Unrestricted Vertex Multicut for W and $\mathcal{C}$. Thus $Opt_{MC}(W, \mathcal{C}) \leq |S| = 3m$. On the other hand, $Opt_{SAT}(\Phi) \geq m/2$ because either a truth assignment for Φ satisfies at least half of the clauses of Φ or its negation satisfies at least half of the clauses of Φ. But then $Opt_{MC}(W, \mathcal{C}) \leq 3m = 6 \cdot m/2 \leq 6 \cdot Opt_{SAT}(\Phi)$. ■

This finishes the first part of the L-reduction, since we can take $\alpha = 6$.

The second part of the L-reduction is the constant β and the algorithm g. Given a vertex set S of W with s edges, g produces in polynomial time a truth assignment for Φ which satisfies t clauses, where t is such that $|t - Opt_{SAT}(\Phi)| \leq \beta|m - Opt_{MC}(W, \mathcal{C})|$. We shall see that $\beta = 1$ suffices.

First note that $opt_{MC}(W, \mathcal{C}) \leq 3m - opt_{SAT}(\Phi)$. To verify this, it is enough to present a solution S' for W with $|S'| = 3m - opt_{SAT}(\Phi)$. Consider an assignment for Φ which satisfies $opt_{SAT}(\Phi)$ clauses of Φ. For each satisfied clause, include into S' two of its literals, leaving out a literal which is *TRUE*. For each non-satisfied clause, include into S' its three literals. Clearly S' has $3m - opt_{SAT}(\Phi)$ (one less vertex per satisfied clause). Now, note that

$$\begin{aligned}
|s - opt_{MC}(W, \mathcal{C})| &= s - opt_{MC}(W, \mathcal{C}) \\
&\geq s - (3m - opt_{SAT}(\Phi)) \\
&= (s - 3m) + opt_{SAT}(\Phi) \\
&= opt_{SAT}(\Phi) - (3m - s)
\end{aligned}$$

Let us present an assignment which satisfies at least $3m - s$ clauses. If $s \geq 3m$ then any assignment does it. If $s < 3m$ then there is a row R_T of W with no vertex in S (because W contains $3m+1$ vertex disjoint rows). Set $\tilde{x}_i$ to *TRUE* if and only if there is a path in $W \setminus S$ from a vertex labeled $\tilde{x}_i$ to a vertex in row R_T. Such assignment is well-defined since a vertex labeled x_i and a vertex labeled $\overline{x}_i$ cannot be both connected to R_T in $W \setminus S$, or S would not be a solution (because these two vertices form a pair).

Let us prove that this assignment satisfies at least $3m - s$ clauses. For this, we need some notation.

Associate to each clause C_j a subgraph W_j of W induced by all vertices appearing in columns $L_{3(j-1)}, \ldots, L_{3j-1}$. Note that the graphs W_j are vertex disjoint.

In each W_j, we have the three vertices labeled by the literals in C_j. Recall that each two of these three vertices form a pair. One can verify that there must be at least two vertices of S in W_j, just because of these three pairs. Thus there are at least $3m - s$ subgraphs W_j which have exactly two vertices of S. But note that, if W_j has only two vertices of S, then at least one of the labeled vertices in W_j is connected to R_T. This means there is at least one literal in C_j which is assigned *TRUE*. Therefore this assignment satisfies at least $3m - s$ clauses and therefore

$$|s - opt_{MC}(W, \mathcal{C})| \geq |t - opt_{SAT}(\Phi))|,$$

where t is the number of clauses in Φ satisfied by the described assignment. This completes the proof of the second part of the L-reduction. ■

7 Acknowledgments

We would like to thank Howard Karloff for suggesting the problem, and for many helpful discussions. The first two authors also would like to acknowledge the help of Robin Thomas in understanding the directed tree-width concept.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 14–23, 1992.

[AL91] S. Arnborg and J. Lagergren, "Problems Easy for Tree-Decomposable Graphs," *Journal of Algorithms*, 12 (2), 308-340, 1991.

[DJPSY94] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour and M. Yannakakis, "The Complexity of Multiterminal Cuts," *SIAM Journal on Computing*, 23 (4), 864-894, 1994.

[D70] E. A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation," *Soviet Mathematics Doklady*, 11, 1277-1280, 1970.

[ENSZ96] G. Even, J. S. Naor, B. Schieber and L. Zosin, "Approximating Minimum Subset Feedback Sets in Undirected Graphs with Applications," *Proc.* 4^{th} *Israel Symposium on Theory of Computing and Systems*, 78-88, 1996.

[FF56] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, 8, 399-404, 1956.

[FF58] L. R. Ford and D. R. Fulkerson, "A Suggested Computation for Maximal Multicommodity Network Flows," *Management Science*, 5, 97-101, 1958.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[GVY96] N. Garg, V. Vazirani and M. Yannakakis. "Approximate Max-Flow Min-(Multi)Cut Theorems and Their Applications," *SIAM Journal on Computing*, 25 (2), 235-251, 1996.

[GVY97] N. Garg, V. Vazirani and M. Yannakakis. "Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees," *Algorithmica*, 18 (1), 3-20, 1997.

[HT84] D. Harel and R. E. Tarjan, "Fast Algorithms for Finding Nearest Common Ancestor," *SIAM Journal on Computing*, 13 (2), 338-355, 1984.

[H63] T. C. Hu, "Multicommodity Network Flows," *Operations Research*, 9, 898-900, 1963.

[I78] A. Itai, "Two-Commodity Flow," *Journal of ACM*, 25, 596-611, 1978.

[JRST99] T. Johnson, N. Robertson, P.D. Seymour and R. Thomas, "Directed Tree-Width," *preprint* (available at `http://www.math.gatech.edu/~thomas/`).

[KARR90] P. Klein, A. Agrawal, R. Ravi and S. Rao, "Approximation through Multicommodity Flow," *Proc.* 31^{st} *IEEE Symposium on Foundations of Computer Science*, 726-737, 1990.

[KPRT97] P. Klein, S. Plotkin, S. Rao and E. Tardos, "Approximation Algorithms for Steiner and Directed Multicuts," *Journal of Algorithms,* 22 (2), 241-269, 1997.

[LR88] F. T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Application to Approximation Algorithms," *Proc.* 29^{th} *IEEE Symposium on Foundations of Computer Science*, 422-431, 1988.

[S95] D. B. Shmoys, "Computing Near-Optimal Solutions to Combinatorial Optimization Problems," in *Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (20), W. Cook and L. Lovász (eds.), AMS Publications, 1995, 355-397.

[PT93] S. Plotkin and E. Tardos, "Improved Bounds on the Max-Flow Min-Cut Ratio for Multicommodity Flows," *Proc.* 25^{th} *Annual ACM Symp. on Theory of Computing*, 691-697, 1993.

[PY91] C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425–440, 1991.

[RS86] N. Robertson and P. Seymour, "Graph Minor II. Algorithmic Aspects of Tree-Width," *Journal of Algorithms*, 7, 309–322, 1986.

[RS84] N. Robertson and P. Seymour, "Graph Minor III. Planar Tree-Width," *Journal of Combinatorial Theory (B)*, 36, 49–63, 1984.

[TV93] E. Tardos and V. V. Vazirani, "Improved Bounds for the Max-Flow Min-Multicut Ratio for Planar and $K_{r,r}$-free graphs," *Information Processing Letters*, 47 (2), 77-80, 1993.

[vL90] J. van Leeuwen, "Graph Algorithms," *Handbook of Theoretical Computer Science*, Volume A, chapter 10, 525-631. The MIT Press/Elsevier, 1990.

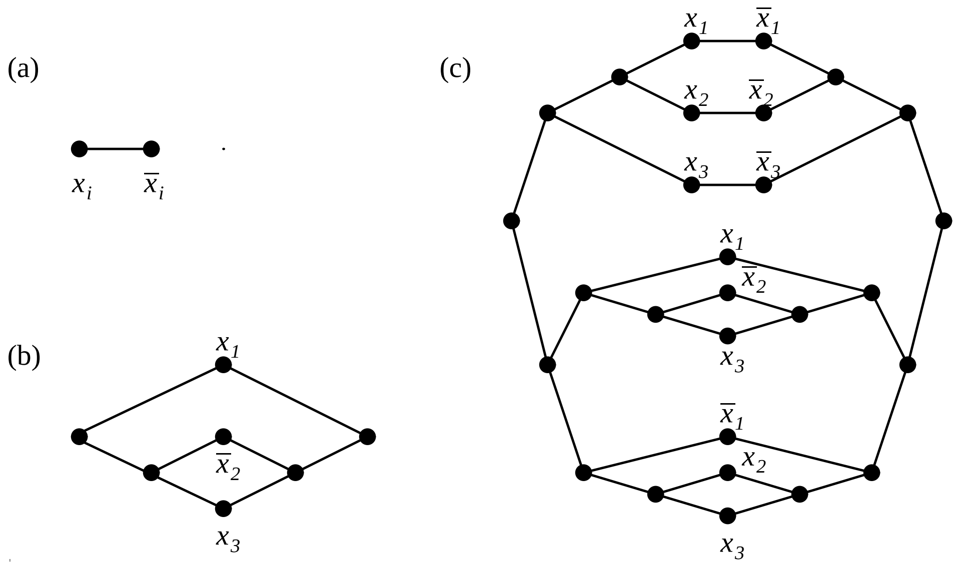

Figure 4: (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Graph G built for the instance $\Phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \overline{x}_2, x_3\}$ and $C_2 = \{\overline{x}_1, x_2, x_3\}$.

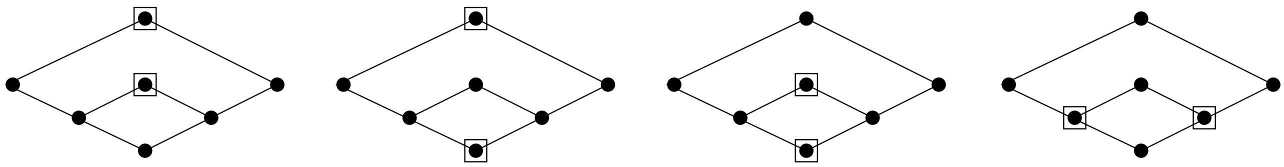

Figure 5: Possible choices of two vertices, the vertices marked by a square, in the gadget for a clause that leave exactly one internal labeled vertex connected to the extremes s and t.

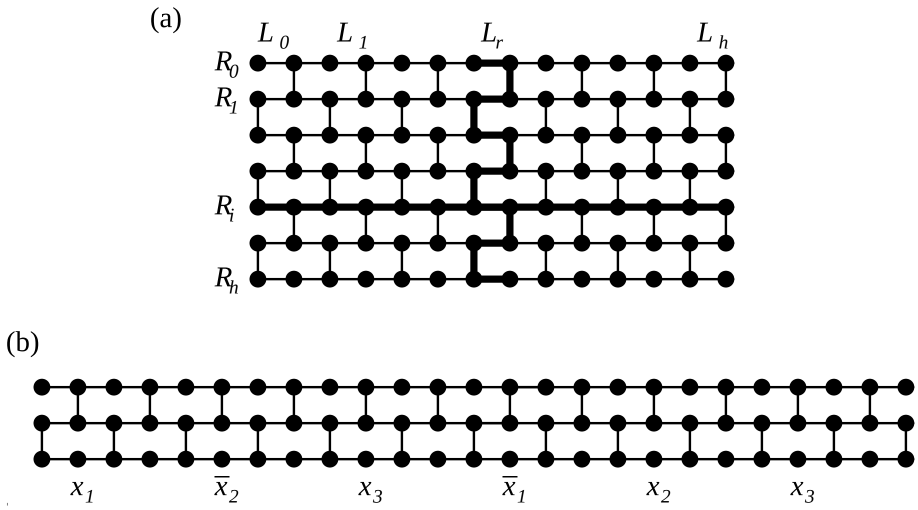

Figure 6: A wall of height six. The dark edges indicate row R_i and column L_r. The labels are given according to the formula $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_2 \vee x_3)$.