Multicuts in Unweighted Graphs 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. The fractional Multicut problem is the dual of the well-known Multicommodity Flow problem. 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 relax any of the three condition (unweighted, bounded-degree, bounded tree-width). We also show that some of these results extend to the vertex 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 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]. Important 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.

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. Research supported in part by NSF grant CCR-9319106. E-mail: gruia@cc.gatech.edu.

†Department of Computer Science, University of São Paulo, Brazil. E-mail: cris@ime.usp.br. Research supported partially by NSF grant CCR-9319106 and by FAPESP (Proc. 96/04505–2).

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

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 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 [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]). 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.

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 $\Theta(n+k)$.

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.

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 4, 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. Finally, in Section 4 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 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 nonactive pairs in $\mathcal{C}$ are disconnected by S. A pair in $\mathcal{C}$ that becomes nonactive 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.

2.1 Bounded-Tree-Width Graphs

Next we present a PTAS for Unrestricted Vertex Multicut in graphs with bounded tree-width. 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 $X_u^{i-1} \subseteq A^i$ */
  $G^i = G^{i-1} \setminus G^{i-1}(u_i)$;
  $T^i = T^{i-1} \setminus T^{i-1}(u_i)$;
  $X_w^i = X_w^{i-1} \setminus V(G^{i-1}(u_i))$, 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 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 be a vertex of T, x be a vertex of $G(u)$ and y be a vertex of $G \setminus G(u)$. Then any path in G from x to y contains a vertex of X_u.*

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 $X^{i-1}_{u_i}$ in the segment of P from s to y. Since $X^{i-1}_{u_i} \subseteq S^f$, 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 $X^{j-1}_{u_j} \subseteq S^f$ 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_0 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 $|A_0| < t$ then proceed to the next vertex in postorder. If $|A_0| \geq t$, then we output u and the set $A = A_0 \cup X_u$. Note that in fact $opt(G^{i-1}(u)) = |A_0| \geq t$, $X_u \subseteq A$ 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 theorem 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.*

We omit the proof in this extended abstract.

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.*

In fact we know how to implement the PTAS given in Section 2.1, for Edge Multicut in bounded-degree trees, in time $O((n + k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$, 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 $\Theta(n + k)$. We omit the description of this linear-time implementation in this extended abstract.

4 Complexity Results

In this section, we examine the complexity of Edge, Vertex and Unrestricted Vertex Multicut. 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 that the Weighted Edge Multicut ins NP-hard in binary tree. Finally we prove that Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls (defined in Section 4).

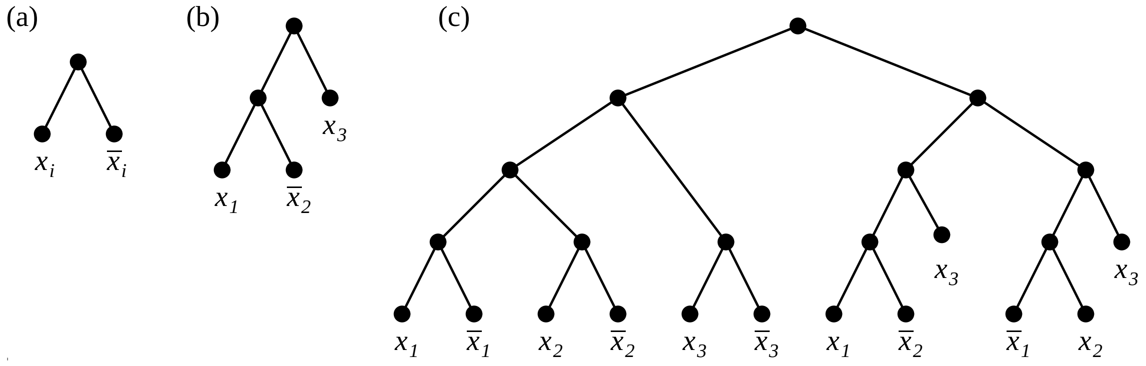

Figure 1: (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Tree T built for the instance $\Phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \overline{x}_2, x_3\}$ and $C_2 = \{\overline{x}_1,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

Theorem 4.1 *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 2 (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 2 (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: 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.

The next lemma completes the proof of Theorem 4.1.

Lemma 4.2 *Φ is satisfiable if an only if there is a solution for T of size exactly $n + 2m$. Moreover, we can construct in polynomial time such a solution for T from a truth assignment for Φ and vice versa.*

We omit the proof in this extended abstract.

■

Theorem 4.3 *Vertex Multicut in tree with maximum degree at most four is NP-hard.*

We omit the proof. The construction is similar to the one used in Theorem 4.1.

Theorem 4.4 *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 2 (a)). For each clause C_j, there is a gadget as depicted in Figure 2 (b), with three vertices labeled by the literals in C_j, as shown in the picture.

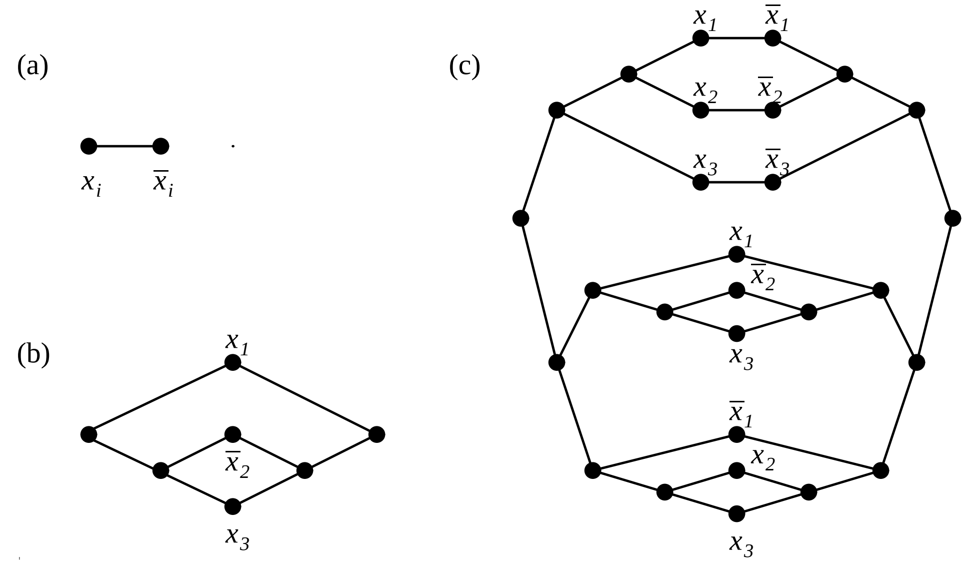

Figure 2: (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Graph G built for the instance $\Phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \overline{x}_2, x_3\}$ and $C_2 = \{\overline{x}_1, x_2, x_3\}$.

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 2 (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.

The next lemma completes the proof of Theorem 4.4. ■

Lemma 4.5 *Φ is satisfiable if an only if there is a solution for G of size exactly $n + 2m$. Moreover, we can construct in polynomial time such a solution for G from a truth assignment for Φ and vice versa.*

We omit the proof in this extended abstract.

Theorem 4.6 *Weighted Edge Multicut is Max SNP-hard in binary tree.*

Proof sketch. 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 3 (a). The reader should be able to complete the definition by considering Figure 3 (a). 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

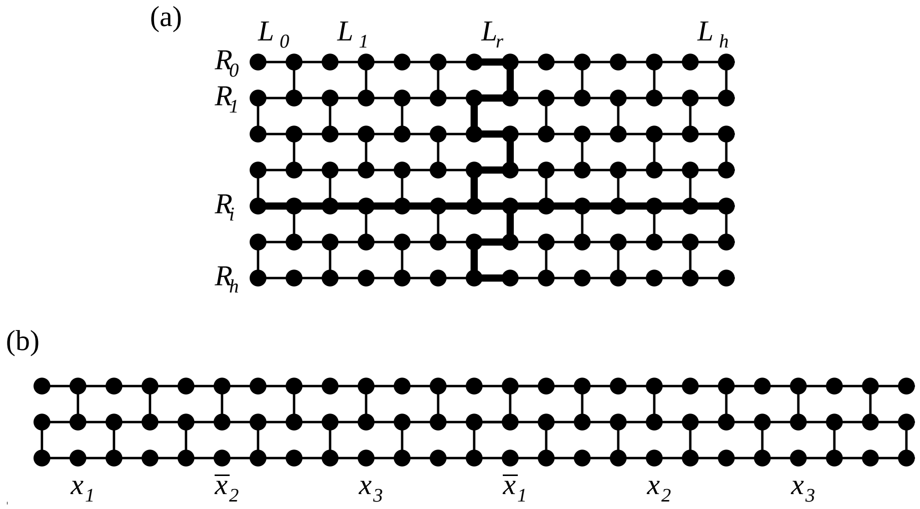

Figure 3: (a) A wall of height six. The dark edges indicate row R_i and column L_r. (b) The three last rows of the wall built from $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_2 \vee x_3)$.

rows, as well as the edge between them. For $i < h$ and odd, 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 even, 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.

Theorem 4.7 *Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls.*

Proof sketch. The reduction is from the well-known Max SNP-hard problem MAX 3-SAT [PY91]. We 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 $Opt_{MC}(W, \mathcal{C})$, 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 $6m$. 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 12. Label the $2^{nd}, 6^{th}$ and 10^{th} vertices in the j^{th} path each with one of the literals in the j^{th} clause. See Figure 3 (b) 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 4.8 $Opt_{MC}(W, \mathcal{C}) \leq 6 \cdot Opt_{SAT}(\Phi)$.

Proof sketch. $W, \mathcal{C}$ clearly has a solution of size $3m$. Also $Opt_{SAT}(\Phi) \geq m/2$. ■

Lemma 4.9 *From a solution to Φ of size s, $0 \leq s \leq m$ we can obtain a solution to $W, \mathcal{C}$ of size $3m - s$ and viceversa.*

We omit the proof of this lemma.

The previous two lemmas can be used in an obvious way to show the reduction we presented is an L-reduction. ■

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33^{rd} 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.

[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.

[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.