Approximating Connected Maximum Cuts via Local Search

Baruch Schieber
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA

Soroush Vahidi
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA

Abstract

The Connected Max Cut (CMC) problem takes in an undirected graph $G(V, E)$ and finds a subset $S \subseteq V$ such that the induced subgraph $G[S]$ is connected and the number of edges connecting vertices in S to vertices in $V \setminus S$ is maximized. This problem is closely related to the Max Leaf Degree (MLD) problem. The input to the MLD problem is an undirected graph $G(V, E)$ and the goal is to find a subtree of G that maximizes the degree (in G) of its leaves. [Gandhi et al. 2018] observed that an α-approximation for the MLD problem induces an $\mathcal{O}(\alpha)$-approximation for the CMC problem.

We present an $\mathcal{O}(\log \log |V|)$-approximation algorithm for the MLD problem via local search. This implies an $\mathcal{O}(\log \log |V|)$-approximation algorithm for the CMC problem. Thus, improving (exponentially) the best known $\mathcal{O}(\log |V|)$ approximation of the Connected Max Cut problem [Hajiaghayi et al. 2015].

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Routing and network design problems; Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation algorithms, graph theory, max-cut, local search

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.93

1 Introduction

The Connected Max Cut (CMC) problem takes in an undirected graph $G(V, E)$ and finds a subset $S \subseteq V$ such that the induced subgraph $G[S]$ is connected and the number of edges connecting vertices in S to vertices in $V \setminus S$ is maximized. This problem is known to be NP-Hard [6]. We give an $\mathcal{O}(\log \log n)$-approximation algorithm for this problem, where $n = |V|$. This improves on the previously known $\mathcal{O}(\log n)$-approximation algorithm given in [6]. (We say that an approximation algorithm for a maximization problem achieves an $\alpha \geq 1$ approximation ratio if the value of its output is at least $\frac{1}{\alpha}$ of the maximum.)

A function $f\colon 2^V \to \mathbb{R}_+$ is *submodular* if for every $A, B \in V$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. It is not difficult to verify that the cut function is submodular and non-monotone, and thus the CMC problem is a special case of non-monotone submodular function maximization subject to connectivity constraints. To the best of our knowledge there are no nontrivial approximation algorithms for the well motivated general non-monotone submodular function maximization subject to connectivity constraints.

The CMC problem is closely related to the Max Leaf Tree (MLT) and Max Leaf Spanning Tree (MLST) problems. The undirected version of the MLT problem takes in a vertex-weighted undirected graph $G(V, E, w)$ and finds a subtree with maximum weight on the leaves. The weighted version of the MLST problem requires the output tree of the MLT instance to be a spanning tree. We note that the MLT problem can also be posed as a non-monotone submodular function maximization problem subject to connectivity constraints.

In this case the submodular function $f: 2^V \to \mathbb{R}_+$ defined on subsets of vertices of G is $f(U) = \sum_{x \in \mathcal{N}(U)} w(x)$, where $\mathcal{N}(U)$ is the set of neighbors of the vertices in U that are not in U (and $f(\emptyset) = 0$).

Gandhi et al. [4] observed that an α-approximation algorithm for the MLT problem implies a 4α-approximation algorithm for the (edge) weighted CMC problem. Based on this observation we consider a special case of the MLT problem we call the MLD problem. In this problem we are given an undirected graph $G(V, E)$ and the goal is to find a subtree that maximizes the degree (in G) of its leaves, that is, the weight of vertices in this special case of the MLT problem is their degree in G. Applying the observation in Gandhi et al. [4] we get that an α-approximation algorithm for the MLD problem gives a 4α-approximation algorithm for the *unweighted* CMC problem. We present an $\mathcal{O}(\log \log n)$-approximation algorithm for the MLD problem and thus an approximation algorithm with the same quality for the CMC problem.

Our algorithm is quite simple and involves successive local improvement steps. The proof that our algorithm indeed achieves an $\mathcal{O}(\log \log n)$ approximation ratio is somewhat involved and relies on the fact that the weights of the vertices in the MLD problem instance are the degrees of these vertices in an undirected graph with no parallel edges. To prove the $\mathcal{O}(\log \log n)$ approximation ratio we prove a stronger property of our solution: we prove that the total number of edges in the graph is $\mathcal{O}(\log \log n)$ times the sum of the degrees of the leaves of the computed subtree. This bound is tight as we can construct an instance that achieves this upper bound.

1.1 Motivation and Prior Art

We motivate both our CMC problem and the general non-monotone submodular function maximization subject to connectivity constraints. The Max Cut problem is a fundamental combinatorial optimization problem and is one of the problems listed in Karp's seminal paper on NP-Complete problems [9]. Extending the Max Cut problem to the CMC problem is natural. We note that a similar extension of the Min Cut problem is trivial since we can always find a minimum cut that separates a connected component.

The Max Cut problem has numerous applications, e.g., in circuit layout design, statistical physics [1], and image segmentation [3, 13]. Some of these applications can be extended to the connected max cut problem. For example, Hajiaghayi et al. [6] noted that the image segmentation application with the additional requirement that the pixels are connected can be modeled as a CMC problem.

There are quite a few applications of non-monotone submodular function maximization among them: maximum facility location and optimal sensor placement, segmentation problems [10], least core value of general supermodular cost games [15], optimal marketing over social networks [7], and revenue maximization for banner advertisement [2]. Some of these applications consider functions that are defined on a vertex set of a graph and thus it is natural to consider the maximization subject to graph constraints. For example, in sensor placement we may constrain the activated sensors to be connected to enable efficient communication, or sets targeted for marketing in a social network may be required to be in the same social community.

The CMC problem was introduced by Hajiaghayi et al. in [5] (see also [6]) who gave an $\mathcal{O}(\log n)$-approximation algorithm for this problem and obtained a polynomial time approximation scheme for the CMC problem on planar graphs and on bounded genus graphs.

Lee et al. [11] generalized the problem considered in [6] in two aspects: (i) they considered more general graph constraints and not just connectivity constraints, and (ii) they allowed the graph constraints to be on a separate graph from the one in which the max cut needs

to be computed (as long as both graphs are defined on the same set of vertices). The graph constraints considered in [11] include independent set, vertex cover, dominating set and connectivity. Lee et al. [11] showed a 2 approximation for this problem on graphs with bounded treewidth. Some of these results (for constraints such as independent set, dominating set, and connectivity) can be achieved also for planar graphs, bounded genus graphs, and graphs with a fixed excluded minor. The algorithm in [11] is not combinatorial but uses an LP relaxation based on the Sherali-Adams hierarchy.

As mentioned above, Gandhi et al. [4] showed that the weighted CMC problem and the MLT problem are closely related. They obtained a constant approximation algorithm for MLT problem in the special case of $\{0,1\}$ weights, and then used a "bucketing" technique to extend it to an $\mathcal{O}(\log n)$-approximation algorithm for the general case. This implies an $\mathcal{O}(\log n)$-approximation algorithm for the weighted CMC problem which improves the $\mathcal{O}\big(\log^2 n\big)$-approximation algorithm for this problem given in [6].

The unweighted MLT problem is simply the problem of finding a subtree with the maximum number of leaves. Note that there is always a spanning tree that maximizes the number of leaves, and thus the unweighted MLT problem and the unweighted MLST problem are equivalent and both can be approximated within a factor of 2 [16]. The weighted MLST problem admits no $n^{1-\varepsilon}$ approximation, for any constant $\varepsilon > 0$ as it is equivalent to the Maximum Independent Set problem, as observed in [8]. This is in contrast to the weighted MLT problem that admits an $\mathcal{O}(\log n)$ approximation [4], and to the MLD problem, which is a special case of the weighted MLT problem in which the weight of every vertex is equal to its degree, that admits an $\mathcal{O}(\log\log n)$ approximation, as shown in this paper.

The rest of the paper is organized as follows. Section 2 gives a formal definition of our problems. Section 3 describes the local search algorithm, and Section 4 proves its approximation ratio. Section 5 has some concluding remarks and open problems. Due to space constraints some of the proofs are omitted from this abridged version of the paper.

2 Preliminaries and Problem Definitions

In all our problems we consider undirected graphs with no parallel edges. Let $G(V,E)$ be such a graph, with $n = |V|$, and $m = |E|$. For a subset $S \subseteq V$, define $\mathcal{N}_G(S)$ to be the subset of the edges in E with exactly one endpoint in S. For a vertex $v \in V$, let $d_G(v)$ denote the degree of v in G; that is, $d_G(v) = |\mathcal{N}_G(\{v\})|$. For a subset $S \subseteq V$, define $G[S]$ to be the subgraph induced by S. Any subset $S \subseteq V$ such that $G[S]$ is connected, defines a connected cut of size $|\mathcal{N}_G(S)|$.

A tree $T(U,F)$ is a *subtree* of graph G if T is a tree, $U \subseteq V$, and $F \subseteq E$. We denote the "subtree" relation by the symbol $\sqsubseteq$, namely, $T \sqsubseteq G$ indicates that T is a subtree of G. Let $V(T)$ be the set of vertices of T, $L(T)$ be the set of leaves of T, and $I(T)$ be the set of the internal vertices of T; that is, $L(T) = \{v \mid v \in U \wedge d_T(v) = 1\}$ and $I(T) = V(T) \setminus L(T)$. Obviously, $T(U,F)$ is a spanning tree of G iff $U = V$.

For any given vertex $r \in U$, if T is *rooted* at r, then all the edges in F are oriented from r outwards; namely, for $(x,y) \in F$, vertex x precedes y in the (unique) path from r to y. In this case we say that x is the *parent* of y and y is a *child* of x. Our convention is to specify the tree edge connecting x to its child y as (x,y). If vertex x is on the (unique) path from r to y then x is an *ancestor* of y and y is a *descendant* of x. We denote this relation by $x \rightsquigarrow y$. A vertex x is considered an ancestor and a descendant of itself (unless specified explicitly otherwise). For a subset $U' \subseteq U$, define $\mathsf{Desc}_T(U')$ to be the set of descendants of the vertices in U', and $\mathsf{LDesc}_T(U')$ to be the set of *leaf* descendants of the vertices in

U'. Namely, $\mathsf{Desc}_T(U') = \{y \mid \exists x \in U' \colon\ x \rightsquigarrow y\}$, and $\mathsf{LDesc}_T(U') = \mathsf{Desc}_T(U') \cap L(T)$. To simplify notation for singleton sets, we define $\mathsf{Desc}_T(u) = \mathsf{Desc}_T(\{u\})$. We assume that $n > 2$ (since the problem is trivial when $n \leq 2$), and for convenience we always root T at a vertex r for which $d_T(r) > 1$.

For a vertex $x \in U$, let $C_T(x)$ be the set of its children in T, and T_x be the subtree rooted at x; that is, the tree induced by the set of the descendants of x. For any two vertices $x, y \in U$, let $LCA(x, y)$ denote the lowest ancestor of x and y; defined as the unique vertex $u \in U$ such that u is an ancestor of both x and y, and none of the children of u is an ancestor of both x and y.

For a graph $G(V, E)$ and a tree $T(U, F)$, where $U \subseteq V$, define the *leaf degree* of T in G, denoted $\mathsf{LD}_G(T)$, as $\mathsf{LD}_G(T) = \sum_{v \in L(T)} d_G(v)$; in words, the leaf degree of T is the sum of the degrees in G of the leaves of T.

The input for both the Connected Max Cut (CMC) problem and the Max Leaf Degree (MLD) problem is a graph $G(V, E)$. The output of the CMC problem is a connected cut of maximum size. The output of the MLD problem is a subtree $T_{\mathsf{opt}} \sqsubseteq G$ with maximum leaf degree, i.e., $T_{\mathsf{opt}} = \arg\max_{T \sqsubseteq G} \{\mathsf{LD}_G(T)\}$.

Hajiaghayi et al. [6] proved that the CMC problem is NP-Hard even when the input is restricted to planar graphs. The MLD problem is NP-Hard as well. To see this note that when the graph $G(V, E)$ is regular the MLD problem is equivalent to the unweighted MLT problem (since all vertices have the same weight), which is equivalent to the unweighted MLST problem, as noted above. The unweighted MLST problem on regular graphs was shown to be NP-Hard in [12, 14] and thus the MLD problem is also NP-Hard even when the input is restricted to regular graphs. The MLD problem is APX-Hard as well since the unweighted MLST problem on 5-regular graphs is APX-Hard [14].

Both problems have a weighted version which is not considered in this paper. In the weighted CMC problem every edge $e \in E$ has a weight and the output of the weighted CMC problem is a connected cut of maximum weight. In the MLT problem, which is the weighted MLD problem, every vertex $v \in V$ has a weight $w(v)$ and the output of the MLT problem is a subtree $T_{\mathsf{opt}} \sqsubseteq G$ with maximum leaf weight, i.e., $T_{\mathsf{opt}} = \arg\max_{T \sqsubseteq G} \left\{\sum_{v \in L(T)} w(v)\right\}$. (Recall that the MLD problem is a special case of the weighted MLT problem in which the weight of the vertices is equal to their degree.)

Since both the CMC and the MLD problems are NP-Hard they call for approximation algorithms. Both are maximization problems. We say that an approximation algorithm for a maximization problem achieves an $\alpha \geq 1$ approximation ratio if the value of its output is at least $\frac{1}{\alpha}$ of the maximum. Gandhi et al. [4] proved a relation between the approximation of these problems.

▶ **Theorem 1** ([4]). *An α-approximation algorithm for the MLD problem implies a 4α-approximation algorithm for the CMC problem.*

We note that [4] considered the weighted CMC and MLT problems. For an edge weighted CMC instance the weight of a vertex v in the corresponding MLT instance is the sum of the weights of the edges in $\mathcal{N}_G(\{v\})$. It is easy to see that the MLD problem corresponds to the unweighted CMC problem since in the MLD problem the weight of every vertex equals its degree.

3 The Local Search Algorithm for MLD

Given a graph $G(V, E)$, we present a local search algorithm that finds a subtree $T^* \sqsubseteq G$ such that $\mathcal{O}(\mathsf{LD}_G(T^*) \log \log n) = \max_{T \sqsubseteq G} \{\mathsf{LD}_G(T)\}$. We start by computing any spanning tree $T(V, F)$ of G, and root it at an arbitrary vertex $r \in V$. The solution generated by our algorithm will be a subtree of G, also rooted at r. The algorithm proceeds iteratively by performing local search improvements. Each local search improvement considers an internal vertex $v \in I(T)$ and determines whether making v a leaf, while potentially removing some of the current leaves in T, results in an improved tree. We call such an operation a *vertex improvement* defined below. The algorithm terminates when no local search improvements are available and the current tree T at this point is returned as T^*.

3.1 Vertex improvement

To define the improvements we first introduce the notion of *dependent* children. Let $T \sqsubseteq G$ be a subtree of G. Consider a vertex $v \in I(T)$, and let $C_T(v) \subseteq V(T)$ be the set of children of v in T. For a vertex $x \in C_T(v)$, consider $G[I(T) \cup \{x\} \setminus \{v\}]$ – the subgraph of G induced by x, and the vertices in $I(T)$, except for v. We say that x is a *dependent* child of v if x is disconnected from r in this induced subgraph. Note that a *dependent* child of v depends on v for its connectivity to r in the subgraph $G[I(T) \cup \{x\}]$ (implying that v is a separating vertex in $G[I(T) \cup \{x\}]$). If a vertex $x \in C_T(v)$ is not a dependent child then it is an *independent* child of v. Let $D_T(v) \subseteq C_T(v)$ be the (possibly empty) set of dependent children of v in T. (For the root r of T, trivially $D_T(r) = C_T(r)$.) Figure 1 illustrates a vertex v and its *independent* child x.

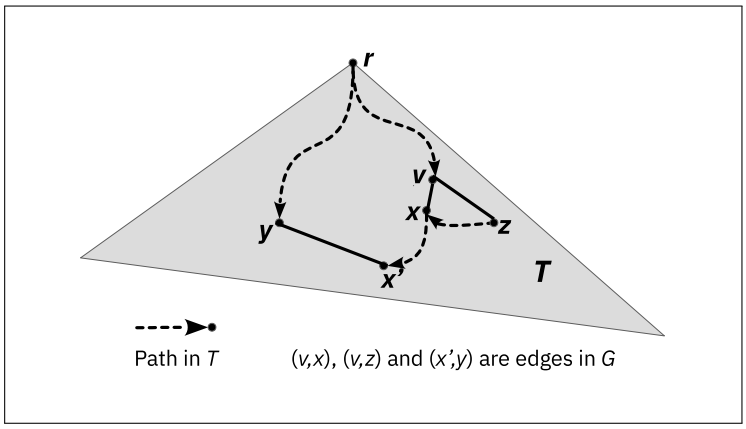

Figure 1 Vertices x and z are independent children of v since the paths from z to x and from x to r via (x', y) use only internal vertices of T and avoid v.

▶ **Lemma 2.** *For a subtree $T(U, F) \sqsubseteq G$ and a vertex $v \in I(T)$, if v has any independent children, then there exists an independent child $x \in C_T(v) \setminus D_T(v)$ and a subtree $T'(U, F') \sqsubseteq G$, such that $C_{T'}(v) = C_T(v) \setminus \{x\}$, $L(T) \subseteq L(T') \subseteq L(T) \cup \{v, x\}$, and $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$.*

Proof. Consider an independent child $z \in C_T(v) \setminus D_T(v)$ (see Figure 1). By the definition of an independent child there exists a path in G that connects z to the root r with the property that all the vertices along this path are in $I(T) \setminus \{v\}$. Follow such a path starting at z. It must include an edge (x', y) such that $LCA(x', y)$ is an ancestor of v other than v. (This is because r is not a descendant of v.) Consider the first such edge along the path. For this

edge x' is a descendant of v (other than v) and y is not a descendant of v. Adding the edge (x', y) to the set F of the edges of T creates a cycle that consists of the tree paths from $LCA(x', y)$ to x' and to y and the edge (x', y). Since x' is a descendant of v other than v, the path from $LCA(x', y)$ to x' contains an edge that connects v to one of its children. Let this child be $x \in C_T(v)$. Clearly, x is also an independent child since the path from x to r that uses the tree path from x to x', the edge (x', y) and the tree path from y to r avoids v.

Let $F' = F \cup \{(x', y)\} \setminus \{(v, x)\}$. (See also Figure 1.) All the vertices in U are reachable from r via edges of F' as follows. The vertices that are not descendants of x are reachable in the same way as before and the ones which are descendants of x are reachable through the edge (x', y) since y (which is not a descendant of x in T) is reachable from r and all the descendants of x are reachable from y via edge (x', y) and the edges in the subtree rooted at x as all these edges are in F'. Since $|F'| = |F| = |U| - 1$ we have that $T'(U, F')$ is a tree that spans the vertex set U, and thus $T'(U, F') \sqsubseteq G$.

The path from x to r in T' avoids v, and thus x is not a descendent of v in T'. This is the only change in the set of children of v, implying that $C_{T'}(v) = C_T \setminus \{x\}$. The only vertices in U whose degree in T' may be larger than their degree in T are y (always) and x' (when $x' \neq x$). However, $y \in I(T)$ and when $x' \neq x$ also $x' \in I(T)$. Thus, none of the leaves of T may become internal vertices in T'. On the other hand the degrees of v and x (when $x' \neq x$) in T' are 1 less than their degrees in T and thus they may become leaves in T', implying that $L(T) \subseteq L(T') \subseteq L(T) \cup \{v, x\}$. Since $L(T) \subseteq L(T')$, we also have $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$. ◀

Given a subtree $T(U, F) \sqsubseteq G$ and a vertex $v \in I(T)$, Lemma 2 can be applied successively to obtain a tree $T'(U, F')$ in which v has no independent children. This is done as follows. Start by initializing T' to T. If v has an independent child in T', apply Lemma 2 to get a modified tree T' that spans the same set of vertices U and satisfies the conditions of the lemma. If vertex v still has an independent child in the modified T', then repeat the application of Lemma 2. The process is bound to terminate since the number of children of v is decreased in each iteration. Note that when the process terminates $L(T) \subseteq L(T')$ and thus $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$.

The function VertexImprove(G, T, v) described in Algorithm 1 gets a graph $G(V, E)$, a subtree $T(U, F) \sqsubseteq G$, and a vertex $v \in I(T)$ as parameters. It determines whether $\mathsf{LD}_G(T)$ can be improved by making v a leaf. It returns a subtree $T'(U', F') \sqsubseteq G$ such that $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$. (In case of improvement $\mathsf{LD}_G(T') > \mathsf{LD}_G(T)$.)

Algorithm 1 Check whether the leaf degree can be improved by making v a leaf.

```
1:  function VertexImprove(G, T, v)
2:      Parameters: graph G(V, E), subtree T(U, F) ⊑ G, vertex v
3:      Returned value: subtree T' ⊑ G, s.t. LD_G(T') ≥ LD_G(T)
4:      T' ← T
5:      while ∃x ∈ C_{T'}(v) ∧ ∃(x', y) ∈ E such that x ⇝ x' and v ↛ y do
6:          Update tree T' by swapping the tree edge (v, x) with the nontree edge (x', y)
7:      if v is a leaf in T' then
8:          return T'
9:      else if d_G(v) ≤ 2LD_G(T'_v) then                ▷ no substantial improvement
10:         return T'
11:     else                          ▷ remove the subtrees rooted at the children of v
12:         return T' ← T'[(U \ Desc_{T'}(v)) ∪ {v}]
```

The function VertexImprove starts by modifying the tree T to a tree $T'(U, F')$ in which v has no independent children as described above. (Note that any independent child of v before the modification becomes an independent child of another vertex in $I(T)$.) At this point v may either be a leaf in T' or an internal vertex with only dependent children. If v is a leaf, then $L(T) \cup \{v\} \subseteq L(T')$ and $\mathsf{LD}_G(T') > \mathsf{LD}_G(T)$, resulting in an improvement of the leaf degree. In this case tree $T'(U, F')$ is returned by the function. If v has only dependent children, then function VertexImprove checks whether removing the subtrees rooted at the children of v, and thus making v a leaf improves the leaf degree (substantially). (See Figure 2.) Specifically, let $T'[\tilde{U}]$ be the subgraph of T' induced by the vertex set $\tilde{U} = (U \setminus \mathsf{Desc}_{T'}(v)) \cup \{v\}$. Clearly, $T'[\tilde{U}]$ is connected and thus $T'[\tilde{U}] \sqsubseteq G$. Compare $\mathsf{LD}_G(T')$ and $\mathsf{LD}_G(T'[\tilde{U}])$: The only leaf in $T'[\tilde{U}]$ that is not a leaf in T' is v. On the other hand the only leaves in T' that are not in $T'[\tilde{U}]$ are the descendant leaves of v in T', namely, $\mathsf{LDesc}_{T'}(v)$. Let T'_v be the subtree of T' rooted at v. We get that $\mathsf{LD}_G(T'_v)$ is the sum of the degrees in G of the leaves in $\mathsf{LDesc}_{T'}(v)$. Consequently, $\mathsf{LD}_G(T'[\tilde{U}]) - \mathsf{LD}_G(T') = d_G(v) - \mathsf{LD}_G(T'_v)$. It follows that if $d_G(v) > \mathsf{LD}_G(T'_v)$, then $\mathsf{LD}_G(T'[\tilde{U}]) > \mathsf{LD}_G(T')$. To achieve our approximation ratio we perform an improvement step in this case only if the resulting improvement is "substantial"; that is, only if $d_G(v) > 2\mathsf{LD}_G(T'_v)$ or equivalently $\mathsf{LD}_G(T'[\tilde{U}]) - \mathsf{LD}_G(T') > \mathsf{LD}_G(T'_v)$. In this case $T'[\tilde{U}]$ is returned by the function VertexImprove(G, T, v).

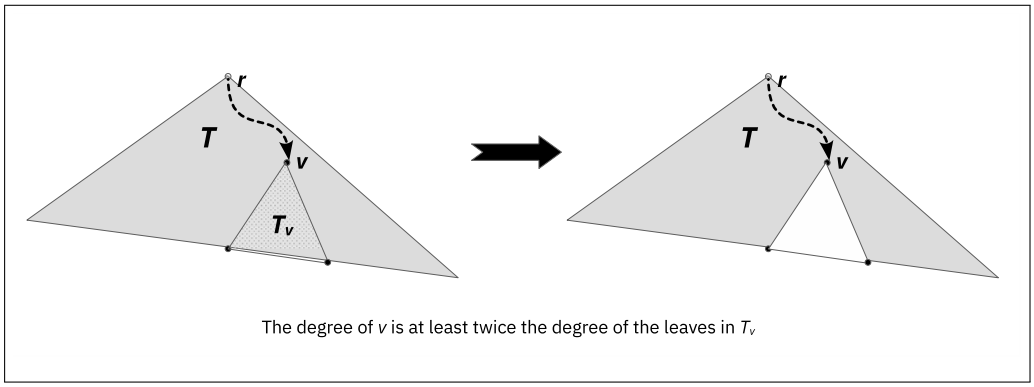

Figure 2 Removing the children of v and all their descendants.

It follows from the proof of Lemma 2 that as long as v has any independent children, the condition of the While loop in Line 5 of Algorithm 1 must be satisfied. After the execution of this While loop all the vertices in $C_{T'}(v)$ must be dependent children of v. Note that Algorithm 1 can be implemented in polynomial time.

3.2 Sequencing the improvement steps

Algorithm 2 starts from an arbitrary spanning tree and calls the function VertexImprove successively, as long as tree T has an internal vertex that was not considered as a candidate for improvement by this function. An internal vertex v is considered as a candidate for improvement only after all its nonleaf children were considered as candidates for improvement. Such an internal vertex is guaranteed to exist whenever there are internal vertices that were not considered for improvement. The output of Algorithm 2 is the final tree, denoted T^*.

We prove the following properties of the tree T^* that is the output of Algorithm 2.

▶ **Lemma 3.** *For every vertex* $v \in I(T^*)$, $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u) \leq 2\mathsf{LD}_G(T^*_v)$.

Algorithm 2 Local improvements.

Input: graph $G(V, E)$, and an arbitrary vertex $r \in V$
Output: a subtree $T^* \sqsubseteq G$ rooted at r

1: $T(V, F) \leftarrow$ a spanning tree rooted at r
2: $tested(v) \leftarrow$ false, $\forall v \in I(T)$
3: $tested(v) \leftarrow$ true, $\forall v \in L(T)$
4: **while** $\exists v \in I(T)$ s.t. $tested(v) =$ false **do**
5: find a vertex $v \in I(T)$ s.t. $tested(v) =$ false $\wedge$ $\forall u \in C_T(v)$ $tested(u) =$ true
6: $T \leftarrow$ VertexImprove(G, T, v)
7: $tested(v) \leftarrow$ true
8: **return** $T^* \leftarrow T$

Proof. Algorithm 2 starts with a spanning tree T rooted at r. The tree T is modified in every call to VertexImprove. Each such change may involve the removal of vertices from T and conversion of some vertices from internal vertices to leaves. However, a leaf is never changed to an internal vertex, and a vertex that was removed from T will never be brought back. Consider an internal vertex $v \in I(T^*)$ and the function call VertexImprove(G, T, v). Fix T to be the tree after this call. Vertex v has to be an internal vertex of T (as otherwise it would not be in $I(T^*)$). It follows that after this call $d_G(v) \leq 2\mathsf{LD}_G(T_v)$, and all the vertices in $C_T(v)$ are dependent children of v. Since all these vertices were children of v also before the function call VertexImprove(G, T, v), then by our algorithm they were considered for improvement before this call. It follows that removing any of these vertices in subsequent calls would imply also the removal of v, and since removed vertices are not brought back this did not happen. Thus, $D_T(v) = C_T(v) \subseteq C_{T^*}(v)$. Since $I(T^*) \subseteq I(T)$ all these vertices are also dependent children of v in T^* and thus $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. Clearly, $2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u) \leq 2\mathsf{LD}_G(T^*_v)$. ◀

▶ **Lemma 4.** *There are no edges connecting vertices in $I(T^*)$ to vertices outside T^*.*

Proof. We show that the following assertion holds throughout the computation of Algorithm 2: there are no edges connecting vertices in $I(T)$ to vertices outside T. The lemma follows since T^* is the final value of T. The assertion holds at the start of Algorithm 2 since initially T is a spanning tree and thus there are no vertices outside T. Suppose that the assertion holds for T until the function call VertexImprove(G, T, v). We need to show that it holds also after this call. This clearly holds in case the vertices of T are not changed in this call. The only nontrivial case is when some vertices are removed from T in this call; that is, Line 12 in Algorithm 1 is executed. Recall that the removed vertices are all the descendants of v (other than v itself) in the tree T', and that in tree T' all the children of v are dependent children; i.e., $D_{T'}(v) = C_{T'}(v)$. This implies that none of the vertices in $desc_{T'}(v)$ are connected to an internal vertex of $T'[(U \setminus \mathsf{Desc}_{T'}(v)) \cup \{v\}]$ (which is the returned value of T) as this will imply that at least one of the vertices in $C_{T'}(v)$ is independent. Thus, the assertion holds for T also after the call. ◀

It is not difficult to see that Algorithm 2 can be implemented in polynomial time. We remark that the efficiency of the algorithm can be improved by avoiding the search for a vertex v for which $tested(v) =$ false $\wedge$ $\forall u \in C_T(v)$ $tested(u) =$ true (Line 5). This can be done by computing a postorder of the vertices of the initial tree T (which is a spanning tree) and following this order when the vertices are considered for improvement. Recall that a vertex v appears in a postorder after all its descendants. So, this is the case for the initial

tree. Now, suppose that Algorithm 2 follows the vertices in this order, and a vertex v is considered for improvement. At this point all the descendants of v in the initial tree were considered already. Any descendant of v at this point that was not a descendant of v in the original tree became a descendant as a result of a move of vertices in a function call to VertexImprove. The only vertices that are moved in this function call must have been considered for improvement already since they are the descendant of the vertex considered for improvement in the call to VertexImprove. Thus, when a vertex v is considered for improvement all its descendants were already considered for improvement.

4 The Approximation Ratio of the Algorithm

Recall that T^* is the tree computed by Algorithm 2. We start by proving that the number of edges with at least one endpoint in $V(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. We prove this bound separately for tree edges and nontree edges. Next, we show that the total number of edges of G outside T^* is also $\mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. It follows that $m = \mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. Since for any subtree S of G, $\mathsf{LD}_G(S) \leq 2m$, we get that for any subtree S of G, $\mathsf{LD}_G(S) = \mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. This implies that Algorithm 2 is an $\mathcal{O}(\log\log n)$-approximation algorithm for both the MLD and the respective CMC problems.

4.1 Bounding the number of tree edges in T^*

We show that the number of tree edges is $\mathcal{O}(\mathsf{LD}_G(T^*))$. For this we upper bound the number of vertices in T^* which also bounds the number of tree edges in T^*. Clearly, the number of leaves is $|L(T^*)|$. Also, the number of vertices of degree at least 3 in T^* is bounded by $|L(T^*)| - 1$. We still need to bound the number of vertices of degree 2 in T^*. To prove this bound we apply a lemma from [6] that states that WLOG it can be assumed that the input graph to the CMC problem and thus also to the respective MLD problem does not contain a path of three vertices of degree 2.

▶ **Lemma 5** ([6]). *WLOG it can be assumed that the input graph to the CMC problem and thus also to the respective MLD problem does not contain a path of three vertices of degree 2.*

We note that [6] considered the slightly more general $\{0,1\}$-weighted case rather than the unit weight case considered here.

For the remainder of our analysis we color the edges of T^* in two colors *red* and *blue*. For $v \in I(T^*)$, a tree edge (v,u) is colored red if u is a dependent child of v and colored blue otherwise. Trivially, all the edges outgoing from r are red. By Lemma 3 every vertex $v \in I(T^*)$ has at least one outgoing red edge. We now bound the number of degree 2 vertices in T^*.

▶ **Lemma 6.** *The number of degree 2 vertices in T^* is bounded by $8|L(T^*)| + 3\mathsf{LD}_G(T^*)$.*

Proof. Consider the subgraph of T^* induced by its degree 2 vertices. This subgraph is a collection of paths. The endpoint of each such path is the parent of a distinct vertex of degree other than 2 in T^*. Hence, the number of these paths is bounded by the number of vertices with degree other than 2 in T^*. As stated above, the number of vertices of degree 1 in T^* is $|L(T^*)|$, and the number of vertices of degree 3 or more is bounded by $|L(T^*)| - 1$. Thus, the total number of these paths is bounded by $2|L(T^*)| - 1$. Consider the paths of length (in edges) at most 3. Let n_3 be the number of these paths. The number of vertices of degree 2 on these paths is bounded by $4n_3$.

Consider the rest of the paths, and denote their number by $n_{\geq 4}$. Let $v_1, v_2, v_3, \ldots, v_\ell$, for $\ell \geq 5$, be such a path. Note that the edges (v_i, v_{i+1}), for $i \in [1..\ell-1]$, must be red since every internal vertex must have at least one dependent child and v_{i+1} is the only child of v_i; namely, $C_{T^*}(v_i) = D_{T^*}(v_i) = \{v_{i+1}\}$. By Lemma 5 G does not contain a path of three vertices of degree 2. Thus, at least one out of every three consecutive vertices on the path $v_2, \ldots, v_{\ell-1}$ must have degree at least 3 in G. Hence, at least $\lfloor \frac{\ell-2}{3} \rfloor$ vertices on the path $v_2, \ldots, v_{\ell-1}$ must have degree at least 3 in G. Let v_i be one of these vertices, and let $y \notin \{v_{i-1}, v_{i+1}\}$ be a vertex connected to v_i in G ($y \notin \{v_{i-1}, v_{i+1}\}$ because G has no parallel edges). By Lemma 4 vertex v_i is not connected to vertices outside T^*, and hence $y \in V(T^*)$. We claim that y must be a leaf. To obtain a contradiction assume that this is not the case and consider $LCA(v_i, y)$. If $LCA(v_i, y) \neq v_i$, then it must be an ancestor of v_{i-1} but not v_{i-1} itself (since v_i is the only child of v_{i-1}). In this case v_i cannot be a dependent child of v_{i-1} because the path from v_i to y using edge (v_i, y) and then up the tree to $LCA(v_i, y)$ avoids v_{i-1}. If $LCA(v_i, y) = v_i$, then y must be a descendant of v_{i+1} but not v_{i+1} itself. In this case the only child of v_{i+1} cannot be a dependent child because the path from this child down the tree to y and then to v_i using edge (v_i, y) avoids v_{i+1}.

It follows that at least $\lfloor \frac{\ell-2}{3} \rfloor \geq \frac{\ell-4}{3}$ vertices on every path $v_1, v_1, v_2, \ldots, v_\ell$ of degree 2 vertices in T^*, where $\ell \geq 4$, are connected to leaves in $L(T^*)$. Thus, each one of the $n_{\geq 4}$ paths of length $\ell \geq 4$ of degree 2 vertices in T^* contributes at least $\frac{\ell-4}{3}$ to the leaf degree of T^*, which equals $\mathsf{LD}_G(T^*)$. Summing this over all such paths implies that the total number of vertices of degree 2 on these paths is bounded by $3\mathsf{LD}_G(T^*) + 4n_{\geq 4}$. Since $n_3 + n_{\geq 4} < 2|L(T^*)|$ the number of degree 2 vertices is bounded by $8|L(T^*)| + 3\mathsf{LD}_G(T^*)$. ◀

We conclude the following theorem.

▶ **Theorem 7.** *The number of tree edges in T^* is bounded by $10|L(T^*)| + 3\mathsf{LD}_G(T^*) = \mathcal{O}(\mathsf{LD}_G(T^*))$.*

4.2 Bounding the number of nontree edges with at least 1 endpoint in $V(T^*)$

The number of nontree edges with at least one endpoint in $L(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*))$. Thus, from now on we concentrate on bounding the number of nontree edges with at least one endpoint in $I(T^*)$, and the other not in $L(T^*)$. We have already established in Lemma 4 that a vertex in $I(T^*)$ cannot be connected to a vertex outside T^*. It follows that it suffices to bound the number of nontree edges with both endpoints in $I(T^*)$.

Recall that for $v \in I(T^*)$, a tree edge (v, u) is colored red if u is a dependent child of v and colored blue otherwise. For a vertex $v \in I(T^*)$, let $B_{T^*}(v)$ be the subset of vertices of T^* that can be reached from v by paths that start at a red edge outgoing from v and then use only blue edges. In other words, $B_{T^*}(v)$ consists of all the dependent children of v and all the vertices that are reachable from these dependent children using only blue edges. (Notice that $v \notin B_{T^*}(v)$.) In the next lemma we prove that an internal vertex $x \in B_{T^*}(v)$ cannot be connected to an internal vertex of T^* that is neither v nor in $B_{T^*}(v) \cup B_{T^*}(x)$.

▶ **Lemma 8.** *If an internal vertex $x \in B_{T^*}(v)$ is connected by a nontree edge to a vertex $y \in I(T^*)$, then $y \in \{v\} \cup B_{T^*}(v) \cup B_{T^*}(x)$.*

Proof. To obtain a contradiction assume that x is connected to a vertex $y \in I(T^*) \setminus (\{v\} \cup B_{T^*}(v) \cup B_{T^*}(x))$. We consider 4 cases.

Case 1. y is a descendant of x. Follow the path from x to y in T^*. The vertex immediately after x along this path must be in $B_{T^*}(v) \cup B_{T^*}(x)$. By our assumption $y \notin B_{T^*}(v) \cup B_{T^*}(x)$. Thus, the path from x to y in T^* must leave either $B_{T^*}(v)$ or $B_{T^*}(x)$. Note that for any vertex $u \in I(T^*)$, any tree edge connecting a vertex in $B_{T^*}(u)$ to a vertex outside $B_{T^*}(u)$ must be red. It follows that the path from x to y in T^* must include a red edge (x', x''), where $x' \neq x$ is a descendant of x. Note that vertex x'' is a dependent child of x' since (x', x'') is red. This is a contradiction since the path from x'' down the tree to y, followed by the edge (x, y), and the path from x down the tree to the parent of x' avoids x', and thus x'' cannot be a dependent child of x'.

Case 2. vertex y is a descendant of a vertex $z \in B_{T^*}(v)$, where $z \neq x$, and y is not a descendant of x. Since $y \notin B_{T^*}(v)$ the path from z to y in T^* must include a red edge (z', z''), where $z' \in B_{T^*}(v)$. In this case z'' is an ancestor of y, and thus z'' is connected to x through the edge (x, y). To obtain a contradiction we show that there exists a path from x to the parent of z' that avoids the vertex z'. This implies that z'' cannot be a dependent child of z' which is a contradiction. Consider $LCA(x, z')$. By our assumption in this case x is not an ancestor of y and thus it also cannot be an ancestor of z'. Thus, $LCA(x, z') \neq x$. In case $LCA(x, z') \neq z'$ the path from x to the parent of z' goes from x to $LCA(x, z')$ and then down the tree to the parent of z'. In case is $LCA(x, z') = z'$ we must have that z' is an ancestor of x. Since both x and z' are in $B_{T^*}(v)$, all the edges on the path in T^* from z' to x are blue. But then the ancestor of x which is the child of z' is not a dependent child of z', and thus there is a path from x to the parent of z' that goes through this child and avoids z'.

In the remaining two cases vertex y is not a descendant of any vertex in $B_{T^*}(v)$.

Case 3. Vertex y is not a descendant of v. In this case there is a path from y to the parent of v that avoids v. Let x' be the child of v that is an ancestor of x. Since $x \in B_{T^*}(v)$, the tree edge (v, x') is colored red, and vertex x' is a dependent child of v but this is a contradiction since the path from x' to x and then through the edge (x, y) to the parent of v avoids v.

Case 4. Vertex y is a descendant of v. Consider the first edge (v, y') on the path from v to y in T^*. Since y is not a descendant of any vertex in $B_{T^*}(v)$ this edge must be colored blue, and thus y' is an independent child of v and there exists a path from y' to the parent of v that avoids v. As in the previous case, let x' be the child of v that is an ancestor of x. The tree edge (v, x') is colored red, and vertex x' is a dependent child of v. This is a contradiction since the path from x' that goes down the tree to x, then to y through the edge (x, y), then up the tree to y', and then from y' to the parent of v, avoids v. ◀

Consider a vertex $x \in V(T^*)$ that is not the root r. Since all the edges outgoing from r are red, there must be at least one red edge (v, v') on the path up the tree from x to r. Thus there exists a vertex $v \in I(T^*)$ such that $x \in B_{T^*}(v)$. Note that v is the unique ancestor of x with the property that the (unique) path from v to x in T^* starts in a red edge and then uses only blue edges. It follows that any two sets $B_{T^*}(v)$ and $B_{T^*}(u)$ are disjoint. Thus, the sets $B_{T^*}(v)$ form a partition of the vertices in T^* (excluding r). This implies that the total number of edges of the form (v, x), where $v \in I(T^*)$ and $x \in B_{T^*}(v) \cap I(T^*)$ is bounded by $|I(T^*)|$ (since there are no parallel edges). Clearly, $|I(T^*)|$ is bounded by the number of tree edges, and thus by Theorem 7 $|I(T^*)| = \mathcal{O}(\mathsf{LD}_G(T^*))$.

It follows from Lemma 8 that we are left with bounding the number of nontree edges (x, y), where both x and y are in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. To bound the number of these edges we modify the tree T^* to a tree $\tilde{T}$ that spans the same set of vertices as T^* in which all the vertices in $B_{T^*}(v)$ become the children of v. Note that unlike the tree T^*, the tree $\tilde{T}$ is not a subtree of G.

4.2.1 The modification of T^* to $\tilde{T}$

Tree $\tilde{T}$ is obtained from $T^*(U, F)$ by making all the vertices in $B_{T^*}(v)$ the children of v, for every $v \in I(T^*)$. Formally, for every $v \in I(T^*)$, edge (v, x) is a tree edge of $\tilde{T}$ iff $x \in B_{T^*}(v)$. To illustrate the definition, consider a vertex $x \in B_{T^*}(v)$ that is not a child of v in T^*, and let y be the parent of x in T^*. Note that the edge (y, x) is a blue edge and x is thus an independent child. We remove the edge (y, x) and instead add the edge (v, x), making v the new parent of x. Let $\tilde{T}$ be the resulting tree. It is not difficult to see that $\tilde{T}$ is indeed a tree that spans the set U. This is because the parent of any vertex v in $\tilde{T}$ is an ancestor of its parent in T^*.

▶ **Lemma 9.** *Any two vertices $\{x, y\} \subseteq B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$ are siblings in $\tilde{T}$ and share v as their parent.*

Proof. If both x and y are independent children in T^* then after the modification both become children of v. The only other vertices in $B_{T^*}(v) \cap I(T^*)$ are the dependent children of v and they remain children of v also in $\tilde{T}$. ◀

Recall that we need to bound the number of nontree edges of the form (x, y), where both x and y are in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. This is equivalent to bounding the number of the edges of G that connect *siblings* in $I(\tilde{T})$.

▶ **Lemma 10.** *For any vertex $v \in I(T^*)$, if x is a descendant of a dependent child of v in T^* then x is a descendant of v in $\tilde{T}$.*

By Lemma 3 for every vertex $v \in I(T^*)$, $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. By Lemma 10 any leaf descendant of a vertex $u \in D_{T^*}(v)$ is a descendant of v in $\tilde{T}$ and thus we have $d_G(v) \leq 2\mathsf{LD}_G(\tilde{T}_v)$. As shown above, to bound the number of nontree edges with both endpoints in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$, it suffices to bound the number of edges that connect siblings in $I(\tilde{T})$. For this we prove the following slightly more general theorem, where instead of considering the leaf degree, we assume general weights on the leaves of $\tilde{T}$. Associate a weight $w(\ell) \geq 1$ with every leaf $\ell \in L(\tilde{T})$. For a vertex $v \in I(\tilde{T})$, let $w(\tilde{T}_v)$ denote the total weight of the leaves of $\tilde{T}_v$, which is the subtree rooted at v. Also define $w(\tilde{T}) = w(\tilde{T}_r)$. The following theorem (proven below) may be interesting on its own.

▶ **Theorem 11.** *Let $\tilde{T}$ be a rooted tree with an integral weight $w(\ell) \geq 1$ associated with every leaf $\ell \in L(\tilde{T})$, and let $\tilde{G}(I(\tilde{T}), \tilde{E})$ be a graph with no parallel edges that all of its edges connect siblings in $I(\tilde{T})$. If for every $v \in I(\tilde{T})$, $d_{\tilde{G}}(v) \leq 2w(\tilde{T}_v)$, then $|\tilde{E}| = \mathcal{O}\big(w(\tilde{T}) \log\log w(\tilde{T})\big)$.*

To prove this theorem we further modify $\tilde{T}$ to $\hat{T}$ in which the degree of every internal vertex is lower and upper bounded by a constant times the weight of the leaves of its rooted subtree.

4.2.2 The modification of $\tilde{T}$ to $\hat{T}$

We obtain $\hat{T}$ from $\tilde{T}$ by removing some of the internal vertices of $\tilde{T}$. While doing so we maintain that the number of edges connecting siblings in $I(\hat{T})$ remains the same as the number of edges connecting siblings in $I(\tilde{T})$.

Initially, $\hat{T}$ is set to $\tilde{T}$ and $\hat{G}$ is set to $\tilde{G}$. Next, we traverse the vertices of $I(\hat{T}) \setminus \{r\}$ top down starting from the children of the root r. When a vertex v is traversed we check whether v has any children that are internal vertices, and if so whether $d_{\hat{G}}(v) < w(\hat{T}_v)$. (Recall that by the conditions of Theorem 11, initially, $d_{\hat{G}}(v) \leq 2w(\hat{T}_v)$.) If this is the case we execute the following modification steps.

Step 1. Remove v from $\hat{T}$ and connect all the children of v that are internal vertices to the parent of v. Denote the set of these children by C.

Step 2. Pick a vertex $x \in C$ and connect x to all the leaves of $\hat{T}$ that were children of v before its removal.

Step 3. For every edge (v, y) in $\hat{G}$ that connected v before its removal to a sibling in $\hat{T}$, find a vertex $z \in C$ such that $d_{\hat{G}}(z) < 3w(\hat{T}_z) - 1$, and add the edge (z, y) to $\hat{G}$ in lieu of the edge (v, y). Lemma 12 implies that such a vertex z always exists.

▶ **Lemma 12.** *After Step 2 of the modification steps*

$$d_{\hat{G}}(v) + \sum_{z \in C} d_{\hat{G}}(z) < 3 \sum_{z \in C} w(\hat{T}_z). \tag{1}$$

Lemma 12 implies that for each of the $d_{\hat{G}}(v)$ edges that connected v to its siblings we can find in Step 3 a vertex $z \in C$ such that $d_{\hat{G}}(z) < 3w(\hat{T}_z) - 1$. (Recall that the weights are integral.)

▶ **Lemma 13.** *After traversing all the vertices of $\hat{T}$ the following properties are satisfied.*

Property 1. *The graph $\hat{G}(I(\hat{T}), \hat{E})$ has no parallel edges and all of its edges connect siblings in $I(\hat{T})$.*

Property 2. *The number of edges of $\hat{G}$ is the same as the number of edges of $\tilde{G}$.*

Property 3. *For every internal vertex $v \in I(\hat{T}) \setminus \{r\}$ that is a parent of a nonleaf vertex in $\hat{T}$, $w(\hat{T}_v) \le d_{\hat{G}}(v)$.*

Property 4. *For every internal vertex $v \in I(\hat{T}) \setminus \{r\}$, $d_{\hat{G}}(v) \le 3w(\hat{T}_v)$.*

4.2.3 Bounding the number of edges connecting siblings in $\hat{T}$

To bound the number of edges connecting siblings in $\hat{T}$ we prove the following theorem.

▶ **Theorem 14.** *Let $\hat{T}$ be a rooted tree with an integral weight $w(\ell) \ge 1$ associated with every leaf $\ell \in L(\hat{T})$, and let $\hat{G}(I(\hat{T}), \hat{E})$ be a graph with no parallel edges that all of its edges connect siblings in $I(\hat{T})$. If for every $v \in I(\hat{T})$ that is a parent of a nonleaf vertex in $\hat{T}$, $w(\hat{T}_v) \le d_{\hat{G}}(v)$, and for every $v \in I(\hat{T})$, $d_{\hat{G}}(v) \le 3w(\hat{T}_v)$, then $|\hat{E}| = \mathcal{O}\Big(w(\hat{T}) \log\log w(\hat{T})\Big)$.*

To prove Theorem 14 we use the following simple inequality that holds for undirected graphs with no parallel edges.

▶ **Lemma 15.** *Let $H(V, E)$ be an undirected graph with no parallel edges and $m = |E|$. Let $U \subseteq V$ be the subset of vertices of degree at most $4\sqrt{m}$. Then, $\sum_{v \in U} d_H(v) > \frac{7}{8}m$.*

Proof of Theorem 14. To bound the number of edges of $\hat{G}$ we use a charging scheme. For a vertex $v \in I(\hat{T})$ that is a parent of at least 2 nonleaf vertices in $\hat{T}$, let $\hat{G}_v = \hat{G}[C_{\hat{T}}(v) \cap I(\hat{T})]$; that is, the subgraph of $\hat{G}$ induced by the children of v that are internal vertices. Note that $\hat{G}$ is the union of all these subgraphs since $\hat{G}$ only contains edges that connect siblings in $I(\hat{T})$. Let m_v be the number of edges in this subgraph. "Charge" these m_v edges to the subset of children of v whose degree in $\hat{G}_v$ and thus also in $\hat{G}$ is at most $4\sqrt{m_v}$. By Lemma 15 the sum of the degrees of all these children is greater than $\frac{7}{8}m_v$. Thus it suffices to charge each such child x of v the amount $\frac{8}{7} \cdot d_{\hat{G}}(x)$ to cover for all the m_v edges.

To bound the total number of edges we need to sum the charges of all the vertices. We distinguish between charged vertices that are parents of a nonleaf vertex and those with only leaf children. There are 3 cases based on the characteristics of the charged vertex $x \in C_{\hat{T}}(v) \cap I(\hat{T})$.

Case 1. The charged vertex x has only leaf children. By the conditions of the theorem $d_{\hat{G}}(x)$ is upper bounded by 3 times the weight of its leaf children. Since the leaf children of any two vertices are disjoint we get that the total amount charged to vertices with only leaf children is upper bounded by $\frac{8}{7} \cdot 3w(\hat{T})$.

Case 2. The charged vertex x is a parent of a nonleaf vertex and $d_{\hat{G}}(x) \leq 48 \cdot 3 = 144$. The amount charged to all such vertices is upper bounded by $\frac{8}{7} \cdot 144$ times the number of vertices in $I(\hat{T})$ that have siblings. The number of internal vertices in $\hat{T}$ that have siblings (and thus are children of a vertex whose degree is at least 3 in $\hat{T}$) is bounded by the number of leaves of $\hat{T}$. Note that $\mathcal{O}\Big(L(\hat{T})\Big) = \mathcal{O}\Big(w(\hat{T})\Big)$ since for every leaf $\ell \in L(\hat{T})$, $w(\ell) \geq 1$. We conclude that the amount charged to all such vertices is $\mathcal{O}\Big(w(\hat{T})\Big)$.

Case 3. The charged vertex x is a parent of a nonleaf vertex and $d_{\hat{G}}(x) > 144$. By our construction since x is a parent of a nonleaf vertex we have $w(\hat{T}_x) \geq \frac{1}{3} d_{\hat{G}}(x) > 48$ and $d_{\hat{G}}(x) > w(\hat{T}_x)$. Recall that the number of edges in $\hat{G}_v$ is m_v. Since $d_{\hat{G}}(x) \leq 4\sqrt{m_v}$ we have also $w(\hat{T}_x) < 4\sqrt{m_v}$. On the other hand we claim that $w(\hat{T}_v) \geq \frac{2}{3} m_v$. This holds since for every child $y \in C_{\hat{T}}(v) \cap I(\hat{T})$ we have $d_{\hat{G}}(y) \leq 3w(\hat{T}_y)$ summing over all such children of v we get that the sum of all degrees, which is $2m_v$, is bounded by 3 times the sum of the weights of the subtrees rooted at all these children. The sum of these weights is upper bounded by the weight of the subtree rooted at their parent v and thus we have $2m_v \leq 3w(\hat{T}_v)$. Combining the two inequalities (i) $w(\hat{T}_x) < 4\sqrt{m_v}$ and (ii) $w(\hat{T}_v) \geq \frac{2}{3} m_v$, we get $w(\hat{T}_v) \geq \frac{2}{3} m_v = \frac{1}{24} \left(4\sqrt{m_v}\right)^2 > \frac{1}{24} \left(w(\hat{T}_x)\right)^2 > 2^2 \cdot 24$ (since $w(\hat{T}_x) > 48$), and thus for any ancestor y of v it also holds that $w(\hat{T}_y) > \frac{1}{24} \left(w(\hat{T}_x)\right)^2$. Note that $\frac{1}{24} \cdot \left(2^{2^{i-1}} \cdot 24\right)^2 = 2^{2^i} \cdot 24$. Thus, applying the inequality recursively on all the ancestors of x that are also charged, implies that if i ancestors of the charged vertex x are also charged, then the weight of the subtree rooted at the highest such ancestor is at least $2^{2^i} \cdot 24$. This implies that the maximum number of such vertices that are charged along any path from the root to a leaf is $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$.

To sum the total amount charged to vertices with a nonleaf child and degree in $\hat{G}$ greater than 144, we consider them in "layers". The first layer consists of all vertices x such that (1) x has a nonleaf child and $d_{\hat{G}}(x) > 144$, (2) x was charged, and (3) none of the ancestors of x (other than x) were charged. Similarly, layer $i > 1$ consists of all vertices x such that (1) x has a nonleaf child and $d_{\hat{G}}(x) > 144$, (2) x was charged, and (3) x has an ancestor (other than x) that belongs to layer $i - 1$. The number of layers is bounded by the maximum number of vertices that are charged along any path from the root to a leaf which is $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$. By the construction of layers if vertices x and y are in the same layer then $\hat{T}_x$ and $\hat{T}_y$ are disjoint. Since the charge of a vertex x that is a parent of a nonleaf vertex is $\frac{8}{7} \cdot d_{\hat{G}}(x) \leq \frac{8}{7} \cdot 3w(\hat{T}_x)$ the total amount charged to the vertices in the same layer is $\frac{8}{7} \cdot 3w(\hat{T})$. The proof follows since there are $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$ layers. ◀

We note that the bound Theorem 14 is tight. We show a family of trees and associated graphs that achieve the upper bound in Theorem 14. Let S_1 be a tree of height 3 with 5 vertices: a root, 2 children of the root, and 2 leaves, each of which is a single child of a child of the root. Assign a weight of 1 to each of the 2 leaves. The graph $\hat{G}_1$ consists of a single edge connecting the 2 children of the root. For $i > 1$, the tree S_i is given by taking $2^{2^{i-1}}$ copies of

S_{i-1} and connecting them to a common root. The graph $\hat{G}_i$ includes $2^{2^{i-1}}$ copies of $\hat{G}_{i-1}$ associated with the copies of S_{i-1}. In addition, partition the $2^{2^{i-1}}$ children of the root of S_i into two sets of size $2^{2^{i-1}-1}$ and add the edges of the complete bipartite graph connecting the vertices in these 2 sets. The number of these edges is $\left(2^{2^{i-1}-1}\right)^2 = 2^{2^i-2}$. Figure 3 shows the tree S_2 and the associated graph $\hat{G}_2$. It can be verified that the construction obeys the conditions in Theorem 14. It can be shown by a simple induction that the number of leaves in S_i is 2^{2^i-1} and the number of edges in $\hat{G}_i$ is $i \cdot 2^{2^i-2}$ which achieves the bound tightly.

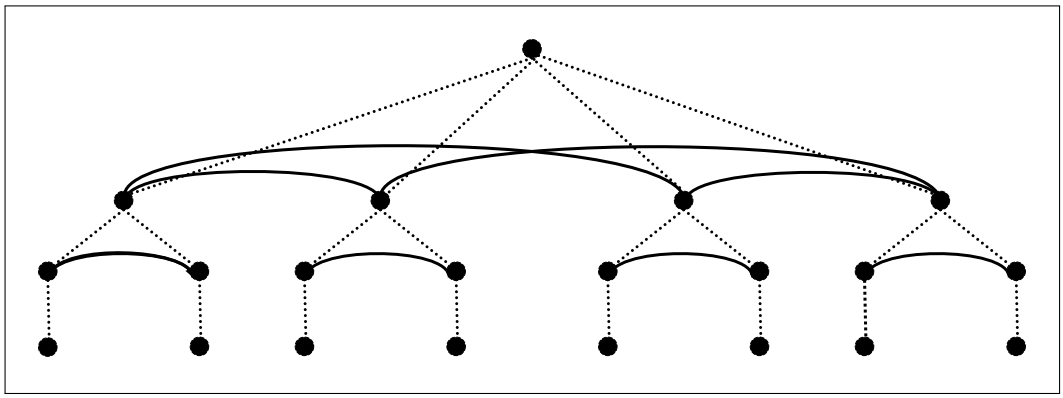

Figure 3 The tree S_2 and the associated graph $\hat{G}_2$. Tree edges are dashed and graph edges are solid.

We use Theorem 14 to prove Theorem 11.

Proof of Theorem 11. The number of edges of $\tilde{G}$ connecting siblings in $I(\tilde{T})$ is the same as the number of edges of $\hat{G}$ connecting siblings in $I(\hat{T})$ (see Property 2 above). Also, by our modification process $w(\tilde{T}) = w(\hat{T})$. By Properties 3 and 4 above the tree $\hat{T}$ and the graph $\hat{G}$ satisfy the conditions of Theorem 14. It follows that the number of edges connecting siblings in $I(\hat{T})$ and thus also in $I(\tilde{T})$ is $\mathcal{O}\big(w(\tilde{T}) \log\log w(\tilde{T})\big)$. ◀

The analysis in Sections 4.1 and 4.2 implies the following theorem.

▶ **Theorem 16.** *The number of edges with at least one endpoint in $V(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$.*

Proof. Theorem 7 implies that the number of tree edges in T^* is $\mathcal{O}(\mathsf{LD}_G(T^*))$. Clearly, the number of edges with one endpoint in $L(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*))$. As shown earlier the number of edges connecting a vertex v to vertices in $B_{T^*}(v) \cap I(T^*)$ is also $\mathcal{O}(\mathsf{LD}_G(T^*))$. By Lemma 8 the only other edges with at least one endpoint in $V(T^*)$ are edges connecting 2 vertices in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. By Lemma 3 for every vertex $v \in I(T^*)$, $d_G(v) \leq 2 \sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. This implies that after modifying T^* to $\tilde{T}$ the conditions of Theorem 11 hold with $w(\ell) = d_G(\ell)$, for every leaf $\ell \in L(T^*)$. The theorem follows since $\log\log|E(G)| = \mathcal{O}(\log\log n)$. ◀

4.3 Bounding the number of edges outside T^*

The next theorem bounds the number of edges in $G[V \setminus V(T^*)]$.

▶ **Theorem 17.** *The number of edges in $G[V \setminus V(T^*)]$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$.*

The proof of the theorem is omitted due to space constraints. It has two parts. First, we prove that there exists some constant c such that the number of edges added to $G[V \setminus V(T^*)]$ when the subtrees rooted at the children of a vertex v are removed from the tree is at most $c \cdot \mathsf{LD}_G(T'_v) \log\log n$. Second, we show that the total number of edges in $G[V \setminus V(T^*)]$ is bounded by $c \cdot \mathsf{LD}_G(T^*) \log\log n$ using the fact that that the subtrees rooted at the children of a vertex v are removed only when the improvement is substantial; namely, $d_G(v) > 2\mathsf{LD}_G(T'_v)$.

We conclude with the main theorem.

▶ **Theorem 18.** *The total number of edges in $G(V, E)$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$. Thus, Algorithm 2 is an $\mathcal{O}(\log\log n)$-approximation algorithm for both the MLD and the respective CMC problems.*

5 Conclusions and Future Research

We gave an $\mathcal{O}(\log\log n)$-approximation algorithm for the MLD problem that implies an approximation algorithm with the same quality to the CMC problem. The approximation algorithm consists of local improvement steps. While we have evidence that the *analysis of our algorithm* is tight, it is open whether *the approximation ratio* we achieved is tight or whether there exists a $o(\log\log n)$-approximation for the MLD and CMC problems.

Our algorithm uses the fact that the weights of the vertices in the MLD problem are the degrees of these vertices in an undirected graph with no parallel edges. It will be interesting to see if the same approach can be extended to other weights. It cannot be extended to general weights as there are examples of graphs with general edge weights in which the ratio of the total edge weight to the weighted connected cut is $\Omega(\log\log n)$.

To the best of our knowledge neither nontrivial approximation algorithms nor hardness of approximation results are known for non-monotone submodular function maximization subject to connectivity constraints. Any progress along these lines for either general or specific classes of graphs is bound to have numerous applications.

References

1. Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.*, 36(3):493–513, 1988.
2. Uriel Feige, Nicole Immorlica, Vahab S. Mirrokni, and Hamid Nazerzadeh. A combinatorial allocation mechanism with penalties for banner advertising. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 169–178, 2008.
3. Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vis.*, 59(2):167–181, 2004.
4. Rajiv Gandhi, Mohammad Taghi Hajiaghayi, Guy Kortsarz, Manish Purohit, and Kanthi K. Sarpatwar. On maximum leaf trees and connections to connected maximum cut problems. *Inf. Process. Lett.*, 129:31–34, 2018.
5. MohammadTaghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi K. Sarpatwar. Approximation algorithms for connected maximum cut and related problems. In Nikhil Bansal and Irene Finocchi, editors, *ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 693–704. Springer, 2015.
6. MohammadTaghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi K. Sarpatwar. Approximation algorithms for connected maximum cut and related problems. *Theor. Comput. Sci.*, 814:74–85, 2020.

7 Jason D. Hartline, Vahab S. Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 189–198, 2008.

8 Bart M. P. Jansen. Kernelization for maximum leaf spanning tree with positive vertex weights. *J. Graph Algorithms Appl.*, 16(4):811–846, 2012.

9 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

10 Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

11 Jon Lee, Viswanath Nagarajan, and Xiangkun Shen. Max-cut under graph constraints. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 50–62, 2016.

12 Paul Lemke. The maximum leaf spanning tree problem for cubic graphs is NP-Complete. Technical report, University of Minnesota, 1988. Technical report, IMA publication no. 428.

13 Slav Petrov. Image segmentation with maximum cuts, 2005.

14 Alexander Reich. Complexity of the maximum leaf spanning tree problem on planar and regular graphs. *Theoretical Computer Science*, 626:134–143, 2016.

15 Andreas S. Schulz and Nelson A. Uhan. Approximating the least core value and least core of cooperative games with supermodular costs. *Discrete Optimization*, 10(2):163–180, 2013.

16 Roberto Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 1998.