Algorithms for Connected Multicuts of Weighted Trees*

Sunny Atalig[†], Marek Chrobak[†], Petr Kolman[§], Claire Mathieu[*], and Hans Raj Tiwary[§]

[†]University of California at Riverside, USA
[§]Charles University, Prague, Czech Republic
[*]CNRS and Université Paris Cité, Paris, France

October 17, 2026

Abstract

In the *connected multicut problem for weighted trees*, the instance is an edge-weighted tree $T = (V, E, \omega)$, a set of terminals $\Gamma \subseteq V$, and a set of pairwise "conflicts" between these terminals represented by edges of a conflict graph $\mathcal{C} = (\Gamma, \mathcal{F})$. A *connected multicut* for this instance is a partition $\mathcal{M}$ of the vertex set V into subsets called the *components of* $\mathcal{M}$, where each component induces a subtree of T, contains at least one terminal, but does not contain any conflicting terminal pair. Two cost functions for multicuts are considered: in the min-sum variant, we minimize the total weight of the edges between different components of $\mathcal{M}$, and in the min-max variant, we minimize the maximum, over all components, of the total weight of edges leaving the component.

In the first part of the paper, we focus on algorithms for the min-max variant, for which we give a 2-approximation algorithm for the unweighted case, a $(2 + \epsilon)$-approximation algorithm for the general case, and a mildly exponential exact algorithm for the unweighted case.

In the second part of the paper, we study algorithms that take advantage of the structure of the conflict graph. We first give a polynomial-time exact algorithm for the min-max variant when $\mathcal{C}$ is a clique (the variant referred to in the literature as the *multiway cut*). Next, we generalize this result by providing polynomial-time exact algorithms, for both variants, for instances whose conflict graph has constant *split complexity* — a new parameter that refines the concept of neighborhood diversity previously studied in the context of parameterized complexity. This unifies and generalizes previous results for the min-sum variant when conflict graphs are complete multipartite graphs or unions of cliques.

[*]Research partially supported by NSF grant CCF-2153723 (Marek Chrobak), grant 24-10306S of GA ČR (Petr Kolman), and by AGATE project funded from the Horizon Europe Programme under Grant Agreement No. 101183743 (Hans Raj Tiwari).

1 Introduction

In the *connected multicut problem for weighted trees*, the instance is an edge-weighted tree $T = (V, E, \omega)$ with n vertices, a set of *terminals* $\Gamma \subseteq V$, and a set of pairwise "conflicts" between these terminals represented by edges of a conflict graph $\mathcal{C} = (\Gamma, \mathcal{F})$. A *connected multicut* for this instance is a partition $\mathcal{M}$ of the vertex set into subsets called the *components of* $\mathcal{M}$, where each component induces a subtree of T, contains at least one terminal, but does not contain any conflicting terminal pair. When $\mathcal{C}$ is a clique, that is when all terminals conflict with each other, multicuts are referred to as *multiway cuts* in the literature.

Min-sum cost. One natural way to measure the cost of a multicut $\mathcal{M}$ is by the total weight of the edges between different components of $\mathcal{M}$, often referred to as the *min-sum* cost. The problem of minimizing the min-sum cost of a connected multicut in trees, denoted here TreeMCut$_\Sigma$, has been well studied in the literature. It is easy to see that the connectivity requirement is redundant for TreeMCut$_\Sigma$ — any multicut without conflicts in its components can be converted, without increasing cost, into a connected one. Garg et al. [18] proved that TreeMCut$_\Sigma$ is $\mathbb{NP}$-hard. They also provided a primal-dual algorithm with approximation ratio of 2, that (under standard assumptions) is optimal, because their $\mathbb{NP}$-hardness proof shows that for star graphs, even without weights, the problem is polynomial-time equivalent to the VertexCover problem. As shown by Calinescu et al. [14], the unweighted version of TreeMCut$_\Sigma$ remains $\mathbb{NP}$-hard even for binary trees, although for such trees an approximation ratio $1 + \epsilon$ can be achieved, for each $\epsilon > 0$. Yannakakis et al. [38] observed (prior to [18]) that TreeMCut$_\Sigma$ is polynomial-time solvable for a constant number of terminals. The complexity of TreeMCut$_\Sigma$ parametrized by the number of edges in the optimum multicut has been extensively studied [20, 31, 4, 6, 25].

Some earlier work on TreeMCut$_\Sigma$ studied its subproblems with some restrictions on the structure of the conflict graph. The case when $\mathcal{C}$ is a clique is the multiway cut problem for trees, for which, as observed earlier [8, 16, 22, 10], the dynamic-programming approach yields an $O(n)$-time algorithm. Motivated by applications to query optimization in database applications, Hasan and Motwani [22] studied a more general model when $\mathcal{C}$ is a complete multi-partite graph, for which they gave an $O(n^2)$-time algorithm. Another natural extension of the clique case is when $\mathcal{C}$ is a union of a constant number of cliques, for which polynomial-time algorithms were given by Liu et al. [31] and Kanj et al. [25].

Min-max cost. The *min-max* cost of a multicut is the maximum weight of the boundaries of $\mathcal{M}$'s components, where a boundary of a component is the set of edges leaving this component. By ConTreeMCut$_{\max}$ we denote the problem of optimizing the min-max objective for connected multicuts in trees. It was first considered by Svitkina and Tardos [37], who observed that the connectivity requirement in ConTreeMCut$_{\max}$ is essential (see also Appendix D). A systematic study of ConTreeMCut$_{\max}$ was initiated by Kolman and Tiwary [27], who focussed on its multiway-cut variant ConTreeMwCut$_{\max}$ (where $\mathcal{C}$ is a clique) and proved that it is fixed-parameter tractable when parametrized by the number of terminals, and showed that its natural polytope has super-polynomial extension complexity.

Our results. In the first part of this paper, in Section 3, we focus on algorithms for ConTreeMCut$_{\max}$ that apply to arbitrary conflict graphs. Unlike TreeMCut$_\Sigma$, that has substantial literature, the only known complexity results for ConTreeMCut$_{\max}$ are those in [27], that only address its multiway-cut restriction ConTreeMwCut$_{\max}$ (when $\mathcal{C}$ is a clique). By the same argument as for TreeMCut$_\Sigma$, ConTreeMCut$_{\max}$ cannot be approximated with ratio smaller than 2. The idea behind the LP-formulation for TreeMCut$_\Sigma$ in [18] does not apply to ConTreeMCut$_{\max}$, and, in fact, ConTreeMCut$_{\max}$ does not seem to have a simple relaxation useful for bounding

the optimal min-max cost. Nevertheless, we formulate an auxiliary optimization problem called the *local cut* that does have a simple linear program relaxation and show that ConTreeMCut$_{\max}$ can be reduced to the problem of computing local cuts, yielding an algorithm with an optimal approximation ratio.

Theorem 1. *(a) For each constant $\epsilon > 0$, there is a polynomial-time $(2+\epsilon)$-approximation algorithm for* ConTreeMCut$_{\max}$. *(b) There is a polynomial-time 2-approximation algorithm for the unweighted variant of* ConTreeMCut$_{\max}$.

Somewhat surprisingly, the approximation algorithm above can be modified to yield a polynomial-time Turing reduction from unweighted ConTreeMCut$_{\max}$ to VertexCover, which is simultaneously approximation-preserving and parameter-preserving with respect to the objective functions. In particular, by applying known vertex cover algorithms [21, 9, 5], we obtain mildly-exponential algorithms for the unweighted case.

Theorem 2. *Consider the variant of* ConTreeMCut$_{\max}$*, where the input instances consist of an unweighted tree T, a conflict graph $\mathcal{C}$, and a cost threshold k.*

(a) There are algorithms with running times $O^*(1.2529^k)$ and $O^*(1.1996^{|\Gamma|})$ that determine whether a multicut of cost at most k exists[1]*.*

(b) There is an algorithm with running time $O^*(1.09^k)$ that either computes a multicut with cost at most $1.5k$ or determines that every multicut has cost greater than k.

In contrast, the fastest known algorithm for constructing an optimal solution to unweighted TreeMCut$_{\Sigma}$ runs in $O^*(1.554^k)$ time [25, 6], where k is the min-sum objective.

The second part of the paper, Sections 4 and 5, studies exact algorithms for both cost versions, with running times dependent on the structure of the conflict graph. We start, in Section 4, by presenting a polynomial-time algorithm for ConTreeMwCut$_{\max}$, the restriction of ConTreeMCut$_{\max}$ to conflict graphs that are cliques.

Theorem 3. ConTreeMwCut$_{\max}$ can be solved in $O(n \log W)$ time for integer weights (where W is the total edge weight) and $O(n^2)$ time for real weights.

This significantly improves the previous results [27], where the solution involved a parametrization by the number of terminals. Our algorithm is substantially more involved than the polynomial-time algorithm for the min-sum analog [8, 16, 22, 10]. In essence, we show that in polynomial time we can verify, using dynamic programming, if there exists a solution with cost not exceeding a given value L, so for integer weights the optimum can be found in polynomial time. To extend it to real weights, the dynamic programming algorithm processes *whole cost functions*, which we show to be piece-wise constant with only a linear number of discontinuities. In addition to its own interest, this algorithm also serves as a stepping stone towards much more general results that we discuss later in the paper.

These more general results are obtained via parametrizations. The hardness result of Garg et al. [18] applies even when the input tree T is a star, or when T has depth 2 and $\mathcal{C}$ is a matching, so fixing some commonly used structural parameters of the input graph (say, maximum vertex degree, modular-width, etc.) cannot produce efficient algorithms for TreeMCut$_{\Sigma}$. Parametrizations by the number of edges in the optimum solution produce running times that are overly pessimistic for some simple instances, for example when the confict graph is a clique. This raises a natural question:

[1] $O^*()$ hides polynomial factors.

is there a parametrization *based only on the conflict graph structure* that includes as special cases the known variants of TreeMCut$_\Sigma$ with polynomial-time algorithms, namely the instances when the conflict graph is a complete multi-partite graph [22] or a union of a constant number of (possibly overlapping) cliques [31, 25]. The same question applies to ConTreeMCut$_{\max}$.

One simple parameter that naturally captures the relevant structure of confict graphs is called *neighborhood-diversity* (see [29]), denoted here by $\mathrm{nd}(\mathcal{C})$, which groups vertices into "types" according to their neighborhoods. In Section 5, we give polynomial-time algorithms for conflict graphs with constant $\mathrm{nd}(\mathcal{C})$.

Theorem 4. *(a)* TreeMCut$_\Sigma$ *can be solved in time* $O(n + |\mathcal{F}| + 4^{\mathrm{nd}(\mathcal{C})}|\Gamma|)$.

(b) ConTreeMCut$_{\max}$ *can be solved in time* $O(n + |\mathcal{F}| + 4^{\mathrm{nd}(\mathcal{C})}|\Gamma|^2)$ *for real weights and in time* $O(n + |\mathcal{F}| + 4^{\mathrm{nd}(\mathcal{C})}|\Gamma| \log W)$ *for integer weights, where* W *is the total edge weight.*

For complete c-partite graphs and for q-clique covers, Theorem 4 already matches the running times in [22, 31, 25] if the parameters c and q are constant.

To further improve Theorem 4, we introduce a generalized notion of types and a new "split-complexity" parameter $\mathrm{sp}(\mathfrak{C})$, defined on a "generalized type graph" $\mathfrak{C}$ associated with $\mathcal{C}$. Roughly, $\mathrm{sp}(\mathfrak{C})$ measures the number of ways one can divide an independent set in $\mathcal{C}$ into two parts knowing only which types it contains. We then show (Section 6) that for both cost versions the problem is tractable for polynomially-bounded $\mathrm{sp}(\mathfrak{C})$.

Theorem 5. *Given a generalized type graph* $\mathfrak{C}$ *for the conflict graph* $\mathcal{C}$*:*

(a) TreeMCut$_\Sigma$ *can be solved in time* $O(n + \mathrm{sp}(\mathfrak{C})|\Gamma|)$.

(b) ConTreeMCut$_{\max}$ *can be solved in time* $O(n + \mathrm{sp}(\mathfrak{C})|\Gamma|^2)$ *for real weights and in time* $O(n + \mathrm{sp}(\mathfrak{C})|\Gamma| \log W)$ *for integer weights, where* W *is the total edge weight.*

Applying appropriate bounds to $\mathrm{sp}(\mathfrak{C})$, we get algorithms for instances when $\mathcal{C}$ has bounded neighborhood diversity (covered by Theorem 4), is a union of cliques, and is a complete multipartite graph.

Corollary 1.1. *(a) If* $\mathcal{C}$ *is a complete* c*-partite graph, then* TreeMCut$_\Sigma$ *can be solved in time* $O(n + c|\Gamma|)$*. Likewise* ConTreeMCut$_{\max}$ *can be solved in time* $O(n + c|\Gamma|^2)$ *time for real weights and* $O(n + c|\Gamma| \log W)$ *for integer weights.*

(b) If $\mathcal{C}$ *is a union of* q *cliques, then* TreeMCut$_\Sigma$ *can be solved in time* $O(n + 3^q|\Gamma|)$*. Likewise* ConTreeMCut$_{\max}$ *can be solved in time* $O(n + 3^q|\Gamma|^2)$ *time for real weights and* $O(n + 3^q|\Gamma| \log W)$ *for integer weights.*

Previously, Corollary 1.1 was only known for TreeMCut$_\Sigma$ [22, 31]. Note that in Corollary 1.1(b) we assume that the clique cover is given on input. This assumption is necessary, as the best algorithm finding a q-clique cover takes $O^*(2^{2^q})$ time [19] and is optimal assuming ETH [13]. For this reason, we also have to assume that the object $\mathfrak{C}$ is given for Theorem 5 in general.[2] But for Theorem 4, no additional information is needed.

Due to space constraints, some proofs are included in the appendix. We also include in the appendix some side results that refine the computational complexity picture of problems related to connected multicuts. Svitkina and Tardos [37] considered a bi-objective version of connected

[2]Technically, Corollary 1.1(a) also assumes the parts are already given, but computing that can be done in $O(|\Gamma| + |\mathcal{F}|)$ time.

multicuts in trees that involves both min-sum and min-max objective functions. In Appendix C we show that that problem is (weakly) $\mathbb{NP}$-hard even if the conflict graph is a clique. In Appendix D we show that the *connectivity* requirement is essential for the min-max variants of $\textsc{MwCut}_{\max}$, proving the ratio of $\Omega(\log n)$ in the case of trees and $\Omega(\sqrt{n})$ for arbitrary graphs, between the optimal solutions without and with this requirement. This strengthens the observation from [37] showing that this ratio is at least 1.25.

Other related work. There is a variety of other variants of $\textsc{TreeMCut}_{\Sigma}$ in the literature. For example, in some models not all conflicting pairs need to be separated. The *partial multicut problem* asks for a set of edges which separate at least a given number of conflicting pairs, while the *prize-collecting multicut problem* adds a penalty for non-separated conflicts. Both problems admit constant-ratio approximations [30, 35]. Further studied extensions involve submodular cost functions on edge sets or submodular penalties for non-separated conflicts [32, 24, 23, 33, 34]. Multicuts for directed trees were studied as well [11].

Some of the results for trees have been extended to bounded-treewidth graphs, see for example, the paper by Bentz and Le Bodic [2].

The multicut problem for arbitrary graphs with the min-sum objective, $\textsc{MCut}_{\Sigma}$, has rich literature. It's known to be computationally hard. Even its multiway-cut version $\textsc{MwCut}_{\Sigma}$ with only three terminals is $\mathbb{NP}$-hard [16]. Research on $\textsc{MCut}_{\Sigma}$ includes studies of its LP-relaxations [7, 3, 8], estimates of the optimum in terms of maximal flows [17], and approximation algorithms [15, 26]. For more information, see the survey by Costa et al. [12].

The min-max variant is less understood. Its multiway-cut subproblem, $\textsc{MwCut}_{\max}$, was studied by Svitkina and Tardos [37], who showed that it is $\mathbb{NP}$-hard even for only four terminals, and developed polylogarithmic approximation algorithms, later improved by Bansal et al. [1]. Svitkina and Tardos also considered the subproblem $\textsc{TreeMwCut}_{\max}$, the restriction of $\textsc{MwCut}_{\max}$ to trees (without the connectivity requirement), for which they proved $\mathbb{NP}$-hardness and gave a $(2+\epsilon)$-approximation algorithm for any $\epsilon > 0$.

Kolman and Tiwary [27] studied the variant of $\textsc{MwCut}_{\max}$ with the connectivity requirement on multicut's components. Their results include several hardness results, for example that $\textsc{ConMwCut}_{\max}$ is (weakly) $\mathbb{NP}$-hard even for three terminals and graphs of treewidth three, a pseudo-polynomial algorithm, and an FPTAS for graphs of bounded treewidth.

2 Preliminaries

We are given a rooted tree $T = (V, E, \omega)$ with a weight function $\omega : E \to \mathbb{R}^+$ and a graph $\mathcal{C} = (\Gamma, \mathcal{F})$, where $\Gamma \subseteq V$. The vertices in Γ are called *terminals*, the edges of $\mathcal{F}$ between them are referred to as *conflicts*, and $\mathcal{C}$ is called the *conflict graph*. Throughout the paper, the number of vertices in T is denoted by $n = |V|$, and by $\omega_{\max} = \max_{e \in E} \omega(e)$ and $W = \sum_{e \in E} \omega(e)$ we denote the maximum and total edge weight, respectively. We use the notation π_v for the parent of $v \in V$ (if v is not the root) and $ch(v)$ for the set of children of v. A *connected multicut* of the instance $(T, \mathcal{C})$ is defined as a partition $\mathcal{M}$ of V into subsets called *components*, where each component X satisfies three conditions: (i) $X \cap \Gamma \neq \emptyset$, (ii) $X \cap \Gamma$ is an independent set in $\mathcal{C}$, and (iii) the subgraph of T induced by X is connected (that is, a subtree of T). For $v \in V$, $\mathcal{M}(v)$ is the component of $\mathcal{M}$ containing v. For $X \subseteq V$ we define the *boundary of* X as $\partial X = E(X, V - X)$; that is ∂X is the set of edges with exactly one endpoint in X. (In particular, $\partial \mathcal{M}(v)$ refers to the boundary of $\mathcal{M}(v) \in \mathcal{M}$.) The boundary of a multicut $\mathcal{M}$ is the union of boundaries of its components: $\partial \mathcal{M} = \bigcup_{X \in \mathcal{M}} \partial X$. By the connectivity requirement, $\mathcal{M}$ is uniquely determined by its boundary; in particular $\mathcal{M}$ consists of the connected components of $T - \partial \mathcal{M}$. We consider two cost functions for multicuts: the *sum-cost*

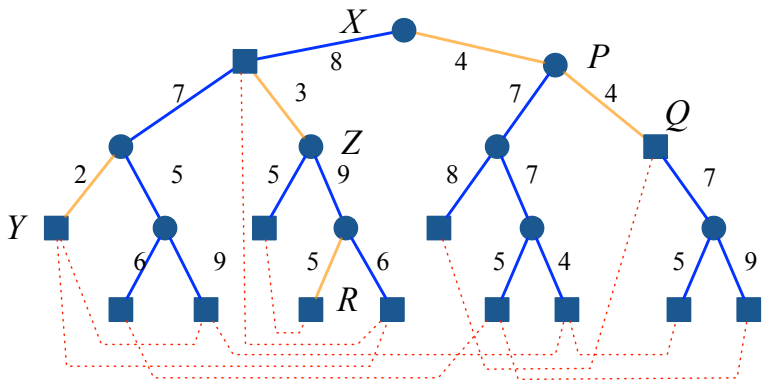


Figure 1: A weighted tree, a conflict graph, and a connected multicut $\mathcal{M} = \{X, Y, Z, P, Q, R\}$. Tree edges are represented by solid lines, non-terminals by circles, terminals by squares, and conflicts by dotted thin lines. The edges within components are in blue (dark), the boundary edges are in yellow. Each component's label is near its top vertex. The boundary costs are $\omega(\partial X) = 9$, $\omega(\partial Y) = 2$, $\omega(\partial Z) = 8$, $\omega(\partial P) = 8$, $\omega(\partial Q) = 4$, and $\omega(\partial R) = 5$. So $\mathrm{cost}_\Sigma(\mathcal{M}) = 18$ and $\mathrm{cost}_{\max}(\mathcal{M}) = 9$.

of $\mathcal{M}$ is the total weight of the edges in its boundary, $\mathrm{cost}_\Sigma(\mathcal{M}) = \omega(\partial\mathcal{M}) = \frac{1}{2}\sum_{X\in\mathcal{M}} \omega(\partial X)$, and its *max-cost* is the maximum cost of its component boundaries, $\mathrm{cost}_{\max}(\mathcal{M}) = \max_{X\in\mathcal{M}} \omega(\partial X)$. (See Figure 1 for an example.)

We focus on two optimization problems for trees, the min-sum and min-max variants of computing optimal connected multicuts:

TreeMCut$_\Sigma$: Given a tree T and a conflict graph $\mathcal{C}$, compute a connected multicut $\mathcal{M}$ that minimizes its sum-cost $\mathrm{cost}_\Sigma(\mathcal{M})$.

ConTreeMCut$_{\max}$: Given a tree T and a conflict graph $\mathcal{C}$, compute a connected multicut $\mathcal{M}$ that minimizes its max-cost $\mathrm{cost}_{\max}(\mathcal{M})$.

We denote the optimal cost of each solution by $\mathrm{opt}_\Sigma(T, \mathcal{C})$ and $\mathrm{opt}_{\max}(T, \mathcal{C})$, respectively. In the TreeMCut$_\Sigma$ notation we do not indicate the connectivity requirement, because, as mentioned in the introduction, it's redundant: any multicut satisfying all other requirements can be converted into a connected one without increasing its sum-cost. This requirement is essential, however, for ConTreeMCut$_{\max}$ (see Appendix D).

Normal forms. We assume that all input trees are rooted. For a rooted tree T and a vertex v, by T_v we will denote the subtree of T rooted at v and containing all v's descendants.

Our algorithms will first simplify the input trees to streamline the computation. We say that T is in *normal form* if all its leaves are terminals and all internal non-terminals have at least two children. If we further require that all internal nodes are non-terminals (so Γ is exactly the set of leaves) and have exactly two children then we say that T is in *binary normal form* (See Figure 1.) It is known, and not difficult to prove that, after $O(n)$ pre-processing, without any loss of generality we can assume that a weighted tree is in binary normal form, and that an unweighted binary tree is in a normal form. This is formalized in the lemma below. For the sake of completeteness, we include its proof in Appendix B.

Lemma 2.1. *Let $T = (V, E, \omega)$ be a weighted tree and $\mathcal{C} = (\Gamma, \mathcal{F})$ a conflict graph. In time $O(n+|\mathcal{F}|)$ one can compute a tree $T' = (V', E', \omega')$ with $\Gamma \subseteq V'$ such that $\mathrm{opt}_\Sigma(T', \mathcal{C}) = \mathrm{opt}_\Sigma(T, \mathcal{C})$, $\mathrm{opt}_{\max}(T', \mathcal{C}) = \mathrm{opt}_{\max}(T, \mathcal{C})$, and*

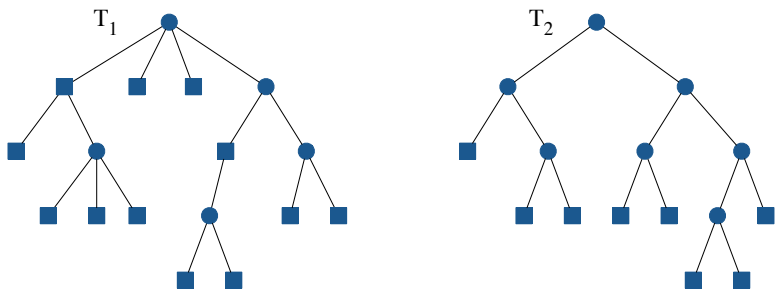


Figure 2: On the left, tree T_1 in normal form. On the right, tree T_2 in binary normal form. Terminals are represented by squares and non-terminals by circles.

(a) If T is unweighted, then T' is an unweighted tree in normal form,

(b) If T is weighted, then T' is in binary normal form and $\max_{e\in E'} \omega'(e) \leq \sum_{e\in E} \omega(e)$.

In both cases and for both objective functions, any multicut $\mathcal{M}'$ for T' can be converted into a multicut $\mathcal{M}$ for T in time $O(n)$, without increasing the cost.

A tree in normal form has $O(|\Gamma|)$ vertices, since non-terminals (except the root) have degree at least 3. Thus we may use $|\Gamma|$ and n interchangeably in the asymptotic formulas.

3 Algorithms for ConTreeMCut$_{\max}$

In this section we study algorithms for ConTreeMCut$_{\max}$, that is computing a connected multicut in trees with the min-max cost function, and we prove Theorems 1 and 2. To develop our algorithms, we proceed in two steps. In Section 3.1 we first identify a simpler optimization problem where the goal is only to compute one "local cut". This problem, while still $\mathbb{NP}$-hard itself, has a simple LP relaxation that can be used to produce a 2-approximate solution. Then, in Section 3.2, we give a polynomial-time Turing reduction from ConTreeMCut$_{\max}$ to this problem using dynamic programming. This will simultaneously prove Theorems 1 and 2.

Intuitions. A natural idea for solving ConTreeMCut$_{\max}$ would be to compute bottom-up a multicut for each rooted sub-tree: to compute a multicut $D(v)$ of a subtree T_v, first include in it the computed edges $D(u)$ for each $u \in ch(v)$, and then add some suitably chosen edges between v and its children to eliminate the conflicts between terminals in subtrees of different children of v. This works for unweighted trees (see Appendix A) but fails for the weighted case, simply because the edges from v to its children could be very expensive.

This observation leads to a somewhat unusual algorithm that, when processing a node v, computes a multicut for T_v using dynamic-programming entries associated with all *descendants* of v, rather than only with its children. Roughly, the information computed for v's descendants will be used to eliminate edges that are too expensive to be in the boundary of v's component. After excluding these edges (by simply contracting them), the component of v is then determined by computing its local cut. This is the essence of the Turing reduction mentioned in the previous paragraph.

Reductions. There are in fact a few Turing reductions in this section. To reduce a minimization problem $\mathbb{P}$ to a minimization problem $\mathbb{P}'$, given a parameter $\rho \geq 1$, the reductions we use have the following form:

- We assume the existence of an oracle for $\mathbb{P}'$ that, given an instance I' of $\mathbb{P}'$ and a value L', either returns a solution of cost at most $\rho L'$ or determines that the optimum cost of I' is larger than L'.
- The algorithm for $\mathbb{P}$ uses the oracle for $\mathbb{P}'$ and, given an instance I of $\mathbb{P}$ and a value L, it either returns a solution of cost at most ρL or determines that the optimum cost of I is larger than L. Furthermore, it runs in polynomial time, and each oracle query (I', L') satisfies $L' \leq L$.

Note that we allow $\rho = 1$ (in which case we just have a parameter-preserving reduction), and that L and L' may be real numbers. When weights are real, we only obtain a $(\rho + \epsilon)$-approximation for $\mathbb{P}$ from a ρ-approximation of $\mathbb{P}'$.

3.1 Computing Local Cuts

Given a tree $T = (V, E, \omega)$, a conflict graph $\mathcal{C}$, and a vertex $v^* \in V$, we define a *local cut for* v^* to be a subset $X \subseteq V$ such that (i) $v^* \in X$, (ii) $T[X]$ is connected (i.e., a subtree of T), and (iii) $X \cap \Gamma$ is an independent set in $\mathcal{C}$ (possibly empty). X is a *proper local cut* if $X \cap \Gamma \neq \emptyset$, and *improper* otherwise. We now define the local cut problem:

LCutTrees: Given a tree $T = (V, E, \omega)$, a conflict graph $\mathcal{C}$, and $v^* \in V$, compute a proper local cut $X \subseteq V$ for v^* minimizing $\omega(\partial X)$.

We let LCutTrees$^\circ$ be the problem variant where improper cuts are allowed, and we denote the optimum cost for both problems by $\mathrm{opt}_{\mathrm{loc}}(T, \mathcal{C}, v^*)$ and $\mathrm{opt}^\circ_{\mathrm{loc}}(T, \mathcal{C}, v^*)$, respectively. First, we observe that LCutTrees is not harder than LCutTrees$^\circ$.

Observation 3.1. *There is a polynomial-time Turing reduction from LCutTrees to LCutTrees$^\circ$ that makes $O(n)$ oracle queries on an input instance with n vertices. Moreover, if the input instance is unweighted, then every instance submitted to the oracle is unweighted.*

Proof. We actually prove a stronger fact: given an (approximate) algorithm for LCutTrees$^\circ$ and a vertex $u \in V$, we compute an (approximate) minimum weight local cut X for $(T, \mathcal{C}, v^*)$ that contains u (if one exists). We then try this procedure for each $u \in \Gamma$ and pick the solution of minimum weight.

Given u, we define $(T', \mathcal{C}', v^*)$ by contracting the unique path in T between u and v^* into a single vertex, then taking a graph quotient in $\mathcal{C}$ by merging all the terminals (if any) that appear on this path. (If two such terminals have a conflict in $\mathcal{C}$, then there is no local cut for v^* containing u.) We have an obvious correspondence between local cuts of T' and local cuts of T containing u, and clearly T' is unweighted if T is. □

Next, we give a 2-approximation algorithm for computing local cuts.

Lemma 3.2. *There is a polynomial-time 2-approximation algorithm for LCutTrees$^\circ$ (and thus also for LCutTrees, by Observation 3.1).*

Proof. Given an instance $(T, \mathcal{C}, v^*)$ with $T = (V, E, \omega)$, $\mathcal{C} = (\Gamma, \mathcal{F})$, and $v^* \in V$, we formulate an LP relaxation with a variable x_e for each edge $e \in E$, where $x_e = 1$ is intended to indicate that e lies in the boundary of a local cut. Letting P_{uv} denote the path in T between $u, v \in V$ (as an edge

set), the relaxation is as follows:

$$\text{minimize} \sum_{e\in E} \omega(e)x_e \tag{1}$$

$$\text{subject to} \sum_{e\in P_{sv^*}} x_e + \sum_{e\in P_{tv^*}} x_e \;\geq\; 1 \quad \forall st \in \mathcal{F} \tag{2}$$

$$x_e \;\geq\; 0 \quad \forall e \in E$$

We now describe a rounding procedure of a solution x for this linear program. Let L denote the optimum solution value. For $\rho \in (0,1)$ let $B_\rho = \{u \in V \mid x(P_{uv^*}) \leq \rho\}$, where $x(F) = \sum_{e\in F} x_e$ for a set $F \subseteq E$; clearly, $T[B_\rho]$ is connected and $v^* \in B_\rho$. We aim to show that for a proper choice of ρ, B_ρ is a local cut for v^* of cost at most $2L$.

We claim that $\int_0^1 \omega(\partial B_\rho)\, d\rho \leq L$. As $|x(P_{uv^*}) - x(P_{vv^*})| = x_{uv}$ for every $uv \in E$, we obtain

$$\begin{aligned}\int_0^1 \omega(\partial B_\rho)\, d\rho \;&=\; \int_0^1 \sum_{e\in\partial B_\rho} \omega(e)\, d\rho \;=\; \sum_{uv\in E} \omega(uv) \int_0^1 1_{uv\in\partial B_\rho}\, d\rho \\ &\leq \sum_{uv\in E} \omega(uv)|x(P_{uv^*}) - x(P_{vv^*})| \;=\; \sum_{uv\in E} \omega(uv)x_{uv} = L\end{aligned}$$

where 1_Π is an indicator function for a predicate Π.

By the above claim, there exists $\rho^* \in (0, \frac{1}{2})$ such that $\omega(\partial B_{\rho^*}) \leq 2L$. To finish the proof, it suffices to verify that $B_{\rho^*} \cap \Gamma$ is an independent set in $\mathcal{C}$. For any distinct $s, t \in B_{\rho^*} \cap \Gamma$, we have $x(P_{sv^*}) + x(P_{tv^*}) \leq 2\rho^* < 1$, so $st \notin \mathcal{F}$ by the LP constraint (2). Thus, B_{ρ^*} is the desired local cut for v^*. As there are at most n distinct sets B_ρ for $\rho \in (0, \frac{1}{2})$, namely those for values of ρ of the form $\rho = x(P_{vv^*})$ for $v \in V$, the radius ρ^* and the corresponding local cut B_{ρ^*} can be computed in polynomial time. □

There is an alternative way to prove Lemma 3.2, via a primal-dual algorithm instead of randomized rounding. We include it in Appendix E. That solution does not require solving a linear program.

We make one last observation about LCutTrees° on unweighted trees. As mentioned earlier, a reduction from VertexCover shows that TreeMCut$_\Sigma$ and ConTreeMCut$_{\max}$ have no $(2-\epsilon)$-approximation under standard assumptions, even on unweighted stars. What's interesting about LCutTrees° is that it is actually equivalent to VertexCover when T is unweighted. For a graph H, let $\mathrm{vc}(H)$ denote the minimum size of a vertex cover of H.

Observation 3.3. *Given an unweighted LCutTrees° instance $(T, \mathcal{C}, v^*)$, we can in linear time compute a graph H and number k' where $\mathrm{vc}(H) + k' = \mathrm{opt}^\circ_{\mathrm{loc}}(T, \mathcal{C}, v^*)$. Also, any vertex cover $A' \subseteq V(H)$ can be converted in linear time to a local cut of v^* with cost $|A'| + k'$.*

Proof. Root T at v^*. Let $A \subseteq ch(v^*)$ be the set of children u such that there are no conflicts between the terminals in $T_u \cup \{v^*\}$. Let $B = ch(v^*) - A$. Define $H = (A, E_H)$ by adding uw to E_H if there exist terminals $s \in T_u$ and $t \in T_w$ such that $st \in \mathcal{F}$. (H can easily be computed in linear time.)

We now claim that $\mathrm{vc}(H) + |B| = \mathrm{opt}^\circ_{\mathrm{loc}}(T, \mathcal{C}, v^*)$. To see this, observe that for any local cut $X \subseteq V$ of v^*, ∂X must contain at least one edge from $\{uv^*\} \cup E(T_u)$ for each $u \in B$, as otherwise X would have a conflict. Furthermore, $A' = \{u \in A : |\partial X \cap (\{uv^*\} \cup E(T_u))| \geq 1\}$ is a vertex cover for H. (If $uw \in E_H$ was not covered, then some conflicting pair in $T_u \cup T_w$ would be in X.) This implies $|\partial X| \geq \mathrm{vc}(H) + |B|$. And given a vertex cover A' for H, the set $X = \{v^*\} \cup \bigcup_{u\in A-A'} T_u$ with boundary $\partial X = \{uv^* : u \in A' \cup B\}$ is a local cut for v^*. □

3.2 Main Algorithm

In this section we prove Theorem 1, by reducing ConTreeMCut$_{\max}$ to LCutTrees$^\circ$.

Lemma 3.4. *There is a polynomial-time Turing reduction from* ConTreeMCut$_{\max}$ to LCutTrees$^\circ$.

To prove Lemma 3.4, we assume that there is a constant ρ and an oracle for LCutTrees$^\circ$ which, given an instance $(T', \mathcal{C}', v^*)$ and a real number L', either returns a solution of cost at most $\rho L'$ or determines that $\text{opt}^\circ_{\text{loc}}(T', \mathcal{C}', v^*) > L'$. Using this oracle we then describe an algorithm that, given an instance $(T, \mathcal{C})$ of ConTreeMCut$_{\max}$ and a real number L, either returns a multicut for $(T, \mathcal{C})$ of cost at most ρL, or determines that $\text{opt}_{\max}(T, \mathcal{C}) > L$. All queries to the oracle for LCutTrees use values $L' \leq L$.

In this section we assume that T is a tree in normal form, rooted at a vertex r. To streamline the presentation, we add to T a "dummy" parent vertex π_r with $\omega(r\pi_r) = 0$.

Intuitions. As a warm-up, we first show a simpler algorithm for ConTreeMCut$_{\max}$ with approximation ratio 8. The algorithm proceeds top-down, starting with the root r. Compute a (proper) local cut $X_r \subseteq V$ for r using the algorithm from Lemma 3.2. Then proceed recursively: for each vertex u with $u\pi_u \in \partial X_r$, compute a local cut X_u in T_u (that is, treating T_u as the input tree) using Lemma 3.2, and so on. The vertices u for which local cuts X_u are computed in this process will be called *hubs*. The computed multicut $\mathcal{M}$ for T is the set of all X_u's for hubs u.

Lemma 3.2 implies that for each hub u, the cost of the local cut X_u in T_u satisfies $\omega(\partial X_u) \leq 2 \max_{v \in V} \text{opt}_{\text{loc}}(T_v, \mathcal{C} \cap T_v, v)$. Every component of $\mathcal{M}$ has a boundary in the form $\partial X_u \cup \{u\pi_u\}$, for some hub u, so its boundary cost is $\omega(\partial X_u) + \omega(u\pi_u) \leq 4 \max_{v \in V} \text{opt}_{\text{loc}}(T_v, \mathcal{C} \cap T_v, v)$. To show an upper bound of 8 on the approximation ratio, it suffices to show that $\text{opt}_{\text{loc}}(T_u, \mathcal{C} \cap T_u, u) \leq 2 \cdot \text{opt}_{\max}(T, \mathcal{C})$ for each vertex u. This inequality follows by considering the component X of the optimal multicut for T that contains u. Let $X' = X \cap V(T_u)$. This X' is a local cut for u in T_u, whose boundary within T_u has cost $\omega(\partial_{T_u} X') \leq \text{opt}_{\max}(T, \mathcal{C})$. If X' is not proper, we can merge X' with any adjacent component below it (which exists, by the normal form, and must be proper), obtaining a proper local cut for u in T_u and at most doubling the cost.

While the bounds in this analysis are tight in the worst case, the extra factors of 2 are really due to a poor choice of components. If one considers an optimal multicut $\mathcal{M}$ of T with min-max cost L, and we take any component $X \in \mathcal{M}$ with common ancestor $u \neq r$, then X is a proper local cut for u in T_u with cost at most $L - \omega(u\pi_u)$. This observation suggests a bottom-up approach: first try to compute min-max multicuts for the sub-trees, then compute a local cut for the root using *only* edges $u\pi_u$ for which T_u admits a "good" solution.

To formalize this idea, we need a slight strengthening of Observation 3.1. For $E' \subseteq E$ and $v^* \in V$, we define an *E'-restricted proper local cut for v^** (or just an *E'-restricted cut*, for short) to be a proper local cut X for v^* such that $\partial X \subseteq E'$. Let $\text{opt}_{\text{loc}}(T, \mathcal{C}, v^*, E')$ denote the minimum weight of all E'-restricted proper local cuts for v^*. (If there is no such cut, let $\text{opt}_{\text{loc}}(T, \mathcal{C}, v^*, E') = \infty$.)

We observe that computing such a cut for v^* can be reduced to $O(n)$ instances of LCutTrees$^\circ$, though this is somewhat subtle[3]. As in Observation 3.1, if v^* is not already a terminal, guess a terminal $t \in \Gamma$ and contract the edges on the path from v^* to t, merging terminals in $\mathcal{C}$ as needed. If two conflicting terminals were merged, then there is no local cut without separating v^* and t. Otherwise, we identify v^* and t and contract the edges $E - E'$ and merge any terminals in $\mathcal{C}$ if they got merged in T. If we merge two terminals that conflict, this does *not* necessarily mean that there is no solution, but merely that v^* must be separated from the merged terminal u, in this case we

[3]We could trivially reduce to LCutTrees by increasing the weights of $E - E'$, but this doesn't work if we want an unweighted tree.

simply add uv^* to $\mathcal{F}$. But if we try to merge v^* and t' and $t'v^* \in \mathcal{F}$, then there is no E'-restricted cut without separating v^* and t. If the previous checks pass, then let $(T'_t, \mathcal{C}'_t)$ be the modified tree and conflict graph. Then by inspection, $\mathrm{opt}_{\mathrm{loc}}(T, \mathcal{C}, v^*, E') = \min_t \mathrm{opt}^{\circ}_{\mathrm{loc}}(T'_t, \mathcal{C}'_t, v^*)$, where t ranges over all terminals where $(T'_t, \mathcal{C}'_t)$ is defined.

We are now ready to describe the reduction, which is simply a dynamic program.

Proof of Lemma 3.4. Let L be a real number denoting our target cost and $\rho \geq 1$ be the approximation ratio of our LCutTrees$^\circ$ oracle. From the discussion above, we may assume that this oracle can compute E'-restricted proper local cuts: given a number $L' \leq L$ and a tuple $(T, \mathcal{C}, v^*, E')$, where $v^* \in V$ and $E' \subseteq E$, the oracle either returns an E'-restricted cut for v^* of cost at most $\rho L'$ or determines that $\mathrm{opt}_{\mathrm{loc}}(T, \mathcal{C}, v^*, E') > L'$.

For $v \in V$, define an (L, ρ)-*approximate multicut for* T_v as a multicut $\mathcal{Q}$ for $(T_v, \mathcal{C} \cap T_v)$ with $\mathrm{cost}_{\max}(\mathcal{Q}) \leq \rho L$ and with v's component $\mathcal{Q}(v)$ satisfying a stricter requirement $\omega(\partial \mathcal{Q}(v)) \leq \rho(L - \omega(v\pi_v))$. For each $v \in V$ the algorithm will compute a table entry D_v whose value is either an (L, ρ)-approximate multicut for T_v or NULL (undefined). Let

$$\begin{aligned} E^1_v &= \{u\pi_u : u \in V(T_v) - \{v\} \text{ and there is an } (L,1)\text{-approximate multicut for } T_u\} \\ E^\rho_v &= \{u\pi_u : \ u \in V(T_v) - \{v\} \text{ and } D_u \text{ is an } (L,\rho)\text{-approximate multicut for } T_u\} \end{aligned}$$

We maintain the following invariant for each vertex $v \in V$:

($\star$) If there is an $(L, 1)$-approximate multicut for T_v, then D_v is (L, ρ)-approximate.

The validity of ($\star$) for $v = r$ ensures that the existence of a multicut in T of cost at most L implies that D_r is a multicut of cost at most ρL, proving the correctness of the reduction.

We now describe an algorithm that guarantees the validity of the invariant ($\star$) at each node v. The computation proceeds bottom-up, starting from the leaves. If v is a leaf, set $D_v = \{\{v\}\}$ if $\omega(v\pi_v) \leq L$, and set $D_v =$ NULL otherwise. Then ($\star$) holds for v.

Now suppose that v is an internal vertex and that ($\star$) holds for all $u \in V(T_v) - \{v\}$. This implies $E^1_v \subseteq E^\rho_v$. Using the oracle for LCutTrees, we now try to compute an E^ρ_v-restricted cut X_v in T_v such that $\partial X_v \subseteq E^\rho_v$ and $\omega(\partial X_v) \leq \rho(L - \omega(v\pi_v))$. If the oracle fails, set $D_v =$ NULL; otherwise it returns X_v and set

$$D_v = \{X_v\} \cup \bigcup_{u\pi_u \in \partial X_v} D_u.$$

We first argue that if the oracle succeeds, then D_v is an (L, ρ)-approximate multicut for T_v. Indeed, the component of D_v containing v is X_v and by construction, $\omega(\partial X_v) \leq \rho(L - \omega(v\pi_v))$. Consider now a component $X \in D_v$ not containing v, and let u be the common ancestor of its nodes in X. Then if $u\pi_u \in \partial X_v$, we have $\omega(\partial X) \leq \rho(L - \omega(u\pi_u)) + \omega(u\pi_u) \leq \rho L$, and otherwise $\omega(\partial X) \leq \rho L$ by the inductive assumption.

Therefore, to maintain ($\star$), it suffices to show that if there is an $(L, 1)$-approximate multicut for T_v, then the oracle succeeds. This is straightforward: suppose we have an $(L, 1)$-approximate multicut and let $Y \subseteq T_v$ be its component containing v. By definition of $(L, 1)$-approximate multicuts, Y contains a terminal, $\omega(\partial Y) \leq L - \omega(v\pi_v)$, and $\partial Y \subseteq E^1_v \subseteq E^\rho_v$. Thus, Y is an E^ρ_v-restricted proper local cut for v; in other words $\mathrm{opt}_{\mathrm{loc}}(T_v, C \cap T_v, v, E^\rho_v) \leq \omega(\partial Y) \leq L - \omega(v\pi_v)$. By the guarantee of our oracle, it succeeds, returning an E^ρ_v-restricted proper local cut X_v for v with $\omega(\partial X_v) \leq \rho(L - \omega(v\pi_v))$, as desired. □

Proof of Theorem 1. Combine Lemma 3.2 and Lemma 3.4 with binary search. For the weighted case we need an initial upper bound on the optimum, which can be obtained from the 8-approximation sketched earlier or the 2-approximation for TreeMCut$_\Sigma$ [18], which gives a multicut of min-max cost at most $2|\Gamma|$ times the optimum. □

Proof of Theorem 2. By combining Observation 3.3, Observation 3.1, and Lemma 3.4, we obtain a polynomial-time reduction from unweighted ConTreeMCut$_{\max}$ to VertexCover that is approximation- and parameter-preserving, in the sense described in the beginning of Section 3. The theorem then follows by using the vertex cover algorithms from [21, 9, 5]. (Regarding the $O^*(1.1996^{|\Gamma|})$ time bound, one can observe that all vertex cover instances computed are graph quotients of $\mathcal{C}$ and thus have at most $|\Gamma|$ vertices.) □

A straightforward implementation of our 2-approximation algorithm (solving LCutTrees$^\circ$ instances with vertex cover) would take time $\tilde{O}(n^2(n + |\mathcal{F}|))$. In Appendix A we give an alternative algorithm whose running time is nearly linear.

4 Polynomial-Time Algorithm for ConTreeMwCut$_{\max}$

In this section we prove Theorem 3; that is we give a polynomial-time algorithm for ConTreeMwCut$_{\max}$, the multiway-cut version of ConTreeMCut$_{\max}$. The result in Theorem 3 is analogous to the result for TreeMwCut$_\Sigma$ in papers [8, 16, 22, 10], although the min-max cost introduces challenges not present in the min-sum version.

At a high-level, rather than computing the optimum solution directly, the algorithm uses an $O(n)$-time dynamic-programming subroutine that, given a real number L, either returns a multiway cut of min-max cost at most L or determines that the optimum min-max cost is greater than L. Using this subroutine, we can compute an optimal multiway cut for integer weights by binary search. To obtain a strongly polynomial-time algorithm for real weights, we view our dynamic program as computing parametric functions over L, which we then show can be represented using only $O(n)$ space per function.

Notation. We assume that T is in binary normal form with root r. We also add a dummy parent π_r to r and set $\omega(r\pi_r) = 0$. For a vertex $v \in V$, let $T_v^+ = T_v \cup \{\pi_v\}$, that is the sub-tree rooted at v plus v's parent.[4] A T_v^+-*cut* is a partition $\mathcal{M}$ of T_v^+ satisfying the same conditions as a connected multicut, with the possible exception that the component $\mathcal{M}(\pi_v)$ may not contain a terminal. We say $\mathcal{M}$ is *proper* if $\mathcal{M}(\pi_v)$ does contain a terminal and *improper* otherwise. In particular, a proper T_r^+-cut is just a connected multicut of T. When v is clear from context, we often call $\mathcal{M}(\pi_v)$ the "top component" of $\mathcal{M}$. To avoid clutter, for a T_v^+-cut $\mathcal{M}$ the "boundaries" $\partial\mathcal{M}(\pi_v)$ and $\partial\mathcal{M}$ always refer to the boundary edges restricted to $E(T_v^+)$. For a non-top component in $\mathcal{M}$, its boundaries in T and T_v^+ are identical.

Algorithm for integer weights. For each $v \in V$ we define two functions $D_v^0, D_v^1 \colon \mathbb{R}^+ \to \mathbb{R}^+$. $D_v^1(L)$ is the minimum, over all proper T_v^+-cuts $\mathcal{M}$ with all non-top boundaries of weight at most L, of the boundary weight $\omega(\partial\mathcal{M}(\pi_v))$ of its top component in T_v^+. (When no T_v^+-cut satisfies these constraints, then $D_v^i(L) = \infty$.) $D_v^0(L)$ is defined analogously, but over improper T_v^+-cuts. We colloquially refer to these as the "D-functions".

We first fix L, and show a dynamic programming algorithm that computes $D_v^0(L)$ and $D_v^1(L)$ for all $v \in V$. These values are stored in a table, and computed in the bottom-up order using a recurrence equation. If t is a leaf (that is, $t \in \Gamma$), we have $D_t^1(L) = 0$ (no edges cut), and $D_t^0(L)$ equals $\omega(t\pi_t)$ if $\omega(t\pi_t) \le L$ (cut the edge $t\pi_t$) and is ∞ otherwise. For an internal vertex $v \in V$

[4] One can formulate a dynamic program using T_v instead of T_v^+, but the latter is slightly more convenient.

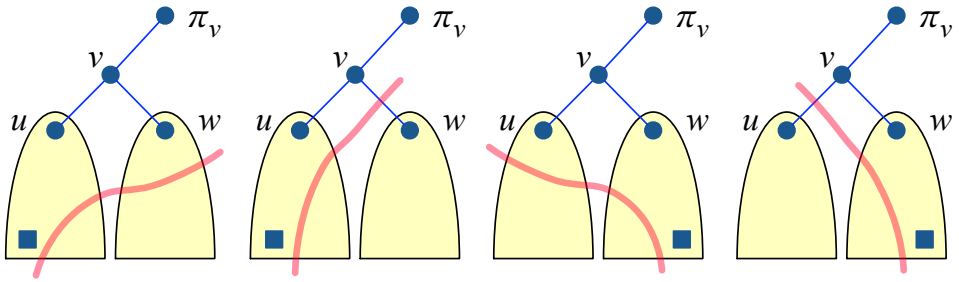


Figure 3: Illustration of equation (3) for $D_v^1(L)$. The wide orange (light) curve represents the boundary of the top component of $D_v^1(L)$. The two first cases correspond to $\sigma = 1$, the two other cases correspond to $\sigma = 0$.

with children u and w, we use the following recurrences:

$$D_v^1(L) = \min_{\sigma \in \{0,1\}} \left\{ D_u^\sigma(L) + D_w^{1-\sigma}(L) \right\} \tag{3}$$

$$D_v^0(L) = \begin{cases} \min\left\{ D_u^0(L) + D_w^0(L), \omega(v\pi_v) \right\} & \text{if } \omega(v\pi_v) + D_v^1(L) \leq L \\ D_u^0(L) + D_w^0(L) & \text{else} \end{cases} \tag{4}$$

Figure 3 illustrates equation (3) for $D_v^1(L)$. For $\sigma = 1$, the top component of $D_v^1(L)$ is obtained from a top (proper) component of $D_u^1(L)$ and the (improper) top component (which may or may not contain w) of $D_w^0(L)$. The case of $\sigma = 0$ is symmetric. The intuition behind the recurrence for $D_v^0(L)$ is similar, except that we also need to take into account the case that its top compoment may only contain π_v.

From the definitions, we have that $\text{opt}_{\max}(T, \mathcal{C}) \leq L$ if and only if $D_r^1(L) \leq L$. Thus, checking whether $\text{opt}_{\max}(T, \mathcal{C}) \leq L$ can be done with a subroutine that computes the D-values at all nodes of the tree by the recurrence (3) and (4), and then checks if $D_r^1(L) \leq L$.

If the weights are integer, the overall algorithm can compute the optimal value L by performing binary search for L in the interval $[0, W]$, in each iteration using the above subroutine to test whether $\text{opt}_{\max}(T, \mathcal{C}) \leq L$.

To complete the proof of Theorem 3 for integer weights, we still need to justify that all formulas for $D_v^0(L)$ and $D_v^1(L)$ are correct. The proof is by induction. If v is a leaf, the correctness follows directly from definition. So assume that v is an internal vertex.

To streamline the argument, some auxiliary terminology will be helpful. Suppose v has children u and w. A T_v^+-cut is *L-feasible* if all its non-top boundaries have weight at most L. If $\mathcal{Q}_u$ and $\mathcal{Q}_w$ are a T_u^+-cut and T_w^+-cut respectively with top components X_u and X_w, and one of them is improper, their *merge* is a T_v^+-cut $\mathcal{Q}_v$ consisting of all non-top components of $\mathcal{Q}_u$ and $\mathcal{Q}_w$ and top component $X_v = X_u \cup X_w \cup \{\pi_v\}$. Clearly, $|X_v \cap \Gamma| \leq 1$ and $\omega(\partial X_v) = \omega(\partial X_u) + \omega(\partial X_w)$. (Recall our convention that the boundary of a top component of a subtree T_z^+ includes only edges in T_z^+.) Also, $\mathcal{Q}_v$ is L-feasible iff both $\mathcal{Q}_u$ and $\mathcal{Q}_w$ are L-feasible. Further, *any* T_v^+-cut $\mathcal{Q}_v$ (with top component X_v containing v) induces a T_u^+-cut $\mathcal{Q}_u$ and a T_w^+-cut $\mathcal{Q}_w$ such that $\mathcal{Q}_v$ is a merge of $\mathcal{Q}_u$ and $\mathcal{Q}_w$: assign each non-top component to $\mathcal{Q}_u$ or $\mathcal{Q}_w$, depending on in which subtree it is contained, and let the respective top components be $X_u = X_v \cap V(T_u^+)$ and $X_w = X_v \cap V(T_w^+)$.

We are now ready to continue the proof. Consider first $D_v^1(L)$. Recall that in the normal form all terminals are leaves. By the above discussion, any T_v^+-cut $\mathcal{Q}_v$ is proper and L-feasible if and only if it is a merge of an L-feasible T_u^+-cut $\mathcal{Q}_u$ and an L-feasible T_w^+-cut $\mathcal{Q}_w$, one proper and

one improper. The correctness of (3) follows from the earlier established relation between the top boundaries, namely $\omega(\partial X_v) = \omega(\partial X_u) + \omega(\partial X_w)$.

Next, consider $D_v^0(L)$. We distinguish two disjoint types of improper L-feasible T_v^+-cuts $\mathcal{Q}_v$: (i) those whose top component X_v is $X_v = \{\pi_v\}$ and (ii) those where $X_v \neq \{\pi_v\}$. It is sufficient to show that the two options in the formula (4) correctly compute the optimal improper L-feasible T_v^+-cuts of these two types, respectively.

Regarding type (i), let $\mathcal{Q}_v$ be an improper L-feasible T_v^+-cut. Observe that $\mathcal{Q}_v$ is of type (i) iff there is a *proper* L-feasible T_v^+-cut $\mathcal{Q}'_v$ such that the non-top components of $\mathcal{Q}_v$ consist of all non-top components of $\mathcal{Q}'_v$ and one additional component $X''_v = X'_v \setminus \{\pi_v\}$ that satisfies $\omega(\partial X''_v) = \omega(v\pi_v) + \omega(\partial X'_v) \leq L$, where X'_v is the top component of $\mathcal{Q}'_v$. (We add $\omega(v\pi_v)$ because the boundary of X''_v, treated as a non-top component in $\mathcal{Q}_v$, also includes $v\pi_v$. Note that there is no type (i) T_v^+-cut if this inequality is false.) This shows the correctness of the top option in (4).

The argument for type (ii) is easier: By the earlier discussion, T_v^+-cut $\mathcal{Q}_v$ is of type (ii) if and only if it is a merge of an improper L-feasible T_u^+-cut $\mathcal{Q}_u$ and an improper L-feasible T_w^+-cut $\mathcal{Q}_w$. Then again the correctness of (4) for type (ii) follows from the earlier established relation between the top boundaries, namely $\omega(\partial X_v) = \omega(\partial X_u) + \omega(\partial X_w)$.

Extension to real weights. The key idea is not to deal with *individual* values $D_v^\sigma(L)$ of the functions, but rather to deal with the *entire* functions $D_v^\sigma(\cdot)$. This is possible because the functions have a very simple structure.

Claim 4.1. *For each $v \in V$ and $\sigma \in \{0,1\}$, the function $D_v^\sigma(\cdot)$ is non-increasing, right-continuous, and piece-wise constant.*[5] *Furthermore, all points of discontinuity of all D-functions are from some set $\mathcal{B} \subseteq \mathbb{R}^+$ of size $n = |V|$.*

Proof. The fact that D_v^σ is non-increasing follows immediately from its definition. To prove the rest of the claim, it is useful to introduce a set of auxiliary functions H_v: for $v \in V$, let $H_v(L) = \omega(v\pi_v)$ if $\omega(v\pi_v) + D_v^1(L) \leq L$, and otherwise let $H_v(L) = \infty$. Then for $t \in \Gamma$, $D_t^0(L) = H_t(L)$, and for each internal vertex $v \in V \setminus \Gamma$ with children u, w, $D_v^0(L) = \min\left\{D_u^0(L) + D_w^0(L), H_v(L)\right\}$. By induction, all D-functions are some combination of H-functions using component-wise min and + operations.

The condition in the definition of H_v can be phrased as an optimization problem: $\omega(v\pi_v) + D_v^1(L) \leq L$ if and only if there is an *improper* T_v^+-cut which cuts $v\pi_v$ and has min-max cost (in T_v^+) at most L. (Or equivalently, the component containing π_v is a singleton. In this interpretation, $D_v^1(L)$ corresponds to the boundary of the component containing v in such a cut.) Then $L_v \coloneqq \min\left\{L : \omega(v\pi_v) + D_v^1(L) \leq L\right\}$ is the minimum max-cost over all such cuts, and $H_v(L) = \omega(v\pi_v)$ if $L \geq L_v$, and $H_v(L) = \infty$ otherwise. That is, H_v is just a right-continuous step function with at most one point of discontinuity. Then, by induction, the points of discontinuity of D_v^σ form a subset of $\mathcal{B}_v^\sigma$, where $\mathcal{B}_v^1 = \{L_u\}_{u \in T_v - \{v\}}$ and $\mathcal{B}_v^0 = \mathcal{B}_v^1 \cup \{L_v\}$. It also follows that D_v^σ is piece-wise constant and right-continuous. □

In the algorithm, we represent a piece-wise constant right-continuous function f by a list $\{(a_0, f(a_0)), ..., (a_l, f(a_l))\}$ where $a_1 < ... < a_l$ are the points of discontinuity and $a_0 = 0$. For $x \geq 0$, $f(x) = f(a_i)$, where i is the largest index such that $a_i \leq x$. Given list-representations for functions f and g with discontinuity sets A and B respectively, we can compute a list representation for their component-wise minimum or sum, whose discontinuities are in $A \cup B$, in $O(|A| + |B|)$ time. Crucially, because the points of discontinuity among our D-functions come from a subset of size n, we can perform these calculations in $O(n)$ time.

[5] To account for ∞-values, "right-continuous" in this context means that for each $0 \leq L < \infty$, there is some $\epsilon > 0$ such that $D_v^i(L + \delta) = D_v^i(L)$ for all $0 < \delta < \epsilon$.

Algorithm 1 Exponential-time algorithm for computing $\text{opt}_\Sigma(T, \mathcal{C})$.

1: Let $\mathcal{S}$ be the family of all independent sets of $\mathcal{C}$, including $\emptyset$
2: For each $v \in V$ and $s \in \mathcal{S}$, create a table entry $D_v^s \in \mathbb{R}^+ \cup \{\infty\}$
3: **for** $t \in \Gamma$ **do** ▷ initialize D_t for all leaves $t \in \Gamma$
4: $\quad D_t^{\{t\}} \leftarrow 0$, $D_t^* \leftarrow 0$, $D_t^\emptyset \leftarrow \omega(t\pi_t)$, $D_t^s \leftarrow \infty$ for $s \in \mathcal{S} \setminus \{\{t\}, \emptyset\}$
5: **for** $v \in V - \Gamma$, processing bottom-up **do**
6: $\quad$ Let u and w be v's children
7: $\quad$ **for** $s \in \mathcal{S} - \{\emptyset\}$ **do**
8: $\quad\quad D_v^s \leftarrow \min_{a \sqcup b = s} \{D_u^a + D_w^b\}$ ▷ $a \sqcup b = s$ is a partition of s
9: $\quad D_v^* \leftarrow \min_{s \neq \emptyset} D_v^s$
10: $\quad D_v^\emptyset \leftarrow \min\{ D_u^\emptyset + D_w^\emptyset , \omega(v\pi_v) + D_v^* \}$.
11: **return** D_r^*

The final algorithm is this: for each $v \in V$ and $\sigma \in \{0, 1\}$, compute the value L_v and a list representation of D_v^σ over discontinuities $\mathcal{B}_v^\sigma$. Then return L_r, which is equal to $\text{opt}_{\max}(T, \mathcal{C})$. Computing lists for the terminals is easy, so suppose v is an internal vertex with children u, w. Computing a list for D_v^1 takes $O(n)$ time from functions D_u^0, D_w^0, D_u^1 and D_w^1, using equation (3). To compute a list for D_v^0, we first need to compute L_v from D_v^1. To do this, suppose D_v^1 has points of discontinuity $a_1, ..., a_l$ and $a_0 = 0$. We then claim $L_v = \min_{0 \le i \le l} a'_i$, where $a'_i = \max\{a_i, \omega(v\pi_v) + D_v^1(a_i)\}$. Indeed, if j is the greatest index where $a_j \le L_v$, then $\omega(v\pi_v) + D_v^1(a_j) \le L_v$ and $a'_j \le L_v$. And by monotonicity, $D_v^1(a'_i) + \omega(v\pi_v) \le a'_i$ for all i, so the formula is correct. Then using the equation $D_v^0(L) = \min\{D_u^0(L) + D_w^0(L), H_v(L)\}$ we compute a list for D_v^0. Clearly, we spend $O(n)$ computing a list for each D_v^σ, and thus the total running time is $O(n^2)$, completing the proof of Theorem 3.

5 Algorithms Parameterized by Neighborhood Diversity

In this section we prove Theorem 4(a). We assume that our tree T is in binary normal form with root r. We add a dummy parent π_r for r and set $\omega(r\pi_r) = 0$. We will also use the notion of T_v^+-cuts and their boundary notation as introduced in Section 4.

To motivate our algorithm, we briefly sketch a simple (albeit inefficient) dynamic program for TreeMCut$_\Sigma$. Its pseudo-code is presented in Algorithm 1. D_v^s and D_v^* have a natural interpretation: for an independent set $s \subseteq \Gamma$, D_v^s is the minimum sum-cost over all T_v^+-cuts where the top cluster contains exactly the terminals from s, while D_v^* is the minimum sum-cost over all proper T_v^+-cuts.

Without giving a complete analysis, we quickly justify the recurrences in Lines 8 and 10. For vertex $v \in V - \Gamma$ with children u and w, let $\mathcal{M}$ be a T_v^+-cut whose top component satisfies $\mathcal{M}(\pi_v) \cap \Gamma = s$. If $v\pi_v \notin \partial\mathcal{M}$ (which holds when $s \neq \emptyset$), then restricting the components of $\mathcal{M}$ to T_u^+ and T_w^+ yields a T_u^+-cut $\mathcal{M}_u$ and a T_w^+-cut $\mathcal{M}_w$, respectively. The terminals in the corresponding top components $\mathcal{M}_u(\pi_u)$ and $\mathcal{M}_w(\pi_w)$ partition s (perhaps trivially, one or both components may have no terminals), while the boundary edges $\partial\mathcal{M} = \partial\mathcal{M}_u \sqcup \partial\mathcal{M}_w$ get partitioned. This justifies Line 8 and the first term in Line 10. When $v\pi_v \in \partial\mathcal{M}$, $\mathcal{M}$ is improper and removing $v\pi_v$ from the boundary creates a proper T_v^+-cut, justifying the second term in Line 10.

The runtime is dominated by the number of independent sets in $\mathcal{C}$ and Line 8, so it's at most

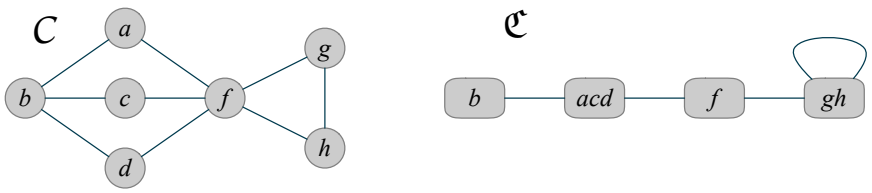


Figure 4: An example of a conflict graph $\mathcal{C}$ and its type graph $\mathfrak{C}$. The types are labeled by the vertices of $\mathcal{C}$ mapped to them. Since $\mathfrak{C}$ has four vertices (types), $\mathrm{nd}(\mathcal{C}) = 4$.

$O(|\mathcal{S}|^2 n)$ in the worst case.[6]

A natural idea for speeding up the algorithm is to group independent sets into equivalence classes, as few as possible, that can be used as states in dynamic programming. Roughly, two independent subsets s, s' of Γ can be considered equivalent if, for any vertex v, swapping s and s' in the top cluster of v will not affect the computation in the ancestors of v. To generalize the recurrence in line 8 of the algorithm, this equivalence relation needs to be preserved under disjoint unions. For our algorithm, we define this equivalence relation on independent sets in terms of a *type graph* and *neighborhood diversity* [29], which we define next. This relation is in general not optimal, but it applies to a broad class of graphs and can be computed efficiently. Later, in Section 6 we consider a natural generalization of these concepts that allows us to establish Corollary 1.1.

Neighborhood diversity. For a conflict graph $\mathcal{C} = (\Gamma, \mathcal{F})$, two terminals $t, t' \in \Gamma$ are *of the same type* (also called *twins*) if $N(t') - \{t\} = N(t) - \{t'\}$, where $N(\cdot)$ is the open neighborhood in $\mathcal{C}$. This property forms an equivalence relation, whose congruence classes $\mathfrak{N}$ are called *types*. The number of types $\mathrm{nd}(\mathcal{C}) = |\mathfrak{N}|$ is then the *neighborhood diversity* of $\mathcal{C}$. We treat types as abstract elements instead of subsets of terminals, using the map $\tau\colon \Gamma \to \mathfrak{N}$ to denote which type $\tau(t)$ that terminal t is assigned to. Associated with the types is a unique *type graph* $\mathfrak{C} = (\mathfrak{N}, \mathfrak{F}, \tau)$, which has the property that for all $t, t' \in \Gamma$, $tt' \in \mathcal{F}$ if and only if $\tau(t)\tau(t') \in \mathfrak{F}$. (A type $\eta \in \mathfrak{N}$ may have a self-loop in $\mathfrak{C}$, which we denote by $\eta\eta \in \mathfrak{F}$.) Intuitively, the types form a partition of Γ where (i) each part is either a clique or independent set and (ii) between each pair of parts is either a biclique or no edges at all. It is a standard fact that $\mathfrak{C}$ can be computed in $O(n + |\mathcal{F}|)$ time (see Figure 4 for an example).

States and splits. Let $\mathfrak{N}^{(\mathrm{c})} \subseteq \mathfrak{N}$ denote the types with self-loops (*c-types*) and $\mathfrak{N}^{(\mathrm{i})} = \mathfrak{N} - \mathfrak{N}^{(\mathrm{c})}$ (*i-types*). Note that the terminal set $\tau^{-1}(\eta)$ is a clique when η is a c-type and is an independent set otherwise. Letting $\tau(\Gamma') = \{\tau(t) \mid t \in \Gamma'\}$ be the image of $\Gamma' \subseteq \Gamma$ under τ, we observe that τ maps independent sets of $\mathcal{C}$ to independent sets of $\mathfrak{C}$ (with self-loops allowed). From here on we refer to the latter as *states* and denote its elements $s \in \mathcal{I}(\mathfrak{C})$ using lower-case letters. (We include $\emptyset \in \mathcal{I}(\mathfrak{C})$.) Naturally, $\mathcal{I}(\mathfrak{C})$ will replace $\mathcal{I}(\mathcal{C})$ in Algorithm 1, but to do so requires obtaining an analog of Line 8; here we need a correspondence between bipartitions $\Gamma' = A \sqcup B$ of an independent set $\Gamma' \in \mathcal{I}(\mathcal{C})$ and pairs of states associated with $\tau(\Gamma')$. Crucially, we in general do *not* want to consider bipartitions of $\tau(\Gamma')$: while the c-types in $\tau(\Gamma')$ will be partitioned between $\tau(A)$ and $\tau(B)$, if $\eta \in \tau(\Gamma')$ is an i-type, then it's possible that A and B could *both* contain terminals from $\tau^{-1}(\eta)$.

This motivates the following definitions. For $s \in \mathcal{I}(\mathfrak{C})$, s^{c} and s^{i} are the c-types and i-types in s respectively, so $s = s^{\mathrm{c}} \sqcup s^{\mathrm{i}}$. The *splits of* s denotes the collection of pairs $\mathfrak{X}(s) = \big\{(a, b) \in \mathscr{P}(\mathfrak{N})^2 \mid a \cup b = s \text{ and } a^{\mathrm{c}} \cap b^{\mathrm{c}} = \emptyset\big\}$. In particular, we allow $a^{\mathrm{i}} \cap b^{\mathrm{i}} \neq \emptyset$.

Our algorithm does not depend so much on the particulars of neighborhood diversity or type

[6] In terms of $|\Gamma|$, one can see that the running time is in fact $O(3^{|\Gamma|} n)$, since Line 8 is just a min-sum subset convolution. Note also that independent sets can be enumerated in $O(|\mathcal{S}| n)$ time [28].

graphs. In fact, we only need the three following properties:

(A1) For all $A \in \mathcal{I}(\mathcal{C})$, $A = \emptyset$ if and only if $\tau(A) = \emptyset$.

(A2) For all $B \in \mathcal{I}(\mathcal{C})$ and $A \subseteq B$, $(\tau(A), \tau(B - A)) \in \mathfrak{X}(\tau(B))$.

(A3) For all $A, B \in \mathcal{I}(\mathcal{C})$ where $A \cap B = \emptyset$, $(\tau(A), \tau(B)) \in \mathfrak{X}(s)$ implies $A \cup B \in \mathcal{I}(\mathcal{C})$ and $\tau(A \cup B) = s$.

Intuitively, (A1) just says that τ distinguishes the empty set in $\mathcal{I}(\mathcal{C})$, (A2) says we can convert bipartitions of $B \in \mathcal{I}(\mathcal{C})$ into splits of $\tau(B)$, (A3) says that $A \sqcup B$ is automatically independent if $(\tau(A), \tau(B))$ is a split of some state (and that $\tau(A \cup B)$ must be this state). That these properties hold is straightforward to verify.

The runtime of our algorithm is dominated by the total number of splits or *split complexity*, denoted by $\mathrm{sp}(\mathfrak{C}) = \sum_{s \in \mathcal{I}(\mathfrak{C})} |\mathfrak{X}(s)|$. It is easy to see that $\mathfrak{X}(s)$ and $\mathfrak{X}(s')$ are disjoint for two distinct states s, s', so the total number of splits is $\mathrm{sp}(\mathfrak{C}) \leq |\mathcal{I}(\mathfrak{C})|^2 \leq 4^{\mathrm{nd}(\mathcal{C})}$.

Algorithm for TreeMCut$_\Sigma$. As mentioned before, we can compute $\mathcal{C}$'s type graph $\mathfrak{C}$ in $O(n + |\mathcal{F}|)$ time. The main algorithm will actually run in time $O(\mathrm{sp}(\mathfrak{C})n)$, which we do by enumerating $\mathcal{I}(\mathfrak{C})$ and the splits directly. Enumerating independent sets can be done in $O(|\mathcal{I}(\mathfrak{C})||\mathfrak{N}|) = O(\mathrm{sp}(\mathfrak{C})n)$ time, and for each $s \in \mathcal{I}(\mathfrak{C})$, $\mathfrak{X}(s)$ can be computed in $O(|\mathfrak{X}(s)|)$ time by listing all bipartitions of s^{c} and pairs of subsets of s^{i}.

We now just need to modify Algorithm 1 as follows: $\mathcal{S}$ is now $\mathcal{I}(\mathfrak{C})$, every instance of $\{t\}$ in Line 4 is replaced by $\{\tau(t)\}$, and the min-sum convolution formula in Line 8 is replaced by $\min_{(a,b) \in \mathfrak{X}(s)} \left\{D_u^a + D_w^b\right\}$. We also reinterpret the meaning of D_v^s and D_v^*, as follows. We say a T_v^+-cut $\mathcal{M}$ is in state $s \in \mathcal{S}$ if the top component satisfies $\tau(\mathcal{M}(\pi_v) \cap \Gamma) = s$. Then D_v^s is the minimum sum-cost over all T_v^+-cuts in state s, and by property (A1) a T_v-cut is in state $\emptyset$ if and only if it is improper, so D_v^* is still the minimum sum-cost over all proper T_v^+-cuts.

The correctness of D_t^s for leaves $t \in \Gamma$ in Line 4 is easy to verify. The formula in Line 10 has not changed, so we only need to justify the new formula for Line 8. Let $\mathcal{M}$ be a T_v^+-cut in state $s \neq \emptyset$ with sum-cost D_v^s, and for $c \in ch(v) = \{u, w\}$, let $\mathcal{M}_c$ be the induced T_c^+-cut by restricting $\mathcal{M}$ to T_c^+. Since $\mathcal{M}$ is proper, $v\pi_v \notin \partial\mathcal{M}$ and $\mathcal{M}_c(\pi_c) = \mathcal{M}_(\pi_v) \cap T_c^+$. Letting $\Gamma_c = \mathcal{M}_c(\pi_c) \cap \Gamma$, we clearly have $\partial\mathcal{M} = \partial\mathcal{M}_u \sqcup \partial\mathcal{M}_w$ and $\mathcal{M}(\pi) \cap \Gamma = \Gamma_u \sqcup \Gamma_w$. Furthermore, $\mathcal{M}_u$ and $\mathcal{M}_w$ are in states $a = \tau(\Gamma_u)$ and $b = \tau(\Gamma_w)$ respectively, and by property (A2) we must have $(a, b) \in \mathfrak{X}(s)$, and so we have $D_v^s \geq D_u^a + D_w^b$. Likewise, if we are given a T_u^+-cut and T_w^+-cut with terminals Γ_u and Γ_w in their respective top-components, and $(\tau(\Gamma_u), \tau(\Gamma_w)) \in \mathfrak{X}(s)$, then by property (A3) $\Gamma_u \cup \Gamma_w$ is independent and $\tau(\Gamma_u \cup \Gamma_w) = s$, and so merging the two cuts yields a T_v^+-cut in state s. This completes the proof of Theorem 4(a).

Algorithm for ConTreeMCut$_{\max}$. The dynamic programming algorithm claimed in Theorem 4(b) is a straighforward combination of the dynamic program formulation above with techniques in Section 4. The key change is that now we need to store the minimum boundary weight of the top component of T_v^+-cuts in state s, subject to the other components having boundary weight at most L. More details can be found in the proof of a more general statement in in Section 6.

6 Generalized Type Graphs and Split Complexity

In this section, we extend the algorithms from Sections 4 and 5 using "generalized" type graphs, then prove Theorem 5 and Corollary 1.1.

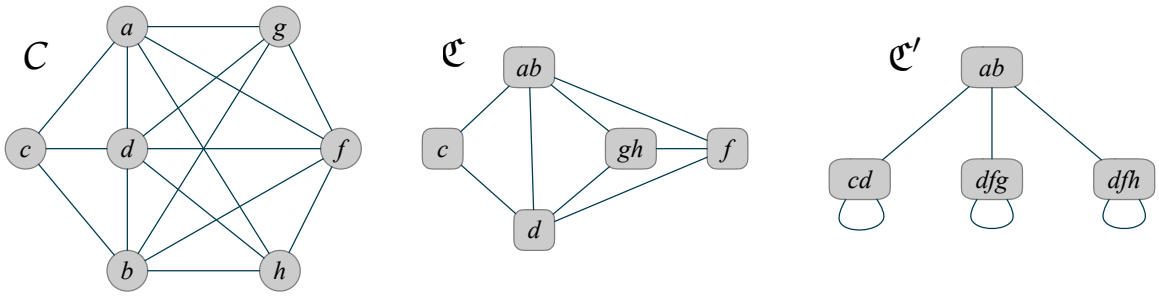


Figure 5: An example of a conflict graph $\mathcal{C}$, its (standard) type graph $\mathfrak{C}$, and one generalized type graph $\mathfrak{C}'$ for $\mathcal{C}$. The types are labeled by the vertices of $\mathcal{C}$ mapped to them. In $\mathfrak{C}'$, vertices may be mapped to multiple types. The example shows that in a generalized type graph the number of types can be smaller.

Basic ideas. We start by briefly explaining the concept of generalized type graphs and split complexity, and outlining the proof of Theorem 5. We are assuming the reader is familiar with the terminology from Section 5. Informally, a "generalized type graph" $\mathfrak{C} = (\mathfrak{N}, \mathfrak{F}, \tau)$ modifies the notion of type graphs from neighborhood diversity, allowing a terminal t to be assigned *multiple* types $\tau(t) \subseteq \mathfrak{N}$ rather than one. A conflict $tt' \in \mathcal{F}$ is represented by the existence of an edge in $\mathfrak{F}$ between some pair of types from $\tau(t)$ and $\tau(t')$. This extension can exponentially reduce the number of dynamic programming states: in particular we only need 2^q states in this model for q-clique cover. Aside from changing the semantics of types, the definition of splits is unchanged, and in fact $\mathrm{sp}(\mathfrak{C})$ was already defined in Section 5 for the standard type graph and is unchanged for our general formulation. Crucially, the properties (A1)-(A3) identified in that section still hold in this model, and the algorithms for Theorem 5 are almost identical to their neighborhood diversity variants.

In general, conflict graphs admitting a type graph $\mathfrak{C}$ with low split complexity $\mathrm{sp}(\mathfrak{C})$ must have bounded neighborhood diversity $\mathrm{nd}(\mathcal{C})$, but the former can be exponentially smaller than the $4^{\mathrm{nd}(\mathcal{C})}$ factor in Theorem 4. The downside is that the following algorithms require specifying $\mathfrak{C}$ on input; because this model extends graphs with q-clique covers, which take $O^*(2^{2^q})$ time to detect, it is unlikely that efficiently computing a minimal generalized type graph is possible.

Generalized Type Graph. For a conflict graph $\mathcal{C}$, a *generalized type graph* (or simply a type graph) $\mathfrak{C} = (\mathfrak{N}, \mathfrak{F}, \tau)$ is an undirected graph (with self-loops allowed) on *types* $\mathfrak{N}$ connected by edges $\mathfrak{F}$, together with a map $\tau \colon \Gamma \to \mathcal{I}(\mathfrak{C}) - \{\emptyset\}$, where $\mathcal{I}(\mathfrak{C})$ denotes the family of independent sets in $\mathfrak{C}$ (allowing self-loops). In order to be a valid type graph for $\mathcal{C}$, $\mathfrak{C}$ must satisfy the following condition: $tt' \in \mathcal{F}$ if and only if there exists some edge $\eta\eta' \in \mathfrak{F}$ with $\eta \in \tau(t)$ and $\eta' \in \tau(t')$. (If $\eta = \eta'$ then $\eta\eta'$ is a self-loop.) As in Section 5, the types $\mathfrak{N}^{(\mathrm{c})} \subseteq \mathfrak{N}$ with self-loops are called c-types and the remaining types $\mathfrak{N}^{(\mathrm{i})} = \mathfrak{N} - \mathfrak{N}^{(\mathrm{c})}$ are called i-types. For simplicity we also impose $|\mathfrak{N}| \leq |\Gamma|$.[7]

If τ maps terminals to singletons, this is equivalent to the standard type graph used to define neighborhood diversity. The crucial difference is that terminals can now be assigned *multiple* types. Intuitively, the conflict graphs induced by a generalized type graph with k types can be constructed in the following way: select k (possibly overlapping) subsets $\Gamma_1, ..., \Gamma_k$ of Γ, make some Γ_i cliques, and add a biclique between any *disjoint* pair Γ_i, Γ_j. See Figure 5 for an example.

We make a few remarks before moving on. For one, conflict graphs may be represented by distinct generalized type graphs, even if they use a minimal number of types. Second, the terminals $\{t \in \Gamma \mid \eta \in \tau(t)\}$ associated with an i-type η may not form an independent set, as pairs of some

[7] If we were only interested in minimizing the number of types, then trivially we can make this assumption without loss of generality. It's not a priori clear that we can make this assumption for minimizing splits, but we'll put this aside.

terminals may be assigned the same c-type or two types adjacent in $\mathfrak{C}$. Third, we observe that $\tau(t) = \tau(t')$ implies that t and t' are twins, which yields the inequality $\mathrm{nd}(\mathcal{C}) \leq 2^{|\mathfrak{N}|}$; in particular, a type graph with a constant number of types induces a conflict graph with constant neighborhood diversity.

States and splits. We now discuss states and splits for a conflict graph $\mathcal{C}$ induced by some generalized type graph $\mathfrak{C}$. Letting $\tau(\Gamma') = \bigcup_{t\in\Gamma'} \tau(t) \subseteq \mathfrak{N}$ for $\Gamma' \subseteq \Gamma$, we observe that, just like in Section 5, τ maps independent sets in $\mathcal{C}$ to independent sets in $\mathfrak{C}$ (perhaps with self-loops). Like before, *states* $s \in \mathcal{I}(\mathfrak{C})$ refer to these independent sets, including the empty set.

The definition of splits is the same as in Section 5. For $s \in \mathcal{I}(\mathfrak{C})$ we define the splits $\mathfrak{X}(s) = \left\{(a,b) \in \mathcal{I}(\mathfrak{C})^2 \mid a \cup b = s \text{ and } a^{\mathrm{c}} \cap b^{\mathrm{c}} = \emptyset\right\}$, where s^{c} refers to c-types which s contains.

We illustrate the definitions with two examples. If $\mathcal{C}_1$ is a union of q cliques, then it is induced by a type graph $\mathfrak{C}_1 = (\mathfrak{N}_1, \mathfrak{F}_1, \tau_1)$, where $\mathfrak{N}_1 = [q]$, $\mathfrak{F}_1$ is the set of all self-loops, and $\tau_1(t)$ indicates which cliques t belongs to. We have $\mathcal{I}(\mathfrak{C}_1) = \mathscr{P}([q])$ (every subset of types is independent) and $\mathfrak{X}_1(s) = \{(a, s-a) \mid a \subseteq s\}$ (every bipartition is a split of s).

Letting $\mathcal{C}_2$ denote a complete c-partite graph (that is not a clique), the standard type graph $\mathfrak{C}_2 = (\mathfrak{N}_2, \mathfrak{F}_2, \tau_2)$ using neighborhood diversity is a complete graph on c types (say $\mathfrak{N}_2 = [c]$) with no self-loops, where $\left\{\tau_2^{-1}(\eta)\right\}_{\eta\in[c]}$ is simply the c different parts. The states $\mathcal{I}(\mathfrak{C}_2) = \{\emptyset\} \cup \{\{\eta\} \mid \eta \in [c]\}$ are exactly the empty set and singleton sets, and the splits of $\{\eta\}$ are $\mathfrak{X}_2(\{\eta\}) = \{(\emptyset, \{\eta\}), (\{\eta\}, \emptyset), (\{\eta\}, \{\eta\})\}$; the last split corresponds to dividing a non-empty subset of a part into two non-empty subsets. (If $\mathcal{C}_2$ is a clique, then the standard type graph is a single node with a self-loop.)

By inspection, the splits and the map τ still satisfy properties (A1)-(A3) from Section 5.

The runtime will be dominated by the total number of splits, denoted by $\mathrm{sp}(\mathfrak{C}) = \sum_{s\in\mathcal{I}(\mathfrak{C})} |\mathfrak{X}(s)|$ like in Section 5. The same reasoning as in Section 5 implies $\mathrm{sp}(\mathfrak{C}) \leq |\mathcal{I}(\mathfrak{C})|^2 \leq 4^{|\mathfrak{N}|}$. For the two examples discussed earlier, $|\mathcal{I}(\mathfrak{C}_1)| = 2^q$ and $\mathrm{sp}(\mathfrak{C}_1) = \sum_{X\subseteq[q]} 2^{|X|} = 3^q$, while $|\mathcal{I}(\mathfrak{C}_2)| = c+1$ and $\mathrm{sp}(\mathfrak{C}_2) = 3c+1$.

Algorithm for TreeMCut$_\Sigma$. We now prove Theorem 5 for TreeMCut$_\Sigma$. Using the type graphs $\mathfrak{C}_1$ and $\mathfrak{C}_2$ from earlier, Corollary 1.1 follows for the min-sum case.

Given $\mathfrak{C} = (\mathfrak{N}, \mathfrak{F}, \tau)$, we can enumerate $\mathcal{I}(\mathfrak{C})$ in time $O(|\mathcal{I}(\mathfrak{C})||\mathfrak{N}|) = O(|\mathcal{I}(\mathfrak{C})|n)$ [28]. For each state $s \in \mathcal{I}(\mathfrak{C})$ we may compute $\mathfrak{X}(s)$ in $O(|\mathfrak{X}(s)|)$ time by enumerating all partitions of s^{c} and all pairs of subsets of s^{i}.

We now just need to modify Algorithm 1 as follows: $\mathcal{S}$ is now $\mathcal{I}(\mathfrak{C})$, every instance of $\{t\}$ in Line 4 is replaced by $\tau(t)$, and the min-sum convolution formula in Line 8 is replaced by $\min_{(a,b)\in\mathfrak{X}(s)} \left\{D_u^a + D_w^b\right\}$. We now just need to reinterpret the meaning of D_v^s and D_v^*. We say a T_v^+-cut $\mathcal{M}$ is in state $s \in \mathcal{S}$ if the top component satisfies $\tau(\mathcal{M}(\pi_v) \cap \Gamma) = s$. Then D_v^s is the minimum sum-cost over all T_v^+-cuts in state s, and by property (A1) a T_v^+-cut is in state $\emptyset$ if and only if it is improper, so D_v^* is still the minimum sum-cost over all proper T_v^+-cuts.

The correctness of D_t^s for leaves $t \in \Gamma$ in Line 4 is easy to verify. The formula in Line 10 has not changed, so we only need to justify the new formula for Line 8. Let $\mathcal{M}$ be a T_v^+-cut in state $s \neq \emptyset$ with sum-cost D_v^s, and for $c \in ch(v) = \{u, w\}$, let $\mathcal{M}_c$ be the induced T_c^+-cut by restricting $\mathcal{M}$ to T_c^+. Since $\mathcal{M}$ is proper, $v\pi_v \notin \partial\mathcal{M}$ and $\mathcal{M}_c(\pi_c) = \mathcal{M}_(\pi_v) \cap T_c^+$. Letting $\Gamma_c = \mathcal{M}_c(\pi_c) \cap \Gamma$, we clearly have $\partial\mathcal{M} = \partial\mathcal{M}_u \sqcup \partial\mathcal{M}_w$ and $\mathcal{M}(\pi_v) \cap \Gamma = \Gamma_u \sqcup \Gamma_w$. Furthermore, $\mathcal{M}_u$ and $\mathcal{M}_w$ are in states $a = \tau(\Gamma_u)$ and $b = \tau(\Gamma_w)$ respectively, and by property (A2) we must have $(a,b) \in \mathfrak{X}(s)$, and so we have $D_v^s \geq D_u^a + D_w^b$. Likewise, if we were given a T_u^+-cut and T_w^+-cut with terminals Γ_u and Γ_w in their respective top-components, and $(\tau(\Gamma_u), \tau(\Gamma_w)) \in \mathfrak{X}(s)$, then by property (A3) $\Gamma_u \cup \Gamma_w$ is independent and $\tau(\Gamma_u \cup \Gamma_w) = s$, and so merging the two cuts yields a T_v^+-cut in state s.

Algorithm for ConTreeMCut$_{\max}$. We now prove Theorem 5 for ConTreeMCut$_{\max}$. Using the type graphs $\mathfrak{C}_1$ and $\mathfrak{C}_2$, Corollary 1.1 for ConTreeMCut$_{\max}$ follows. We assume that our tree T is in binary normal form with root r, and we add a dummy parent π_r for r with $\omega(r\pi_r) = 0$. We use the same notion of T_v^+-cuts and their boundary convention as in the previous sections.

Let $\mathfrak{C} = (\mathfrak{N}, \mathfrak{F}, \tau)$ be a generalized type graph for $\mathcal{C}$ given on input and let $\mathcal{S} = \mathcal{I}(\mathfrak{C})$. As explained before, in $O(\mathrm{sp}(\mathfrak{C})n)$ time we can enumerate both $\mathcal{S}$ and the corresponding splits.

Algorithm For Integer Weights. We now describe the main dynamic sub-routine, which in time $O(n \sum_{s\in\mathcal{S}} |\mathfrak{X}(s)|)$ determines whether $\mathrm{opt}_{\max}(T, \mathcal{C}) \le L$ for a given real $L \in \mathbb{R}^+$. This automatically implies Theorem 5(b) for integer weights. For now we treat L as fixed.

As in Section 5, we say a T_v^+-cut $\mathcal{M}$ is in state $s \in \mathcal{S}$ if the top component satisfies $\tau(\mathcal{M}(\pi_v) \cap \Gamma) = s$. Furthermore, $\mathcal{M}$ is L-feasible if all non-top boundaries have weight at most L. For each $s \in \mathcal{S}$ and $v \in V$, we define the table entry $D_v^s(L)$ to be the minimum, over all L-feasible T_v^+-cuts $\mathcal{M}$ in state s, of the boundary weight $\omega(\partial\mathcal{M}(\pi_v))$ of its top component in T_v^+. (When no such cut exists, set $D_v^s(L) = \infty$.) We also create a table entry $D_v^*(L) = \min_{s\neq\emptyset} D_v^s(L)$, denoting the minimum top-boundary weight over all L-feasible proper T_v^+-cuts. (By Property (A1), $\mathcal{M}$ is in state $\emptyset$ if and only if it is improper.) From the definitions, $\mathrm{opt}_{\max}(T, \mathcal{C}) \le L$ if and only if $D_r^*(L) \le L$.

For a leaf $t \in \Gamma$, we set $D_t^{\tau(t)}(L) = 0$, set $D_t^{\emptyset}(L) = \omega(t\pi_t)$ if $\omega(t\pi_t) \le L$ and ∞ otherwise, and set $D_t^s(L) = \infty$ for $s \notin \{\tau(t), \emptyset\}$. We now describe the recurrence for an internal vertex $v \notin \Gamma$ with children u, w.

$$D_v^s(L) = \min_{(a,b)\in\mathfrak{X}(s)} \left\{ D_u^a(L) + D_w^b(L) \right\} \qquad \text{for } s \neq \emptyset \tag{5}$$

$$D_v^*(L) = \min_{s\neq\emptyset} D_v^s(L) \tag{6}$$

$$D_v^{\emptyset}(L) = \begin{cases} \min\left\{ D_u^{\emptyset}(L) + D_w^{\emptyset}(L), \omega(v\pi_v) \right\} & \text{if } \omega(v\pi_v) + D_v^*(L) \le L \\ D_u^{\emptyset}(L) + D_w^{\emptyset}(L) & \text{else} \end{cases} \tag{7}$$

The runtime is dominated by the evaluation of Eq. (5), which takes $O(\sum_{s\in\mathcal{S}} |\mathfrak{X}(s)|)$ time per vertex.

The formula for the leaves holds from the definitions and inspection. For an internal vertex $v \notin \Gamma$, observe that $D_v^{\emptyset}(L)$ and $D_v^*(L)$ have the same semantics as the table entries $D_v^0(L)$ and $D_v^1(L)$ from Section 4; in particular $D_v^{\emptyset}$ optimizes over improper cuts, while D_v^* optimizes over proper cuts. Indeed, Eq. (7) matches Eq. (4), so it suffices to show that Eq. (5) is correct.

That Eq. (5) holds follows from a similar argument as in Section 4. Recall that we have a correspondence between T_v^+-cuts $\mathcal{Q}_v$ (whose top component contains v, which is necessary if $\mathcal{Q}_v$ is proper) and pairs of a T_u^+-cut $\mathcal{Q}_u$ and T_w^+-cut $\mathcal{Q}_w$. Specifically, given $\mathcal{Q}_u$ and $\mathcal{Q}_w$ which are L-feasible, we can merge them to obtain $\mathcal{Q}_v$ which is L-feasible and satisfies $\mathcal{Q}_v(\pi_v) = \{\pi_v\} \cup \mathcal{Q}_u(\pi_u) \cup \mathcal{Q}_w(\pi_w)$. (We do not know a priori that $\mathcal{Q}_v(\pi_v)$ doesn't contain a conflict, but we can still interpret $\mathcal{Q}_v$ as a partition.) This implies $\partial\mathcal{Q}_v(\pi_v) = \partial\mathcal{Q}_u(\pi_u) \sqcup \partial\mathcal{Q}_w(\pi_w)$. Likewise, given an L-feasible $\mathcal{Q}_v$ (whose top component contains v, which is necessary if $\mathcal{Q}_v$ is proper), we can obtain an L-feasible pair $\mathcal{Q}_u$ and $\mathcal{Q}_w$ satisfying the same relations.

Importantly, in both directions the three cuts satisfy the relation $\Gamma_v = \Gamma_u \sqcup \Gamma_w$, where $\Gamma_x = \mathcal{Q}_x(\pi_x) \cap \Gamma$ denotes the terminals in the respective top components. This implies that if we started with $\mathcal{Q}_v$ and $\tau(\Gamma_v) = s \neq \emptyset$ ($\mathcal{Q}_v$ is in state s), then $(\tau(\Gamma_u), \tau(\Gamma_w)) \in \mathfrak{X}(s)$ is a split of s by Property (A2). Conversely, if we started with $\mathcal{Q}_u$ and $\mathcal{Q}_w$ and $(\tau(\Gamma_u), \tau(\Gamma_w)) \in \mathfrak{X}(s)$, then, by Property (A3), Γ_v is guaranteed to be independent and satisfy $\tau(\Gamma_v) = s$, thus $\mathcal{Q}_v$ as the result of merging is in fact a T_v^+-cut in state s. This correspondence then implies the correctness of Eq. (5), as desired.

Extension to Real Weights. As in Section 4, for each $v \in V$ we can define the function H_v and real number L_v, where $H_v(L)$ equals $\omega(v\pi_v)$ if $\omega(v\pi_v) + D_v^*(L) \leq L$ and is ∞ otherwise, and $L_v \coloneqq \min\{L : \omega(v\pi_v) + D_v^*(L) \leq L\}$. Since $D_v^*(L)$ has the same semantics as the table entry $D_v^1(L)$ from Section 4, H_v and L_v have the same interpretation as before. In particular, H_v is a right-continuous step function with at most one point of discontinuity at L_v. And by induction, D_v^s (for $s \neq \emptyset$) and D_v^*, as functions of L, are non-increasing right-continuous functions whose discontinuities are a subset of $\mathcal{B}_v^1 = \{L_u\}_{u \in T_v - \{v\}}$, while $D_v^\emptyset$ has the same form with discontinuities in $\mathcal{B}_v^0 = \mathcal{B}_v^1 \cup \{L_v\}$.

This implies that we can compute a list representation for each dynamic programming table entry, using $O(n)$ space per function and $O(n)$ time for computing component-wise sums and minimums. In the same way as in Section 4, the discontinuities L_v are computed dynamically from the list representation of D_v^*. This completes the proof of Theorem 5(b) for real weights.

Information on the use of generative AI. The ideas of using the local cuts and the linear program (2) in Section 3, and of working with the entire functions D_v^σ in Section 4, were suggested by the LLM-based service Bolzano. We thank Vašek Rozhoň for providing us with access to the service. In addition, GPT-6 suggested the half-integrality proof in Appendix E.

References

[1] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. SeffiNaor, and R. Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.

[2] C. Bentz and P. Le Bodic. Complexity of the multicut problem, in its vanilla, partial and generalized versions, in graphs of bounded treewidth. *Theoretical Computer Science*, 809:239–249, 2020.

[3] D. Bertsimas, C.-P. Teo, and R. Vohra. Analysis of LP relaxations for multiway and multicut problems. *Networks*, 34(2):102–114, 1999.

[4] N. Bousquet, J. Daligault, S. Thomassé, and A. Yeo. A polynomial kernel for multicut in trees. In *Proc. of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *LIPIcs*, pages 183–194. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009.

[5] L. Brankovic and H. Fernau. A novel parameterised approximation algorithm for minimum vertex cover. *Theoretical Computer Science*, 511:85–108, 2013. Exact and Parameterized Computation.

[6] J. Chen, J. Fan, I. A. Kanj, Y. Liu, and F. Zhang. Multicut in trees viewed through the eyes of vertex cover. In *Proc. of Algorithms and Data Structures - 12th International Symposium (WADS)*, volume 6844 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2011.

[7] B. Cherkasskij. Solution of a problem on multicommodity flows in a network. *Ehkon. Mat. Metody*, 13:143–151, 1977.

[8] S. Chopra and M. R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991.

[9] K. Clinch, S. Gaspers, T. Z. He, S. Mackenzie, and T. Zhang. A faster randomized algorithm for vertex cover: An automated approach, 2025.

[10] M.-C. Costa and A. Billionnet. Multiway cut and integer flow problems in trees. *Electronic Notes in Discrete Mathematics*, 17:105–109, 2004. Workshop on Graphs and Combinatorial Optimization.

[11] M.-C. Costa, L. Létocart, and F. Roupin. A greedy algorithm for multicut and integral multiflow in rooted trees. *Operations Research Letters*, 31(1):21–27, 2003.

[12] M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005. Logistics: From Theory to Application.

[13] M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.

[14] G. Călinescu, C. G. Fernandes, and B. Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *Journal of Algorithms*, 48(2):333–359, 2003.

[15] G. Călinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60(3):564–574, 2000.

[16] 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.

[17] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

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

[19] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Exp. Algorithmics*, 13, Feb. 2009.

[20] J. Guo and R. Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Netw.*, 46(3):124–135, Oct. 2005.

[21] D. G. Harris and N. S. Narayanaswamy. A Faster Algorithm for Vertex Cover Parameterized by Solution Size. In *Proc. of 41st International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

[22] W. Hasan and R. Motwani. Coloring away communication in parallel query optimization. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, page 239–250, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[23] C. Hou, S. Gao, W. Liu, W. Wu, D.-Z. Du, and B. Hou. An approximation algorithm for the submodular multicut problem in trees with linear penalties. *Optimization Letters*, 15(4):1105–1112, Jun 2021.

[24] X. Hou, W. Liu, and B. Hou. An approximation algorithm for the k-prize-collecting multicut on a tree problem. *Theoretical Computer Science*, 844:26–33, 2020.

[25] I. Kanj, G. Lin, T. Liu, W. Tong, G. Xia, J. Xu, B. Yang, F. Zhang, P. Zhang, and B. Zhu. Improved parameterized and exact algorithms for cut problems on trees. *Theoretical Computer Science*, 607:455–470, 2015. Combinatorial Optimization and Applications.

[26] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research*, 29(3):436–461, 2004.

[27] P. Kolman and H. R. Tiwary. Min-max connected multiway cut, 2026.

[28] K. Kurita, K. Wasa, T. Uno, and H. Arimura. A constant amortized time enumeration algorithm for independent sets in graphs with bounded clique number. *Theoretical Computer Science*, 874:32–41, 2021.

[29] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, Sep 2012.

[30] A. Levin and D. Segev. Partial multicuts in trees. *Theoretical Computer Science*, 369(1):384–395, 2006.

[31] H. Liu and P. Zhang. On the generalized multiway cut in trees problem. In G. Lin, editor, *Combinatorial Optimization and Applications*, pages 151–162, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] X. Liu and W. Li. A primal dual approximation algorithm for the multicut problem in trees with submodular penalties. In *Algorithmic Aspects in Information and Management*, pages 203–211, Cham, 2019. Springer International Publishing.

[33] X. Liu and W. Li. An approximation algorithm for the b-prize-collecting multicut problem in trees. In *Theory and Applications of Models of Computation*, pages 262–271, Cham, 2022. Springer International Publishing.

[34] X. Liu and W. Li. An approximation algorithm for the $\boldsymbol{K}$-prize-collecting multicut problem in trees with submodular penalties. *Mathematical Structures in Computer Science*, 34(3):193–210, 2024.

[35] J. Mestre. Lagrangian Relaxation and Partial Cover. In *Proc. of 25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 539–550. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008.

[36] G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, Dec 1974.

[37] Z. Svitkina and É. Tardos. Min-max multiway cut. In *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX, and 8th International Workshop on Randomization and Computation, RANDOM 2004*, volume 3122 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2004.

[38] M. Yannakakis, P. C. Kanellakis, S. S. Cosmadakis, and C. H. Papadimitriou. Cutting and partitioning a graph after a fixed pattern. In *Automata, Languages and Programming*, pages 712–722, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

A Faster 2-Approximation for Unweighted Trees

In Section 3 we gave a polynomial-time 2-approximation algorithm for the unweighted version of ConTreeMCut$_{\max}$. That algorithm was rather inefficient. In this section we show an alternative and much faster algorithm.

Theorem 6. *There is a 2-approximation algorithm for unweighted* ConTreeMCut$_{\max}$ with running time $O(|\mathcal{F}| + n \log n)$.

From now on, we fix an instance of ConTreeMCut$_{\max}$ consisting of a normal-form tree $T = (V, E, \omega)$ (where $\omega(e) = 1$ for all $e \in E$) with root r and a conflict graph $\mathcal{C} = (\Gamma, \mathcal{F})$. To prove Theorem 6, it is sufficient to prove the lemma below, as the algorithm from this lemma can be combined with binary search for k to compute the 2-approximate solution.

Lemma A.1. *Let $(T, \mathcal{C})$ be an unweighted* ConTreeMCut$_{\max}$ instance. After $O(n + |\mathcal{F}|)$ pre-processing, there is an $O(n)$ algorithm that, given $k \in \mathbb{N}$, either returns a multicut of cost at most $2k$ or determines that $\text{opt}_{\max}(T, \mathcal{C}) > k$.

The rest of this section consists of the proof of Lemma A.1. For $v \in V$, we define a *T_v-cut* to be a connected multicut $\mathcal{Q}$ for the restricted instance $(T_v, \mathcal{C} \cap T_v)$, with the possible exception that the component containing v (called the "top component" of $\mathcal{Q}$) may not contain a terminal. $\mathcal{Q}$ is called *proper* if this component has a terminal and *improper* otherwise. $\mathcal{Q}$ is called *k-feasible* if all non-top components have cost at most k. Furthermore, the *local cost* of $\mathcal{Q}$ is the top component boundary weight (within T_v). Let λ_v° and $\lambda_v^\bullet$ denote the optimum local cost over all k-feasible improper and proper T_v-cuts respectively. We follow the convention that if X is a component of a T_v-cut, ∂X always refers to the boundary edges in T_v, excluding the edge $v\pi_v$. (This is similar to T_v^+-cuts from Section 4, with T_v^+ replaced by T_v.)

The essence of the algorithm is to extend Observation 3.3 to a lower bound on λ_v° and $\lambda_v^\bullet$. Using this, we reduce the problem to solving a set of vertex cover instances, one for each sub-tree T_v, then perform a dynamic-programming bottom-up computation on T. For each vertex v, we will define an auxiliary graph H_v whose nodes are some children of v, and we use a 2-approximate vertex cover for H_v to compute the top component of a multicut for T_v. Crucially, these vertex cover instances do not depend on k, and can thus be pre-processed, and their combined size does not exceed $n + |\mathcal{F}|$.

A.1 Preprocessing

Here we describe how to construct the vertex cover instances needed for our algorithm and how to solve them all in $O(n + |\mathcal{F}|)$ time total.

For an internal vertex $v \in V$, let $A_v \subseteq ch(v)$ be the set of children $u \in ch(v)$ of v such that the terminals in $T_u \cup \{v\}$ do not conflict. An *auxiliary conflict graph* H_v is $H_v = (A_v, F_v)$ where, for vertices $u, u' \in A_v$, $uu' \in F_v$ if and only if there exist terminals $t \in V(T_u)$ and $t' \in V(T_{u'})$ such that $tt' \in \mathcal{F}$.

For vertices $u, v \in V$, let $\text{LCA}(u, v)$ denote their least common ancestor in T (which may be u or v). For a vertex v and descendant $u \in T_v - \{v\}$, let $child_u(v)$ be the unique child $w \in ch(v)$ such that $u \in T_w$. Trivially, $\sum_{v \in V} |A_v| \leq n$. Furthermore, we claim that $\sum_{v \in V} |F_v| \leq |\mathcal{F}|$. This inequality follows by mapping each $tt' \in \mathcal{F}$ with $\text{LCA}(t, t') \notin \{t, t'\}$ to $uu' \in F_v$, where $v = \text{LCA}(t, t')$, $u = child_t(v)$ and $u' = child_{t'}(v)$.

The pseudo-code below shows how to compute every H_v in $O(n + |\mathcal{F}|)$ time. Correctness follows directly from the definition of H_v and the correspondence described above. Since $\text{LCA}(u, v)$ and $child_u(v)$ can be computed in constant time with $O(n)$ pre-processing using standard data structures, the computation of all graphs H_v will take time $O(n + |\mathcal{F}|)$.

Algorithm: Computing graphs H_v and vertex covers S_v
  for each $v \in V$ **do** $a_v \leftarrow$ false, $A_v \leftarrow \emptyset$, $F_v \leftarrow \emptyset$
  for each $tt' \in \mathcal{F}$ **do**
    $v \leftarrow \mathrm{LCA}(t, t')$ and $a_v \leftarrow$ true
    if $t = v$ **then** $a_{u'} \leftarrow$ true for $u' = \mathit{child}_{t'}(v)$
    if $t' = v$ **then** $a_u \leftarrow$ true for $u = \mathit{child}_t(v)$
  for each $v \in V$ (processing bottom-up) **do**
    $a_v \leftarrow a_v \vee \bigvee_{u \in ch(v)} a_u$
    if $v \neq r$ and $a_v =$ false **then** add v to A_{π_v}
  for each $tt' \in \mathcal{F}$ **do**
    $v \leftarrow \mathrm{LCA}(t, t')$
    if $v \notin \{t, t'\}$ **then**
      $u \leftarrow \mathit{child}_t(v)$ and $u' \leftarrow \mathit{child}_{t'}(v)$
      if $u, u' \in A_v$ **then** add uu' to F_v
  for each internal $v \in V$ **do**
    $H_v \leftarrow (A_v, F_v)$
    if $F_v \neq \emptyset$ **then**
      $S_v \leftarrow$ vertex cover of H_v with $|S_v| \leq \min\{2 \cdot \mathrm{vc}(H_v), |A_v|\} - 1$
    else $S_v = \emptyset$

To compute S_v in the last step, we sligtly modify the standard matching-based 2-approximation for VertexCover: First choose any edge $(t', t'') \in F_v$. (We know that $|F_v| \neq \emptyset$.) Let H' and H'' be the graphs obtained from H_v by removing t' and t'', respectively. Compute 2-approximate solutions S'_v and S''_v for VertexCover on H'_v and H''_v. If $|S'_v| \leq |S''_v|$ then let $S_v = S'_v \cup \{t'\}$, else let $S_v = S''_v \cup \{t''\}$. Since one of t', t'' must be in the optimal vertex cover of H_v, we obtain that $|S_v| \leq 2 \cdot \mathrm{vc}(H_v) - 1$. If it so happens that $S_v = A_v$ (and $A_v \neq \emptyset$), we can simply remove any vertex from S_v so that we also have $|S_v| \leq |A_v| - 1$. The running time for computing each S_v is $O(|A_v| + |F_v|)$, so the total cost of computing all sets S_v amortizes to $O(n + |\mathcal{F}|)$.

A.2 Main Algorithm

Let $k \in \mathbb{N}$ be the given integer. If $k = 0$, the algorithm checks if $\mathcal{F} = \emptyset$, and if that is the case, it returns the optimal multicut consisting of just one component $\{T\}$, and otherwise it reports that $\mathrm{opt}_{\max}(T, \mathcal{C}) > 0$.

So we assume now that $k \geq 1$. Our algorithm proceeds bottom-up, computing table entries $D^\bullet_v$ and D°_v, for all $v \in V$, that contain a proper T_v-cut and an improper T_v-cut, respectively. We will also keep track of the respective top components that we denote by $X^\bullet_v$ and X°_v. Similar to Sections 4 and 5, $\partial X^\bullet_v$, ∂X°_v, $\partial D^\bullet_v$, and ∂D°_v are by convention restricted to T_v. So $|\partial X^\bullet_v|$ and $|\partial X^\circ_v|$ are the local costs of $D^\bullet_v$ and D°_v. (It's possible that the solution for v will be undefined, which we represent by NULL. By default, the corresponding local cost is then ∞.)

Computing solutions for the leaves is trivial: if t is a leaf (and thus $t \in \Gamma$), let $D^\bullet_t = \{X^\bullet_t\}$ for $X^\bullet_t = \{t\}$, and $D^\circ_t = X^\circ_t =$ NULL. So assume now that v is an internal vertex such that all vertices in $T_v \setminus \{v\}$ have already been processed. Let $B_v = ch(v) \setminus A_v$. We partition B_v into two subsets $B^\bullet_v$ and B°_v:

$$\begin{aligned} B^\bullet_v &= \{u \in B_v : |\partial X^\bullet_u| \leq 2k - 1\} \\ B^\circ_v &= B_v \setminus B^\bullet_v \end{aligned}$$

(Note that this partitioning depends on the algorithm.)

Computing $D_v^\bullet$ and $X_v^\bullet$: We distinguish three cases.

(P1) If $v \in \Gamma$ or $A_v \neq \emptyset$: Set

$$\begin{aligned} X_v^\bullet &= \{v\} \cup \bigcup_{u \in A_v \setminus S_v} V(T_u) \cup \bigcup_{u \in B_v^\circ} X_u^\circ \\ D_v^\bullet &= \{X_v^\bullet\} \cup \{V(T_u) : u \in S_v\} \cup \bigcup_{u \in B_v^\bullet} D_u^\bullet \cup \bigcup_{u \in B_v^\circ} (D_u^\circ \setminus \{X_u^\circ\}) \end{aligned}$$

(P2) Otherwise, if $B_v^\bullet \neq \emptyset$: Select $u^* \in B_v^\bullet$ for which $|\partial X_{u^*}^\bullet|$ is minimized, and set

$$\begin{aligned} X_v^\bullet &= \{v\} \cup X_{u^*}^\bullet \cup \bigcup_{u \in B_v^\circ} X_u^\circ \\ D_v^\bullet &= \{X_v^\bullet\} \cup (D_{u^*}^\bullet \setminus \{X_{u^*}^\bullet\}) \cup \bigcup_{u \in B_v^\bullet \setminus \{u^*\}} D_u^\bullet \cup \bigcup_{u \in B_v^\circ} (D_u^\circ \setminus \{X_u^\circ\}) \end{aligned}$$

(P3) Otherwise (that is, for $B_v^\circ = ch(v)$ and $v \notin \Gamma$): set $X_v^\bullet, D_v^\bullet = \text{NULL}$.

Computing D_v° and X_v°: We have two cases.

(I1) If $v \notin \Gamma$, set

$$\begin{aligned} X_v^\circ &= \{v\} \cup \bigcup_{u \in B_v^\circ} X_u^\circ \\ D_v^\circ &= \{X_v^\circ\} \cup \{V(T_u) : u \in A_v\} \cup \bigcup_{u \in B_v^\bullet} D_u^\bullet \cup \bigcup_{u \in B_v^\circ} (D_u^\circ \setminus \{X_u^\circ\}) \end{aligned}$$

(I2) Otherwise (for $v \in \Gamma$), set $X_v^\circ, D_v^\circ = \text{NULL}$

In the event that NULL appears in the formulas above, we set $Y \cup \text{NULL} = \text{NULL}$ for any set Y. With this caveat, we are done describing our recurrence equations. We observe that (when not NULL) the T_v-cuts $D_v^\bullet$ obtained via these recurrences are proper. In Case (P1), the top component of $D_v^\bullet$ either contains v as a terminal or a terminal leaf from A_v. In Case (P2), it contains a terminal from the top component of $D_{u^*}^\bullet$. As for D_v°, in Case (I1) it does not contain a terminal, because v is not a terminal and top components X_u° for $u \in B_v^\circ$ also do not contain terminals.

After processing all vertices, the algorithm returns $D_r^\bullet$ if $|\partial X_r^\bullet| \leq 2k-1$ and otherwise declares $\text{opt}_{\max}(T, \mathcal{C}) > k$. This completes the description of the algorithm.

Regarding the running time, for a linear-time implementation, we only need to keep track of the edges in $X_v^\bullet, X_v^\circ$ which are incident to v. Then by standard backtracking we can obtain the final multicut.

Analysis. To complete the proof of Lemma A.1, it remains to prove its correctness. We first prove a conditional bound on local costs for each T_v-cut in Lemma A.2, then in Lemma A.7 we show that bounding the local costs of our computed cuts yields a bound on the min-max cost.

Throughout the rest of the proof, for a real number a, we use notation $\langle a \rangle^+ = \max(a, 0)$.

Lemma A.2. *For each vertex v:*

(a) If $\lambda_v^\bullet \le k$, then $|\partial X_v^\bullet| \le \langle 2\lambda_v^\bullet - 1\rangle^+$.

(b) Likewise, $\lambda_v^\circ \le k$ implies $|\partial X_v^\circ| \le \langle 2\lambda_v^\circ - 1\rangle^+$.

By the normal form of T, each leaf is a terminal, which implies that $\lambda_v^\circ \ge 1$ for each node v (λ_v° could be ∞), so in (b) we can write $2\lambda_v^\circ - 1$ instead of $\langle 2\lambda_v^\circ - 1\rangle^+$ on the right-hand side of the equation.

Proof. We proceed by induction on the tree, assuming *both* implications in our induction hypothesis. The lemma is trivial when v is a leaf, so assume v is an internal vertex and the lemma holds for all $u \in ch(v)$ (for both solution types).

First we make one general observation about the children of v in B_v°.

Claim A.3. *If $\mathcal{Q}'$ is a k-feasible T_v-cut (either proper or improper) with local cost at most k, then for each $u \in B_v^\circ$, v's component in $\mathcal{Q}'$ contains u but it does not contain any terminal from T_u.*

Proof: Let $X' \in \mathcal{Q}'$ be the component containing v. If X' did not contain u then, by k-feasibility of $\mathcal{Q}'$, $\mathcal{Q}'$ restricted to T_u would be a proper k-feasible T_u-cut with local cost at most k. (In particular, the top component in this T_u-cut would be a non-top component in $\mathcal{Q}'$, so it would contain a terminal.) Similarly, if X' contained u and some terminal from T_u then again, $\mathcal{Q}'$ restricted to T_u would be a proper k-feasible T_u-cut with local cost at most k. (In this case, the top component is $X' \cap T_u$ and contains a terminal.) Either way, we have $\lambda_u^\bullet \le k$, and by our induction hypothesis, this implies $|\partial X_u^\bullet| \le \langle 2\lambda_u^\bullet - 1\rangle^+ \le 2k - 1$, contradicting the definition of B_v°. ◁

As a corollary, our assumption that $\min\{\lambda_v^\circ, \lambda_v^\bullet\} \le k$ implies $\lambda_u^\circ \le \min\{\lambda_v^\circ, \lambda_v^\bullet\} \le k$ (and $|\partial X_u^\circ| \le 2\lambda_u^\circ - 1$ by induction) for all $u \in B_v^\circ$.

Proof of (a): computing $D_v^\bullet$. We first analyze $D_v^\bullet$, breaking into cases corresponding to the cases in its computation. For the next few claims, it is convenient to introduce the following notation. Given a component $X \in \mathcal{Q}$ of a T_v-cut and $u \in ch(v)$, let $\partial_u X = \partial X \cap (E(T_u) \cup \{uv\})$. Then let $\partial_A X = \bigsqcup_{u \in A_v} \partial_u X$, $\partial_B^\bullet X = \bigsqcup_{u \in B_v^\bullet} \partial_u X$, and $\partial_B^\circ X = \bigsqcup_{u \in B_v^\circ} \partial_u X$. Clearly we have $\partial X = \partial_A X \sqcup \partial_B^\bullet X \sqcup \partial_B^\circ X$.

Claim A.4. *If $\lambda_v^\bullet \le k$ then*

$$\lambda_v^\bullet \;\ge\; \mathrm{vc}(H_v) + |B_v^\bullet| + \sum_{u \in B_v^\circ} \lambda_u^\circ$$

Proof: Consider a proper k-feasible T_v-cut $\mathcal{Q}'$ with local cost $\lambda_v^\bullet \le k$, and let $X' \in \mathcal{Q}'$ be the top component. We produce three bounds on $\partial_A X'$, $\partial_B^\bullet X'$, and $\partial_B^\circ X'$. The set $S' = \{u \in A_v \mid |\partial_u X'| \ge 1\}$ is a vertex cover for H_v, implying $|\partial_A X'| \ge \mathrm{vc}(H_v)$. (If edge $uw \in F_v$ was not covered by S', then $\partial_u X' = \partial_w X' = \emptyset$ and X contains both T_u and T_w. But $uw \in F_v$ implies there is some conflicting terminal pair $tt' \in \mathcal{F}$ where $t \in T_u$ and $t' \in T_w$, implying X' has a conflict.) If $u \in B_v^\bullet$, then $|\partial_u X'| \ge 1$, otherwise X' would contain $T_u \cup \{v\}$ and a conflicting pair. Therefore $|\partial_B^\bullet X'| \ge |B_v^\bullet|$. If $u \in B_v^\circ$, then by Claim A.3, $\mathcal{Q}'$ induces an improper k-feasible T_u-cut with local cost $|\partial_u X'|$, implying $|\partial_u X'| \ge \lambda_u^\circ$ and $|\partial_B^\circ X'| \ge \sum_{u \in B_v^\circ} \lambda_u^\circ$. Summing inequalities yields our desired bound. ◁

Suppose now that v satisfies Case (P1) of the algorithm, that is $v \in \Gamma$ or $A_v \ne \emptyset$. In this case,

$$|\partial X_v^\bullet| \;=\; |S_v| + |B_v^\bullet| + \sum_{u \in B_v^\circ} |\partial X_u^\circ|$$

If $F_v \ne \emptyset$ then $|S_v| \le 2\,\mathrm{vc}(H_v) - 1$. If $B_v^\bullet \ne \emptyset$ then $|B_v^\bullet| \le 2|B_v^\bullet| - 1$. If $B_v^\circ \ne \emptyset$ then for all $u \in B_v^\circ$ we have $|\partial X_u^\circ| \le 2\lambda_u^\circ - 1$, by induction. Therefore, if at least one of these conditions holds, Claim A.4

implies that

$$|\partial X_v^\bullet| \;\le\; 2\,\mathrm{vc}(H_v) + 2|B_v^\bullet| + 2\sum_{u\in B_v^\circ}\lambda_u^\circ - 1 \;\le\; 2\lambda_v^\bullet - 1$$

On the other hand, if all three conditions are false then $\partial X_v^\bullet = \emptyset$. Thus $\partial X_v^\bullet \le \langle 2\lambda_v^\bullet - 1\rangle^+$ in Case (P1), as needed.

The next claim handles the case when $v \notin \Gamma$ and $A_v = \emptyset$.

Claim A.5. *If $\lambda_v^\bullet \le k$, $v \notin \Gamma$ and $A_v = \emptyset$, then $B_v^\bullet \neq \emptyset$ and*

$$\lambda_v^\bullet \ge |B_v^\bullet| - 1 + \min_{u\in B_v^\bullet}\lambda_u^\bullet + \sum_{u\in B_v^\circ}\lambda_u^\circ$$

Proof: Consider a proper k-feasible T_v-cut $\mathcal{Q}'$ with local cost $\lambda_v^\bullet \le k$, and let $X' \in \mathcal{Q}'$ be the top component. By similar reasoning as before, Claim A.3 implies $|\partial_u X'| \ge \lambda_u^\circ$ for $u \in B_v^\circ$, so $|\partial_B^\circ X'| \ge \sum_{u\in B_v^\circ}\lambda_u^\circ$. Combined with $v \notin \Gamma$ and $A_v = \emptyset$, Claim A.3 also implies that X' contains a terminal from T_w for some $w \in B_v^\bullet$. (In particular, $B_v^\bullet \neq \emptyset$.) Then $\mathcal{Q}'$ induces a proper k-feasible T_w-cut of local cost $|\partial_w X'|$, implying $|\partial_w X'| \ge \lambda_w^\bullet \ge \min_{u\in B_v^\bullet}\lambda_u^\bullet$. We have $|\partial_u X'| \ge 1$ for the remaining children $u \in B_v^\bullet - \{w\}$, which implies our desired bound. ◁

Suppose now that v satisfies Case (P2) of the algorithm, that is $v \notin \Gamma$, $A_v = \emptyset$, and $B_v^\bullet \neq \emptyset$. Claim A.5 implies there is some $w \in B_v^\bullet$ where $\lambda_w^\bullet = \min_{u\in B_v^\bullet}\lambda_u^\bullet \le \lambda_v^\bullet \le k$. By the algorithm, we have $|\partial X_{u^*}^\bullet| \le |\partial X_w^\bullet|$, so

$$|\partial X_v^\bullet| \;\le\; |\partial X_w^\bullet| + |B_v^\bullet| - 1 + \sum_{u\in B_v^\circ}|\partial X_u^\circ|$$

Now, if $\lambda_w^\bullet \ge 1$ then, by induction, $|\partial X_w^\bullet| \le 2\,\lambda_w^\bullet - 1$. If $|B_v^\bullet| \ge 2$ then $|B_v^\bullet| - 1 \le 2(|B_v^\bullet| - 1) - 1$. If $B_v^\circ \neq \emptyset$ then for all $u \in B_v^\circ$ we have $|\partial X_u^\circ| \le 2\lambda_u^\circ - 1$, by induction. So if at least one of these three conditions holds then

$$|\partial X_v^\bullet| \;\le\; 2\lambda_w^\bullet + 2(|B_v^\bullet| - 1) + 2\sum_{u\in B_v^\circ}\lambda_u^\circ - 1 \;\le\; 2\lambda_v^\bullet - 1$$

On the other hand, if all three conditions are false then $\partial X_v^\bullet = \emptyset$, So $|\partial X_v^\bullet| \le \langle 2\lambda_v^\bullet - 1\rangle^+$ holds in Case (P2).

Regarding Case (P3), note that Claim A.5 implies that $A_v \cup B_v^\bullet$ is non-empty, which means that (under the assumption that $\lambda_v^\bullet \le k$) the algorithm will never execute Case (P3) for computing the proper T_v-cut where $B_v^\circ = ch(v)$. This concludes the analysis of $D_v^\bullet$.

Proof of (b): computing D_v°. We now analyze D_v° and derive one more bound.

Claim A.6. *If $\lambda_v^\circ \le k$ then*

$$\lambda_v^\circ \ge |A_v| + |B_v^\bullet| + \sum_{u\in B_v^\circ}\lambda_u^\circ$$

Proof: Let $\mathcal{Q}'$ be an improper k-feasible T_v-cut with local cost $\lambda_v^\circ \le k$, and let $X' \in \mathcal{Q}'$ be the top component. By similar reasoning as before, Claim A.3 implies $|\partial_B^\circ X'| \ge \sum_{u\in B_v^\circ}\lambda_u^\circ$. For $u \in A_v \cup B_v^\bullet$, $|\partial_u X'| \ge 1$, for otherwise X' would contain T_u and its terminals. (Recall that T is in normal form, so all leaves are terminals.) ◁

For computing D_v° the algorithm has two cases. In Case (I2), $v \in \Gamma$ implies that $\lambda_v^\circ = \infty$, and the inductive step holds. So consider Case (I1), that is $v \notin \Gamma$. Then

$$|\partial X_v^\circ| \;=\; |A_v| + |B_v^\bullet| + \sum_{u\in B_v^\circ}|\partial X_u^\circ|$$

If $A_v \cup B_v^\bullet \neq \emptyset$ then $|A_v| + |B_v^\bullet| \leq 2(|A_v| + |B_v^\bullet|) - 1$. If $B_v^\circ \neq \emptyset$ then, as in the previous cases, for $u \in B_v^\circ$ we have $\lambda_u^\circ \geq 1$, so the induction implies $|\partial X_u^\circ| \leq 2\lambda_u^\circ - 1$. One of these cases must hold (since v is not a leaf), so

$$|\partial X_v^\circ| \;\leq\; 2|A_v| + 2|B_v^\bullet| + 2\sum_{u \in B_v^\circ} \lambda_u^\circ - 1 \;\leq\; 2\lambda_v^\circ - 1$$

This finishes the proof of Lemma A.2. □

Lemma A.7. *For each vertex v:*

(a) If $|\partial X_v^\bullet| \leq 2k-1$ then $D_v^\bullet$ has max-cost (in T_v) at most $2k$.

(b) Likewise, if $|\partial X_v^\circ| \leq 2k-1$ then D_v° has max-cost (in T_v) at most $2k$.

Proof. We proceed by induction on T, again assuming both implications in the induction hypothesis. The lemma is trivial for leaves, so suppose v is an internal vertex. Whether $|\partial X_v^\bullet|$ or $|\partial X_v^\circ|$ is at most $2k-1$, we have the following bounds on the children's local costs: $|\partial X_u^\bullet| \leq 2k-1$ for $u \in B_v^\bullet$ by definition and $\sum_{u \in B_v^\circ} |\partial X_u^\circ| \leq \min\{|\partial X_v^\bullet|, |\partial X_v^\circ|\}$ by construction. This means our induction hypothesis applies to all children $u \in B_v$. Specifically, for $u \in B_v^\bullet$ and $u' \in B_v^\circ$, $D_u^\bullet$ and $D_{u'}^\circ$ have each max-cost at most $2k$ in their respective sub-trees.

We now prove part (a), by estimating the max-cost of $D_v^\bullet$ when $|\partial X_v^\bullet| \leq 2k-1$. (The analysis for part (b) is nearly identical, so we omit it.) Let X be a component of $D_v^\bullet$, so ∂X denotes X's boundary in T_v. If X is the top component (i.e. $v \in X$), then $|\partial X| \leq 2k-1$, by our assumption.

So we can now assume that X is a non-top compoment of $D_v^\bullet$. If ∂X contains an edge uv where $u \in A_v$, then $X = T_u$ and $|\partial X| = 1$. If instead $uv \in \partial X$ and $u \in B_v^\bullet$, then X is the top component of $D_u^\bullet$ and $|\partial X| = |\partial X_u^\bullet| + 1 \leq 2k$. Lastly, if X does not satisfy any of the previous conditions, then X must be a non-top component of a previously computed T_u-cut $\mathcal{Q}$, for some $u \in B_v$. Either $u \in B_v^\bullet$ and $\mathcal{Q} = D_u^\bullet$, or $u \in B_v^\circ$ and $\mathcal{Q} = D_u^\circ$; either way, by induction we have $|\partial X| \leq 2k$. □

If $\mathrm{opt}_{\max}(T, \mathcal{C}) \leq k$, then $\lambda_r^\bullet \leq \mathrm{opt}_{\max}(T, \mathcal{C}) \leq k$, which by Lemma A.2 implies $|\partial X_r^\bullet| \leq 2k-1$. Then by Lemma A.7, $D_r^\bullet$ is a multicut of $(T, \mathcal{C})$ with max-cost at most $2k$, as desired.

B Normal Forms

In Section 2 we defined a normal form and a binary normal form for trees and stated Lemma 2.1 that says that each tree can be converted into an appropriate normal form. In this section we prove this lemma.

We first describe how to modify a non-weighted tree T into normal form, which can be done by repeatedly applying two reduction rules: (1) If v is a non-terminal leaf, remove it; (2) If v is a non-terminal with exactly two neighbors u and w (and v is not the root), add an edge between u and w and delete v. These steps will eventually convert T into a tree T' in normal form. To justify correctness, consider just one step, converting T into a tree T'', and show that it satisfies all necessary properties except possibly (a). For instance, in rule (2), a multicut in T that contains uv or wv (it cannot contain both) has the same cost as the multicut T'' that contains uw instead.

The above construction also works for the weighted case: in rule (2) simply let $\omega''(uw) = \min\{\omega(uv), \omega(vw)\}$. Thus to show part (b), we can assume that T is in the normal form already. Let $W = \sum_{e \in E} \omega(e)$. To obtain T' we then repeatedly apply the following rules:

(3) If $t \in \Gamma$ is an internal vertex with children $u_1, ..., u_d$, temporarily remove t and create a new vertex v in its place. That is, v is now the parent of each u_i with the same weight $\omega'(vu_i) = \omega(tu_i)$. Then reattach t to v as a leaf and set $\omega'(tv) = W$.

(4) If $v \notin \Gamma$ is an internal vertex with children $u_1, ..., u_d$ for $d \geq 3$, create a new vertex w and make it the new parent of $u_2, ..., u_d$ with the same weights $\omega'(wu_i) = \omega(vu_i)$. Then make w a child of v and set $\omega'(wv) = W$.

These steps will eventually transform T into binary normal form. To justify correctness, let T'' be the tree obtained after one step of this process from T, and argue that it has the required properties, except possibly (b). The idea, in both cases (3) and (4), is that W is so large that any multicut of T'' that cuts tv can be modified to cut other edges without increasing cost, so again we have an obvious correspondence between multicuts of T and T''. Both rules preserve the normal form conditions, so the resulting tree will eventually be in binary normal form.

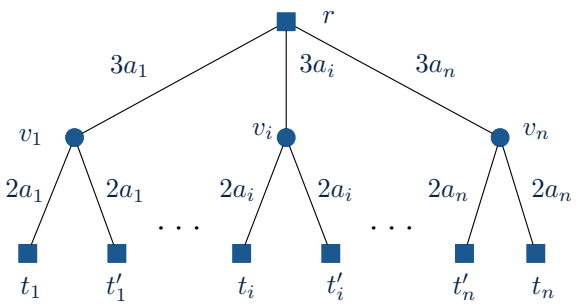


Figure 6: The constructed instance of TreeCutting. Squares represent terminals, circles represent non-terminals.

C $\mathbb{NP}$-Hardness of Svitkina/Tardos' TreeCutting Problem [37]

Here we consider the TreeCutting problem of Svitkina and Tardos [37] which is essentially a bi-objective combination of ConTreeMwCut$_\Sigma$ and ConTreeMwCut$_{\max}$: Given a weighted tree $T = (V, E, \omega)$, a set of terminals $\Gamma \subseteq V$, and a bound B, the objective is to compute a partition of T into subtrees $T_1, ..., T_h$, each with at most one terminal and with $\omega(\partial T_i) \leq B$, that minimizes total boundary weight $\frac{1}{2}\sum_{i=1}^{h} \omega(\partial T_i)$. Svitkina and Tardos [37] gave a pseudo-polynomial algorithm for TreeCutting that they used as a subroutine to design an approximation algorithm for the TreeMwCut$_{\max}$ problem for weighted trees (without the connectivity requirement). They did not analyze, however, the computational complexity of TreeCutting itself. Here, we show that the problem is $\mathbb{NP}$-hard, and thus a pseudo-polynomial time algorithm is all we can hope for if we insist on finding the exact optimum.

The proof is by reduction from Partition. Consider an instance of Partition with integers $a_1, ..., a_n$ and let $A = \frac{1}{2}\sum_{i=1}^{n} a_i$. We convert it into the following instance of TreeCutting: the tree has a root vertex r, and for each $i = 1, ..., n$ it has three vertices v_i, t_i and t'_i, and edges (r, v_i) with weight $3a_i$, and (v_i, t_i), (v_i, t'_i), both with weight $2a_i$. (See Fig. 6.) The terminals are $\Gamma = \{r, t_1, t'_1, ..., t_n, t'_n\}$ and the boundary bound is $B = 7A$. We claim that

($*$) The instance of Partition has a solution iff the constructed instance of TreeCutting has a solution of cost at most $9A$.

The proof of ($*$) is quite simple, so we only sketch the idea, leaving the details to the reader. In the instance of TreeCutting, for each i, in order to separate terminals r, t_i and t'_i each partition must cut at least two edges among (r, v_i), (v_i, t_i), and (v_i, t'_i). Cutting all three unnecessarily increases the total cost, so we can assume that only two are cut. By symmetry, we can then assume that (v_i, t_i) is cut. So we have two types of cuts: Type 1 with (v_i, t_i) and (r, v_i) cut, and Type 2 with (v_i, t_i) and (v_i, t'_i) cut, at costs $5a_i$ and $4a_i$, respectively. Have the cut of Type 1 represent i being in the solution I of the instance of Partition, and Type 2 of being in the complement $\bar{I}$. All clusters except the top one (containing r) have small cost, so only the top cluster is of concern. Then to satisfy the condition on TreeCutting in ($*$) we need:

$$\sum_{i\in I} 3a_i + \sum_{i\in \bar{I}} 4a_i \;\leq\; 7A \quad \text{and} \quad \sum_{i\in I} 5a_i + \sum_{i\in \bar{I}} 4a_i \;\leq\; 9A$$

By simple algebra, these inequalities are equivalent to $\sum_{i\in I} a_i = A$, so the equivalence ($*$) is true.

D On the Connectivity Requirement for MwCut$_{\max}$

In this section we examine how the connectivity requirement for MCut$_{\max}$ affects the value of the optimum solution. We focus on the simple case in which the conflict graph is a clique, that is on MwCut$_{\max}$. (Recall that in MwCut$_{\max}$ the conflict graph is a clique, so all terminals must be separated.) We refer to the worst-case ratio between the optimal solution for MwCut$_{\max}$ and the optimal connected solution (namely the solution for ConMwCut$_{\max}$) as the *connectivity gap*.

Trees. We start with the case of weighted trees, that is the solution of TreeMwCut$_{\max}$ versus that of ConTreeMwCut$_{\max}$. Svitkina and Tardos [37] gave an example of an instance showing that for trees the connectivity gap for TreeMwCut$_{\max}$ is at least 1.25. We show here that this gap is $\Theta(\log n)$.

For the lower bound, we use a non-weighted and perfectly balanced binary tree T and the set Γ of terminals that are the leaves of T. Consider a partition of T's vertices into pairs, where each pair is formed by grouping each non-terminal with a unique terminal (one leaf will not be paired with any internal vertex). The min-max cost of this partition is $O(1)$. On the other hand, in a connected partition, the part containing the root would also contain the path P from the root to the unique terminal t of this part. Then any cut that separates P from $\Gamma \setminus \{t\}$ must have $\Omega(\log n)$ edges. (We can assume that P be the leftmost path from the root. Then for each vertex $v \in P \setminus \{t\}$, the rightmost path from v to a leaf must contain at least one edge in this cut.) Thus the boundary weight of the partition containing the root is $\Omega(\log n)$, and we can conclude that the connectivity gap, even for non-weighted trees, is $\Omega(\log n)$.

We next argue that connectivity gap for weighted trees is $O(\log n)$. Consider an instance consisting of a tree $T = (V, E, \omega)$ and a set of terminals Γ. Let $\mathcal{M}$ be an optimal multicut of TreeMwCut$_{\max}$ for this instance, and let $L = \text{cost}_{\max}(\mathcal{M})$. (Here $\mathcal{M}$ is represented as a partition into connected components, each without conflicts. For brevity, here we will refer to these connected components simply as *components*, even though some may not contain a terminal.) We will convert $\mathcal{M}$ into a connected multicut $\mathcal{M}'$ satisfying $\text{cost}_{\max}(\mathcal{M}') = O(L \log |\Gamma|)$, thus establishing the claimed upper bound on the connectivity gap.

Each component of $\mathcal{M}$ contains at most one terminal. We refer to the pieces with terminals as *terminal components* and other as *nonterminal components*. To obtain $\mathcal{M}'$, we combine the components into larger subtrees, each containing exactly one terminal. To this end, consider a graph Q whose nodes are the components of $\mathcal{M}$ and edges connect the pairs of components that have an edge (clearly, exactly one) of T between them. Then Q is itself a tree, which we root at an arbitrary node r.

First, repeatedly merge any nonterminal leaf into its parent. This operation does not increase the boundary weight of the parent component. Consequently, every remaining component has boundary weight at most L, and every leaf of Q is a terminal component. Next, ensure that all nonterminal components of degree two are non-adjacent. To do this, consider any inclusion-wise maximal path of degree two nonterminal components and merge them together into one subtree. The boundary weight of this new subtree is at most $2L$. After repeating this process, each nonterminal component in Q of degree two is only adjacent to terminal components.

For each nonterminal component C, choose a nearest terminal node among its descendants, and retain only the edge from C to the child on the path to that terminal (i.e., delete every other edge to C and from C). Remove from Q also all edges connecting two terminal components. This way, by the properties of Q described in the previous paragraph, the tree Q is partitioned into paths of length at most $\log_2 |\Gamma|$, each ending at a terminal component. Now merge the components of each such path, producing a subtree of T that contains exactly one terminal. The boundary of this

subtree will have weight $O(L \log |\Gamma|)$, completing the proof.

Arbitrary graphs. We now show that for arbitrary graphs, the connectivity gap for $\text{MWCUT}_{\max}$ could be much larger than for trees, namely at least $\Omega(\sqrt{n})$, even in the non-weighted case. Our example is a non-weighted graph $G = (V, E)$ with terminals $\Gamma \subseteq V$. Let $V' = V \setminus \Gamma$ be the set of non-terminals. Γ is partitioned into two sets, $\Gamma = \Gamma' \cup \Gamma''$, where $|\Gamma'| = k'$ and $|V'| = |\Gamma''| = \ell$, so that $n = k' + 2\ell$. The topology of G is this: the vertices in $V' \cup \Gamma'$ form a clique of size $k' + \ell$, and the vertices in Γ'' are isolated.

Let $\mathcal{M}$ be a partition representing the following multicut: each terminal in Γ' is a singleton, and all other vertices are grouped into ℓ disjoint pairs, with one vertex from V' and one from Γ''. Then $\text{cost}_{\max}(\mathcal{M}) = \ell + k' - 1 \leq n$.

To show our lower bound, it is sufficient to prove that for some choice of k' and ℓ each connected multicut has cost $\Omega(n^{1.5})$. In any connected multicut, the components of the terminals in Γ'' are singletons, and $V' \cup \Gamma'$ is partitioned into components, each containing one terminal from Γ'. Let B be a component with maximum cardinality. Observe that $|\partial B| = |B|(\ell + k' - |B|)$. Choosing $k' = \sqrt{n}$ and $\ell = \frac{1}{2}(n - \sqrt{n})$, with n sufficiently large, we have $|B| = \Omega(\sqrt{n})$, which implies that the boundary weight of B is $|\partial B| = \Omega(n^{1.5})$. (To see this, consider the cases when $|B| \leq n/4$ and $|B| > n/4$.) Thus the cost of this multicut is $\Omega(n^{1.5})$, completing the proof of our $\Omega(\sqrt{n})$ lower bound.

E Primal-Dual Algorithm for LCutTrees°

In this section we present a primal-dual algorithm for LCutTrees° from Section 3, providing an alternative proof of Lemma 3.2. Let $(T, \mathcal{C}, v^*)$ be our instance. The dual of the linear program in the proof of Lemma 3.2 is as follows:

$$\text{maximize} \sum_{st \in \mathcal{F}} \alpha_{st} \tag{8}$$

$$\text{subject to} \sum_{st \in \mathcal{F}} \alpha_{st}(1_{e \in P_{sv^*}} + 1_{e \in P_{tv^*}}) \leq \omega(e) \quad \forall e \in E \tag{9}$$

$$\alpha_{st} \geq 0 \quad \forall st \in \mathcal{F}$$

Naturally, we can interpret the dual as a flow problem: for each conflict edge $st \in \mathcal{F}$, we send *two* flows from v^*, one to s along P_{sv^*} and another to t along P_{tv^*}. We send α_{st} units of flow along both paths. (The intersection $P_{tv^*} \cap P_{sv^*}$ thus receives $2\alpha_{st}$ units of flow.) The right hand side of inequality (9) is then the total flow going through edge e.

We greedily construct a dual solution, similar to the primal-dual algorithms for VertexCover and TreeMCut$_\Sigma$. The variables α_{st} are initially zero, and we say a tree edge $e \in E$ is *saturated* when its corresponding constraint is tight. Then for each conflict edge $st \in \mathcal{F}$, we simply raise the value of α_{st} until a tree edge (possibly many) in $P_{sv^*} \cup P_{tv^*}$ become saturated. Let $J \subseteq E$ be the set of saturated tree edges after applying this process to all conflict edges. Our local cut X is then the connected component in $T - J$ containing v^*.

By rooting T at v^*, we can obtain the tree edges $P_{sv^*} \cup P_{tv^*}$ for any $st \in \mathcal{F}$ in $O(n)$ time, and so this algorithm can be implemented in $O(n|\mathcal{F}|)$ time. Furthermore, X is indeed a local cut: for each conflict edge $st \in \mathcal{F}$, ∂X must contain at least one saturated edge from $P_{sv^*} \cup P_{tv^*}$ by construction, implying that at least one of s, t is not in X.

We now show that $\omega(\partial X) \leq 2 \sum_{st \in \mathcal{F}} \alpha_{st}$, where α_{st} refers to the constructed dual variables. The boundary $\partial X \subseteq J$ contains only (but not necessarily all) saturated edges, and since X is connected and contains v^*, we have $|\partial X \cap P_{tv^*}| \leq 1$ for all $t \in \Gamma$. This implies

$$\begin{aligned}\omega(\partial X) &= \sum_{e \in \partial X} \omega(e) = \sum_{e \in \partial X} \sum_{st \in \mathcal{F}} \alpha_{st}(1_{e \in P_{sv^*}} + 1_{e \in P_{tv^*}}) \\ &= \sum_{st \in \mathcal{F}} \alpha_{st} \sum_{e \in \partial X} (1_{e \in P_{sv^*}} + 1_{e \in P_{tv^*}}) \leq 2 \sum_{st \in \mathcal{F}} \alpha_{st}\end{aligned}$$

as desired. Then, by duality, X is a 2-approximate solution.

Half-Integrality. We remark that the linear program from Section 3.1 is also half-integral. In fact, the distances $x(P_{tv^*})$ for each $t \in \Gamma$ lie in $\{0, 1/2, 1\}$ at the polytope's extremal points. To see this, we root T at v^* and observe that our LP is equivalent to the following program:

$$\text{minimize} \sum_{v \in V - \{v^*\}} \omega(v\pi_v)(x_v - x_{\pi_v}) \tag{10}$$

$$\text{subject to } x_v \geq x_{\pi_v} \quad \forall v \in V - \{v^*\} \tag{11}$$

$$x_s + x_t \geq 1 \quad \forall st \in \mathcal{F} \tag{12}$$

$$x_{v^*} = 0, x_v \geq 0 \quad \forall v \in V - \{v^*\}$$

where our variables are $\{x_v\}_{v \in V - \{v^*\}}$ (with $x_{v^*} = 0$ as a dummy variable). Intuitively, we've simply done a change of variables, where now we assign distances to vertices that induces edge weights

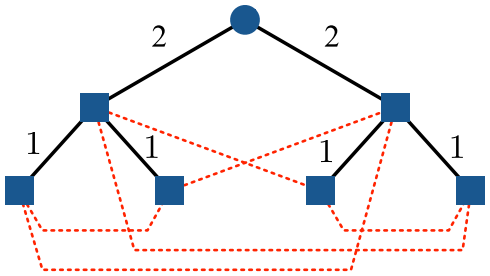


Figure 7: An instance of the "non-split LP" with non-half-integral extremal points; v^* is the single non-terminal at the top, and conflicts are illustrated in dashed red lines. One can use an LP solver to verify that the unique optimal solution of cost $8/3$ is achieved by setting $x_e = 1/3$ for all edges e. By inspection, the best half-integral and integral solutions have cost 3.

$x(v\pi_v) = x_v - x_{\pi_v}$, using the observation that weights of a tree can be reconstructed from distances from a fixed source. To ensure that the induced weights are non-negative, we impose an ordering constraint in Eq. 11. Clearly, extremal points of our original LP correspond to extremal points of this new one.

Ignoring the slightly more complicated objective function, it's now clear that our new LP constraints are just a modified version of the VertexCover LP. Indeed, standard arguments proving the half-integrality of the VertexCover LP (e.g. [36]) also apply here; the extra ordering constraints make no difference.

Interestingly, we can slightly modify the linear program from Section 3.1 by not double-counting edges in $P_{sv^*} \cap P_{tv^*}$, which simply means replacing Eq. 2 with $\sum_{e \in P_{sv^*} \cup P_{tv^*}} x_e \geq 1$. This "non-split LP" is still a relaxation of LCutTrees$^\circ$, and the same LP rounding and primal-dual algorithms apply to this variant. However, this variant is *not* half-integral. An example is shown in Figure 7, and indeed it's not to hard to construct for every integer $d \geq 3$ an example containing an extremal point with denominator d.