Kernelization for Maximum Leaf Spanning Tree with Positive Vertex Weights

Bart Jansen

Technical Report UU-CS-2009-027
Revised May 2010 (original version November 2009)

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Kernelization for Maximum Leaf Spanning Tree with Positive Vertex Weights

Bart Jansen*

bart@cs.uu.nl

Abstract. In this paper we consider a natural generalization of the well-known Max Leaf Spanning Tree problem. In the generalized Weighted Max Leaf problem we get as input an undirected connected graph G, a rational number k not smaller than 1 and a weight function $w : V \mapsto \mathbb{Q}_{\geq 1}$ on the vertices, and are asked whether a spanning tree T for G exists such that the combined weight of the leaves of T is at least k. We show that it is possible to transform an instance $\langle G, w, k \rangle$ of Weighted Max Leaf in linear time into an equivalent instance $\langle G', w', k' \rangle$ such that $|V(G')| \leq 5.5k$ and $k' \leq k$. In the context of fixed parameter complexity this means that Weighted Max Leaf admits a kernel with $5.5k$ vertices. The analysis of the kernel size is based on a new extremal result which shows that every graph $G = (V, E)$ that excludes some simple substructures always contains a spanning tree with at least $|V|/5.5$ leaves. [1]

1 Introduction

The area of fixed parameter complexity theory was pioneered by Downey and Fellows [1] to cope with the "rock of intractability" of NP-complete problems. Much of the complexity theoretic work of the past decades has been spent proving that there is an abundance of natural and important problems that are NP-complete, and for which the existence of a polynomial-time algorithm therefore seems unlikely. Parameterized complexity is an approach to deal with the intractability of such problems; rather than trying to find a polynomial-time algorithm for some decision problem $L \subseteq \Sigma^*$, we look more carefully at problem instances and associate every instance with a *parameter* value k that describes the structure of the instance. We then try to confine the seemingly unavoidable exponential factor in the running time of an algorithm to some function that depends only on k. This leads to the natural definition of a *parameterized problem* as a set $L_p \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(x, k) \in L_p$ of a parameterized problem we can think of x as the "classical" problem, whereas k is the new parameter that expresses some (structural) property of x. A natural choice of the parameter value is the desired *solution size*. Taking the Vertex Cover problem as an example, where we are asked whether a given graph has a vertex cover of a certain size, we could take the desired size k of the vertex cover as the parameter. Through a long series of successive improvements it was shown that the vertex cover problem can be decided in $O(1.2738^k + kn)$ time [2]. This shows that if the vertex cover we are looking for is small, then the problem can be solved efficiently - even on very large graphs! One possible formalization of this concept is the notion of the complexity class (strongly uniform) FPT (for *Fixed Parameter Tractable*), that consists of all parameterized problems L_p for which there is an algorithm that decides whether $(x, k) \in L_p$ running in time $f(k)p(|x|)$, where f is a computable function and p a polynomial. Parameterized complexity theory thus serves as a tool to deal with the intractability of NP-complete problems by exploiting the structure that lots of real-world problems have. This brings about a shift in perspective from negative results (NP-completeness) to positive results (FPT algorithms). We argue that this also calls for a shift in the type of problems that should be considered.

* This work was supported by the Netherlands Organisation for Scientific Research (NWO), project "KERNELS: Combinatorial Analysis of Data Reduction".

[1] This technical report is a revision of an earlier work with the same name. We have added Section 5 on hardness of approximation, and Appendix (A) containing a simple proof of a weaker form of the extremal result.

When a problem is *intractable* it is interesting to study the restrictions under which it remains intractable; this yields fundamental information about the structure of the problem, and it might also lead to the conclusion that there are restricted yet practically relevant versions of the problem which are tractable. When dealing with problems that are *tractable* we can ask ourselves a similar question: which generalizations of the problem are still tractable? Since practical problems are highly complex, being able solve more general problems will often allow real-world problems to be modeled (and hence solved) more accurately. This style of research is popular in the community of polynomial-time approximation algorithms; many studies [3,4,5,6] have been undertaken to see which generalizations of well-known problems are still "tractable" to approximate, i.e. can be approximated efficiently with good bounds on the error ratio. One important type of generalization that is often relevant for combinatorial graph problems is to introduce weights for each vertex: instead of finding a subset of vertices of minimum (or maximum) cardinality that satisfies some criteria, we instead look for a subset of minimum (maximum) weight that satisfies the criteria. These weights can then be used to model costs or benefits in real-world applications. We suggest applying the practical techniques from fixed parameter complexity theory to such generalized problems.

A powerful technique in fixed parameter complexity theory is that of polynomial-time kernelization. The goal is to preprocess an instance (x, k) in time $p(|x| + k)$ for some polynomial p to obtain a *reduced* instance (x', k') that preserves the answer to the decision problem, such that $|x'|, k' \leq f(k)$ for a computable function f. A procedure that performs such preprocessing is called a *kernelization algorithm* (or *kernel*), and the function f is the *size* of the kernel. Thus kernelization can be seen as a form of preprocessing with a performance guarantee on the compression that is obtained with respect to the parameter value k. Kernelization algorithms are often valuable in practice because they can be combined with any other type of algorithm (either heuristic or exact in nature); since the kernelization step does not change the answer to the problem, it "never hurts" to start by first kernelizing the instance, and then using a heuristic approach or exact exponential-time algorithm on the reduced instance.

Given the practical importance of weighted problems and the practical relevance of kernelization algorithms, it is surprising to note that only few kernelization algorithms exist for weighted problems. The classic Vertex Cover kernelization by Buss can also be applied for Weighted Vertex Cover if all weights are at least 1, and generalizes to Weighted d-Hitting Set for fixed d. The Weighted Cluster Editing problem where each edge is given a weight has a $O(k^2)$ kernel as shown in [7]. Aside from these examples, no kernelization algorithms for weighted problems are known to us.

In this work we will study the fixed parameter tractability of a generalization of the well-known Maximum Leaf Spanning Tree problem (abbreviated as Max Leaf from now on). In the Max Leaf problem we are given an undirected, connected graph G and an integer k, and are asked whether G has a spanning tree with at least k leaves. The problem was originally proven NP-complete by Garey and Johnson, even when restricted to planar graphs of maximum degree 4. Peter Lemke [8] showed several years later that the problem remains NP-complete when restricted to d-regular graphs for any $d \geq 3$. The Max Leaf problem has been a popular topic of research from the parameterized complexity standpoint. When parameterized by the requested number of leaves k, it has been shown that the problem has a kernel with $3.75k$ vertices [9] and that there is an algorithm to solve the problem in $O(4^k k^2 + n^{O(1)})$ time [10], which was later improved to $O(3.4575^k \cdot n^{O(1)})$ [11]. Using a different type of analysis this line of research also lead to a $O(1.8966^n)$ algorithm for the classical (non-parameterized) problem [12]. Max Leaf has also been studied from the perspective of extremal graph theory [13,14,15,16]. This paper focuses on the following natural generalization:

Weighted Max Leaf
Instance: An undirected connected graph $G = (V, E)$; a weight function $w : V \mapsto \mathbb{Q}_{\geq 1}$ on the vertices; a rational number $k \geq 1$.
Question: Does G have a spanning tree with leaf set L such that $\sum_{v \in L} w(v) \geq k$?
Parameter: The value k.

Observe that this definition requires vertex weights to be rational numbers not smaller than 1. There is a good motivation for this restriction; when vertex weights are allowed to be arbitrarily small fractions then the problem is NP-complete for $k = 1$, since an unweighted graph G has a spanning tree with k leaves if and only if that same graph has a spanning tree with leaf weight 1 if we set all vertex weights to $1/k$. If the weight 0 is allowed then it can be shown by a reduction from INDEPENDENT SET that the resulting problem is hard for $W[1]$, a result which will be published elsewhere. Therefore we focus on weights that are at least 1. This may still lead to small (and hence practical) values for the parameter value k since the relative weight differences between vertices may be small, thus yielding a small parameter value for the overall target weight.

The problem of finding a spanning tree with k leaves is equivalent to finding a connected dominating set with $|V| - k$ vertices, and these problems have many applications in circuit layout [13] and network design. We now give an example of a practical application: setting up a broadcasting network. Suppose we have a set of villages, each equipped with a receiver for radio transmissions but without any transmitters. We can choose to place a transmitter at some village, and we know which villages will be within the range of this transmitter. The purpose of the broadcasting network is to send emergency messages in case of natural disasters. Since this is vital information, we require that such a message can be forwarded by transmitters so that every village can receive the message. The question we now ask ourselves is: what is the best combination of transmitter sites, such that we obtain a broadcasting network that can reach all the villages? If the "best" combination is interpreted as the combination that requires the smallest number of transmitters, then this problem can be modeled as an instance of the MAX LEAF problem: any village where we do not place a transmitter then corresponds to a leaf of the spanning tree that is found. But what if the "best" combination is the one that minimizes some cost function? Suppose that for each village we know the cost of placing a transmitter there. The costs may differ per village because of local terrain, existing infrastructure and so on. The "best" combination of transmitters is then the combination that minimizes the sum of the costs of the selected sites. In this context the best combination of transmitters can be found by weighting every vertex with the cost of placing a transmitter at the corresponding village, and finding a spanning tree with the maximum leaf weight in the resulting graph.

Our Contribution The main result of this paper is that the WEIGHTED MAX LEAF problem has a kernel with $5.5k$ vertices when every weight is a rational number not smaller than 1. The kernelization is achieved by a small set of simple reduction rules that can be applied in linear time. The reduction rules make non-trivial use of the vertex weights, thus giving an example of how kernelization can be applied to weighted problems. The existing $3.75k$ kernelization by Estivill-Castro et al. [9] does not work in the weighted case, because it relies on the fact that two adjacent degree-2 vertices can always be leaves in an optimal spanning tree if the edge between them is not a bridge. Since this no longer holds in the weighted variant of the problem, we have to devise new reduction rules.

We also show that the multiplicative constant of 5.5 in the kernel size is best-possible with respect to the given set of reduction rules, which means that our analysis of the size of the reduced instances is tight. This analysis relies on a new result in the style of extremal graph theory: we give a constructive proof that every connected undirected graph $G = (V, E)$ that avoids some simple subgraphs (see Definition 1) has a spanning tree with at least $|V|/5.5$ leaves. To prove this result we extend the technique of "amortized analysis by keeping track of dead leaves", which was originally used by Griggs et al. [14] to show that every connected cubic graph G has a spanning tree with at least $\lceil |V|/4 + 2 \rceil$ leaves.

Organization We give some preliminaries in Section 2. In Section 3 we obtain a structural result on the existence of spanning trees with many leaves in graphs that avoid some simple subgraphs. Section 4 uses this structural result to present the kernelization algorithm. We consider the optimization version of WEIGHTED MAX LEAF in Section 5, and prove that the problem is very hard to approximate.

2 Preliminaries

A graph G is a pair (V, E) where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. We also use $V(G)$ and $E(G)$ to denote the vertex and edge sets of G, respectively. We only consider simple, undirected, connected graphs. For $v \in V$ we denote the *open* neighborhood of v by $N_G(v) := \{u \in V \mid uv \in E\}$ and the *closed* neighborhood by $N_G[v] := N_G(v) \cup \{v\}$. Throughout this work we omit subscripts if this does not lead to confusion. We overload the neighborhood notation for a set $S \subseteq V$ such that $N_G(S) := \cup_{v \in S} N_G(v) \setminus S$. The degree of a vertex v in graph G is denoted by $\deg_G(v) := |N_G(v)|$. We write $G' \subseteq G$ if G' is a subgraph of G. For $X \subseteq V$ we denote by $G[X]$ the subgraph of G that is induced by the vertices in X. Noting that V is the vertex set of G we abbreviate the construction $G[V \setminus X]$ by $G - X$. A *cutset* for a connected graph G is a set $S \subseteq V$ such that $G - S$ is not connected. Vertex v is a *cut vertex* if $\{v\}$ is a cutset. To *contract* an edge uv means to replace vertices u and v by a new vertex x with $N_G(x) := N_G(\{u, v\})$.

If T is a tree subgraph of G and e is an edge with $e \in E(G)$ and $e \notin E(T)$, then we say that T *avoids* edge e. If $e \in E(G)$ and $e \in E(T)$ then tree T *uses* edge e. If $T \subseteq G$ is a tree with $V(T) = V(G)$ then T is a *spanning tree* for G. A vertex of degree 1 is called a *leaf*. If v is a vertex in a tree and v is not a leaf, then it is an *internal node* of the tree. The *leaf set* of a graph $G = (V, E)$ is the set of degree-1 vertices, denoted as $\textsc{Leaves}(G) := \{v \in V \mid \deg_G(v) = 1\}$. If we have a weight function $w : V \mapsto \mathbb{Q}_{\geq 1}$ for graph G then we can define its *leaf weight* as $\textsc{lw}_w(G) := \sum_{v \in \textsc{Leaves}(G)} w(v)$.

A *path component* in a graph G is a path P on vertices $\langle u, s_1, s_2, \ldots, s_q, v\rangle$ such that successive vertices are connected by an edge, all the vertices s_i are distinct and have degree 2, and such that $\deg(u), \deg(v) \neq 2$. Note that we explicitly allow u and v to be the same vertex. The vertices u, v are called the *endpoints* of the path component. The vertices s_i are the *inner vertices* of the path component. We define the *size of a path component* to be equal to the number q of inner vertices.

To simplify the exposition we use K_n to refer to the complete graph on n vertices, and C_n to refer to the simple cycle on n vertices. The class of d-degenerate graphs consist of all graphs for which every vertex-induced subgraph has a vertex of degree at most d. Degenerate graphs form a generalization of sparse graph classes such as planar graphs, bounded-genus graphs and H-minor free graphs. Consult Bollobas [17] for more information. The set of rational numbers not smaller than 1 is denoted by $\mathbb{Q}_{\geq 1}$. We use a simple folklore lemma regarding spanning trees that will simplify the proofs that follow later.

Lemma 1. *If $S \subseteq V$ forms a cutset for graph $G = (V, E)$ then there is no spanning tree $T \subseteq G$ in which all vertices in S are leaves.*

Proof. Proof by contradiction. Suppose S forms a cutset for G and there is a spanning tree $T \subseteq G$ in which all vertices of S are leaves.

Since S is a cutset, there are two vertices $x, y \in V(G) \setminus S$ such that there is no simple path between x and y in $G - S$. Since T is a spanning tree, there is a simple path P between x and y in tree T. Now observe that no simple path can visit a degree-1 vertex if it is not an endpoint of the path. But since all vertices in S have degree 1 in T and $x, y \notin S$ the path P cannot use any vertices in S. This implies that P is also a path connecting x and y in $G - S$, which contradicts the choice of x and y. ◻

3 Spanning Trees with Many Leaves in Graphs Without Long Path Components

In this section we prove a lower bound on the number of leaves that can be obtained in spanning trees for graphs that do not contain long path components and which avoid some simple subgraphs. Appendix (A) contains a shorter proof of a weaker form of the result presented here.

Definition 1. *Let $\mathcal{C}$ be the class of graphs $G = (V, E)$ that satisfy the following properties:*

(i) Graph G is simple and connected.
(ii) The graph is not isomorphic to a simple cycle.
(iii) The maximum size of a path component in G is at most 3.
(iv) Every vertex $v \in V$ with $\deg(v) = 1$ is adjacent to a vertex of degree at least 3.
(v) If G contains a triangle on three vertices x, y, z as a subgraph (i.e. $\{xy, xz, yz\} \subseteq E$), then at least one of the vertices x, y, z has a degree in G of at least 4.
(vi) If x, y are two distinct degree-2 vertices then $N_G(x) \neq N_G(y)$.

Theorem 1. *Every graph $G = (V, E) \in \mathcal{C}$ has a spanning tree with at least $|V|/5.5$ leaves.*

Proof. Consider $G = (V, E) \in \mathcal{C}$. Observe that connected graphs with fewer than 3 vertices do not satisfy Property (iv), so any graph $G \in \mathcal{C}$ must have $|V| \geq 3$. Our proof uses the method of "amortized analysis by keeping track of dead leaves", as introduced by Griggs et al. [14]. The proof is constructive and consists of a series of operations that can be used to initialize a tree $T \subseteq G$, and to augment T if it does not yet span G. We will prove that the resulting tree T has sufficient leaves by showing for every augmentation step that the increase in the total size of the tree is balanced against the increase in the number of leaves of the tree. To analyze the number of leaves in the resulting spanning tree we use the notion of *dead leaves.* A leaf $v \in \textsc{Leaves}(T)$ is called *dead* if all its neighbors in G are also in T. More formally, leaf v is dead iff. $N_G(v) \subseteq V(T)$. Every leaf vertex that is not dead, is alive. We define the following abbreviation for the set of live leaves:

$$\underset{G}{\textsc{LiveLeaves}}(T) := \{v \in \textsc{Leaves}(T) \mid N_G(v) \setminus V(T) \neq \emptyset\}. \tag{1}$$

If vertex $v \in V$ has a neighbor $u \in N_G(v)$ but the vertex u is not in T, then we say that v has a neighbor u outside the tree. If $u \in N_G(v)$ and $u \in V(T)$ then vertex v has a neighbor u inside the tree. A vertex $x \in N_G(V(T))$ is said to be *adjacent* to the tree T. We also need the following concept.

Definition 2. *For a tree $T \subseteq G$, we say that a vertex v is a* split vertex *if and only if v is a live leaf of degree 2, and $N_G(v) \setminus V(T) = \{u\}$ for some vertex u with $\deg_G(u) = 2$. We denote the set of all split vertices of T with respect to G by $\textsc{Split}_G(T)$.*

Our analysis uses the following properties of the tree T that is under construction:

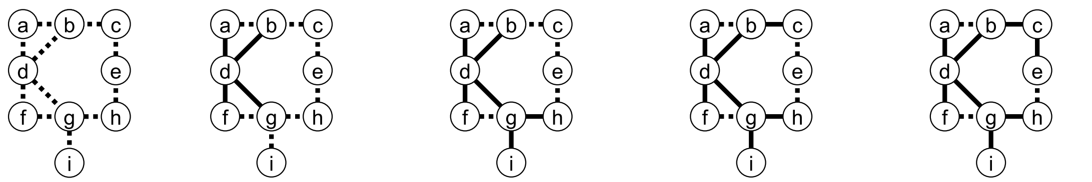
- L, the number of *leaves* of T,
- D, number of *dead* leaves of T,
- S, the number of *split vertices* of T,
- N, the total number of vertices of T.

We will give a series of augmentation operations that satisfy the following *incremental inequality*:

$$4\Delta L + 1.5\Delta D + \Delta S \geq \Delta N. \tag{2}$$

The Δ values in this inequality represent the changes in the respective quantities in the new tree compared to the old tree. For example, if the tree had 5 leaves before the augmentation operation and it has 7 leaves after the operation then $\Delta L = 7 - 5 = 2$.

Observe that in a spanning tree there can be no live leaves: all neighbors to all vertices must be in the tree. This means that all leaves must be dead and hence $L = D$. Since there are no live leaves in a spanning tree we also have $S = 0$ for such trees. It follows that if we grow a spanning tree such that every augmentation step satisfies the incremental inequality, then by summing the inequalities over the individual augmentation steps we find that the final tree satisfies $4L + 1.5D + S \geq N$; using the fact that $L = D$ and $S = 0$ on spanning trees we can then conclude that $|\textsc{Leaves}(T)| \geq |V|/5.5$ for a tree T that is grown by operations that respect the incremental inequality. So to prove the claim all that remains is to show that there exists a set of augmentation operations that can grow a spanning tree while respecting the incremental inequality.



(a) (b) $(4, 2, 0, 4)$ (c) $(1, 1, 1, 2)$ (d) $(0, 0, 1, 1)$ (e) $(0, 2, -2, 1)$

Fig. 1. This sequence shows the effect of successive vertex expansions. Dotted lines indicate edges in $E(G) \setminus E(T)$, and solid lines represent edges in $E(G) \cap E(T)$. Each expansion is labeled with the corresponding vector $(\Delta L, \Delta D, \Delta S, \Delta N)$ of changes in the measured quantities. 1(a): the graph G without a tree subgraph. 1(b): initialized T as the star around vertex d, causing a, f to become *dead* leaves and b, g to become *live* leaves. 1(c): expanded g, adding i, h to T. Vertex h is now a *split* vertex. 1(d): expanded b, causing c to become a split vertex as well. 1(e): expanded c, which causes vertex h to transform from a split vertex into a dead leaf. $C_{T \to T'} = \{h\}$ for this step.

Tree Initialization The initialization of the tree T is simple: we just pick a vertex v of maximum degree in G as the root of the tree, and we add edges to all neighbors of v to the tree. So after initialization we have a tree with one internal vertex v and leaf set $\text{LEAVES}(T) = N_G(v)$. For this operation we have $\Delta N = 1 + |N_G(v)|$ since the tree is a star rooted at v, and $\Delta L = |N_G(v)|$ because all neighbors of v have become leaves. The values ΔD and ΔS cannot be negative because there were no leaves before the tree was initialized, so no dead leaves or split vertices can be lost. With this information it can easily be seen that the initialization of the tree satisfies the incremental inequality.

Extending the Tree We need the following definition to describe the operations that augment the tree.

Definition 3 (Vertex Expansion). *Let $T \subseteq G$ be a tree. The* expansion *of a vertex $v \in V(T)$ yields an augmented tree $T' \subseteq G$, where T' is obtained by adding all edges $vu \in E(G)$ with $u \in N_G(v) \setminus V(T)$ to the tree T.*

Figure 1 shows an example of vertex expansions. It follows from the definition that the expansion of a vertex can never decrease the number of leaves in the tree. If v has a neighbor $u \in N_G(v) \setminus V(T)$, then expansion of v will cause v to become internal (decreasing the number of leaves), but it will also cause u to become a new leaf. If v has no neighbors in G that are outside T, then the expansion has no effect. The operations that we will present to augment the tree consist of series of vertex expansions. This means that $\Delta L \geq 0$ for every augmentation step. Since the expansion of a vertex can not change the fact that a leaf is dead, we also have $\Delta D \geq 0$ for all our augmentation operations. By growing the tree through expansion operations we will also maintain the invariant that for every internal vertex of T, all its neighbors in G are also in T. In other words, we will grow an *inner-maximal* spanning tree. This implies that whenever a vertex $v \in V(G) \setminus V(T)$ is adjacent to some $u \in V(T)$, then u must be a leaf of T. We will use this invariant of the tree construction process in the correctness proofs of the augmentation operations. To simplify the bookkeeping we introduce the following concept.

Definition 4. *Let $T \subseteq G$ be a tree subgraph of G, and let $T' \subseteq G$ be the tree that is obtained from T by a sequence of vertex expansions. We define the set $C_{T \to T'}$ of* converted split vertices *as the set of vertices that are split vertices in T, but dead leaves in T':*

$$C_{T \to T'} := \left\{ v \in V(G) \mid v \in \underset{G}{\text{SPLIT}}(T) \wedge v \in \underset{G}{\text{LEAVES}}(T') \wedge N_G(v) \setminus V(T') = \emptyset \right\}. \tag{3}$$

Lemma 2. *Let $T \subseteq G$ be a tree subgraph of G, and let T' be the tree that results after the successive expansion of vertices $x_1, \ldots, x_k$. Let $c := |\text{SPLIT}_G(T) \cap \{x_1, \ldots, x_k\}|$. For this operation it holds that $\Delta S \geq -|C_{T \to T'}| - c$ and $\Delta D \geq |C_{T \to T'}|$.*

Proof. Assume the conditions stated in the lemma hold. We will first prove the lower bound on ΔD. The expansion of a vertex cannot change the status of a dead leaf, since all its neighbors were already in the tree before the expansion. Therefore all the vertices that are dead leaves in T, must still be dead leaves in T'. Additionally we have by the definition of $C_{T \to T'}$ that all vertices in $C_{T \to T'}$ were not dead leaves in T, but have become dead leaves in T' which proves $\Delta D \geq |C_{T \to T'}|$.

Now we will prove the lower bound on ΔS. Consider some split vertex $v \in \textsc{Split}_G(T)$ that is not expanded, i.e. $v \notin \{x_1, \ldots, x_k\}$. We will argue that if $v \notin \textsc{Split}_G(T')$ then v is a dead leaf in T'. So assume that $v \notin \textsc{Split}_G(T')$. By the definition of a split vertex we have $N_G(v) \setminus V(T) = \{x\}$ for some vertex x with $\deg_G(x) = 2$. Every split vertex is a live leaf, and a leaf of the tree can only become internal by expanding that leaf. Since $v \notin \{x_1, \ldots, x_k\}$ we did not expand v when building T'. Therefore the vertex v must still be a leaf in T'. From the definition of a split vertex we can now deduce that the only way for v to cease being a split vertex is if it has in fact become a dead leaf, because its neighbor x was added to T' by the expansion of $x_1, \ldots, x_k$. This shows that all split vertices of T that are not expanded and which are no longer split vertices in T', must have become dead leaves and are therefore in the set $C_{T \to T'}$ of converted split vertices. Since we expand c vertices that are split vertices in T, we lose at most $|C_{T \to T'}| + c$ split vertices by the expansions from which the bound $\Delta S \geq -|C_{T \to T'}| - c$ follows. □

Order of Augmentation Operations We are now ready to present the augmentation operations. When describing an augmentation, we will always assume that no earlier augmentation is applicable. This is important because for some rules their correctness depends on the fact that certain situations are reduced by earlier augmentations. In our description of the operations we use T and T' to denote the tree before and after the augmentation, respectively.

Augmentation Operation 1 If there is a live leaf $v \in \textsc{LiveLeaves}_G(T)$ with at least 2 neighbors outside of T, then expand v.

Lemma 3. *Augmentation operation 1 satisfies the incremental inequality.*

Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. Any tree T is initialized to contain at least two vertices, hence every vertex in T has at least one neighbor inside T. This implies that if v has at least 2 neighbors outside T, then the degree of v is at least three and therefore it is not a split vertex. All neighbors of v that are not in T will have become leaves in T'. The vertex v itself is internal in T'. We now find $\Delta L = |N_G(v) \setminus V(T)| - 1 \geq 1$ and $\Delta N = |N_G(v) \setminus V(T)|$. Since we do not expand any split vertices in this operation, we find by Lemma 2 that $\Delta D \geq |C_{T \to T'}|$ and $\Delta S \geq -|C_{T \to T'}|$, which implies that this operation satisfies the incremental inequality. □

Observation 1 If augmentation operation 1 is not applicable, every live leaf $v \in \textsc{LiveLeaves}_G(T)$ has exactly one neighbor outside T.

Augmentation Operation 2 If there is a simple path $P = \langle x, y \rangle$ in G such that $\{x, y\} \cap V(T) = \{x\}$ and $|N_G(y) \setminus V(T)| \geq 2$ then expand x and y.

Lemma 4. *Augmentation operation 2 satisfies the incremental inequality.*

Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. By Observation 1 we may assume that x has y as its unique neighbor outside T. We find that $\Delta N = |N_G(y) \setminus V(T)| + 1$ and $\Delta L = |N_G(y) \setminus V(T)| - 1$. Since y must have a degree of at least 3 we find that x and y cannot be split vertices in T, and therefore $\Delta D \geq |C_{T \to T'}|$ and $\Delta S \geq -|C_{T \to T'}|$ by Lemma 2. This combination satisfies the incremental inequality since $|N_G(y) \setminus V(T)| \geq 2$. □

Augmentation Operation 3 If there is a vertex $v \in N_G(V(T))$ with at least 2 neighbors x, y inside T, then expand the neighbor x.

Lemma 5. *Augmentation operation 3 satisfies the incremental inequality.*

Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. Since x is a live leaf it must have exactly one neighbor outside T by Observation 1. Expansion of x causes x to become internal in T' (decreasing the number of leaves by one), but this is compensated by v becoming a leaf in T' which implies that $\Delta L = 0$ and $\Delta N = 1$. Vertex y must have only one neighbor outside T (again by Observation 1). We find that $N_G(y) \setminus V(T) = \{v\}$, and since v will be added to T' by the expansion of x this implies that all neighbors of y will be in T'. Because the expansion of x does not change the fact that y is a leaf we know that y must be a dead leaf in T'. To determine the values of ΔD and ΔS, we consider the local situation.

1. If $\deg_G(v) \geq 3$ then none of its neighbors can be split vertices by Definition 4, and hence no split vertices can be involved in the expansion; we have $\Delta S = 0$ and $\Delta D \geq 1$, which implies that this operation satisfies the incremental inequality.
2. If $\deg_G(v) = 2$ then x and y can be split vertices, but since no other split vertices are lost we have $\Delta S \geq -2$. Observe that v only has the vertices x, y as its neighbors, which are both in T and hence in T'. Therefore v is a dead leaf in T'. Using the fact that y is also a dead leaf in T' we find $\Delta D \geq 2$; this also satisfies the incremental inequality.

Since we assumed that v has at least 2 neighbors inside T we know that $\deg_G(v) \geq 2$ and hence the case analysis above is exhaustive. This concludes the proof. □

Observation 2 If Operations 2 and 3 are not applicable, then no vertex $v \notin V(T)$ with $\deg_G(v) \geq 3$ is adjacent to T (since such a vertex would have two neighbors inside T, two neighbors outside T, or both). All vertices outside T have at most one neighbor in T.

Augmentation Operation 4 If there is a vertex $v \in \textsc{LiveLeaves}_G(T)$ that has a degree-1 neighbor u outside of T, then expand v.

Lemma 6. *Augmentation operation 4 satisfies the incremental inequality.*

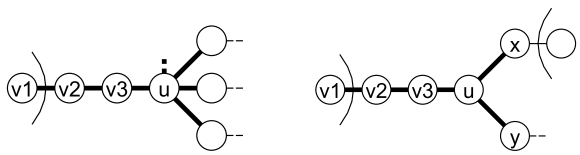
Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. Since v has a degree-1 neighbor outside of T, it is not a split vertex. By Observation 1 we may assume that $N_G(v) \setminus V(T) = \{u\}$. Since u has a degree of 1 it will be a dead leaf in T'. Therefore we find that $\Delta L = \Delta S = 0$ and $\Delta D = \Delta N = 1$ which satisfies the incremental inequality. □

Augmentation Operation 5 If there is a simple path $P = \langle x, y, z \rangle$ in G such that $\{x, y, z\} \cap V(T) = \{x\}$, $\deg_G(x) \geq 3$ and $\deg_G(y) = \deg_G(z) = 2$, then expand vertex x.

Lemma 7. *Augmentation operation 5 satisfies the incremental inequality.*

Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. By Observation 1 the vertex x has y as its unique neighbor outside T. Vertex y is a leaf in T', and in fact must be a split vertex in T' since it has a single neighbor z outside of T' and $\deg_G(y) = \deg_G(z) = 2$. Since vertex x has a degree of at least 3, it is not a split vertex by definition. We find that $\Delta L = \Delta D = 0$ and $\Delta S = \Delta N = 1$. □

Augmentation Operation 6 If there is a simple path $P = \langle x, y, z \rangle$ in G such that $\{x, y, z\} \cap V(T) = \{x\}$, vertex x is not a split vertex and $\deg_G(z) \geq 3$, then expand vertices x, y, z.



(a) (b)

Case	$\Delta N =$	$\Delta L =$	$\Delta S \geq$	$\Delta D \geq$
2(a)	$\deg_G(u) + 2$	$\deg_G(u) - 2$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert$
2(b)	5	1	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert + 1 \geq 2$

Fig. 2. This figure gives graphical representations of the augmentation operations after number 6. Each subfigure represents the structure of the boundary of the tree to which its augmentation is applicable, i.e. each subfigure shows a substructure of G and the tree T near that substructure. All augmentations start from the structure described in Lemma 10. Vertices of G that are also in T are drawn to the inside of an arc; the remaining vertices are in G, but not yet in T. Vertices incident to a dotted edge may have an arbitrary number of neighbors besides the vertices that are shown in the structure. The degrees of vertices without incident dotted edges should match their degree in the image exactly. The augmentation operation is visualized by drawing all edges that are added to T by the augmentation as thick lines. Edges of the graph that are not added to the tree by the augmentation are drawn as thin lines. This implies that all vertices that are incident to at least two thick edges were expanded during the tree augmentation. The table gives bounds on the Δ values for the relevant variables. The ΔN and ΔL values follow directly from the structure represented in the image. The bounds on ΔS and ΔD often rely on Lemma 2, noting that each augmentation expands exactly one split vertex (v_1).

Lemma 8. *Augmentation operation 6 satisfies the incremental inequality.*

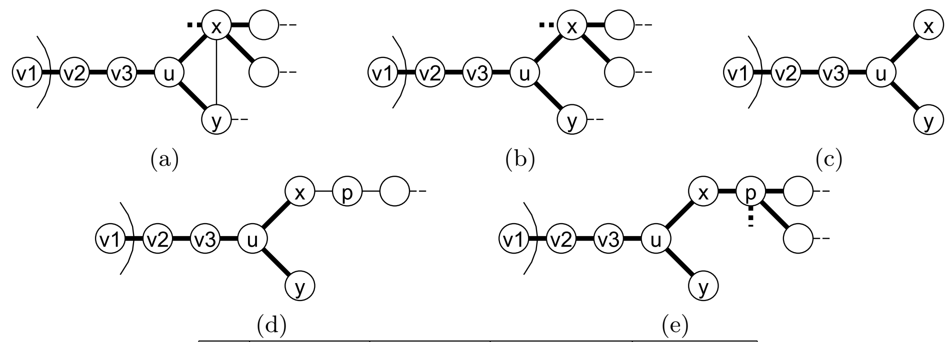
Proof. Assume the preconditions for the augmentation hold for a tree $T \subseteq G$, and let T' be the tree after the augmentation. By Observation 2 the vertex z is not adjacent to T, and the degree of vertex y must be 2. Therefore all neighbors of z except y will be leaves in T'. Since x is no split vertex by assumption, we find by Lemma 2 that $\Delta D \geq |C_{T\to T'}|$ and $\Delta S \geq -|C_{T\to T'}|$. For the remaining variables we have $\Delta N = |N_G[z]|$ and $\Delta L = |N_G[z]| - 2$ which satisfies the incremental inequality. □

Lemma 9. *If augmentation operations 1-6 cannot be applied, then it holds that every live leaf* $v \in \textsc{LiveLeaves}_G(T)$ *is a split vertex.*

Proof. Proof by contradiction. Assume there is a live leaf $v \in \textsc{LiveLeaves}_G(T)$ that is not a split vertex, and that none of the presented augmentation operations can be applied. By Observation 1 we know that v has a unique neighbor u outside T, and by Observation 2 we know that u has exactly one neighbor in T and $\deg_G(u) \leq 2$. If $\deg_G(u) = 1$ then Operation 4 is applicable; since this contradicts our assumptions, we find that $\deg_G(u) = 2$. Let $\{w\} = N_G(u) \setminus \{v\}$, and observe that $w \notin V(T)$ by Observation 2. Using the assumption that v is not a split vertex we can conclude (by the definition of a split vertex) that $\deg_G(v) \neq 2$. Since v must have at least one neighbor inside T and one neighbour outside of T (because trees are always initialized to contain at least two vertices, and v is a live leaf) we know that the degree of v must be at least 3. We now consider the situation of w.

1. If $\deg_G(w) = 2$ then Operation 5 is applicable to the path $\langle v, u, w\rangle$.
2. If $\deg_G(w) \geq 3$ then by Observation 2 the vertex w is not adjacent to T. Since v is not a split vertex by assumption, we now find that Operation 6 is applicable to the path $\langle v, u, w\rangle$.

All possibilities lead to the conclusion that an augmentation operation is applicable, which contradicts our assumptions and finishes the proof. □



Case	$\Delta N =$	$\Delta L =$	$\Delta S \geq$	$\Delta D \geq$
3(a)	$\deg_G(x) + 3$	$\deg_G(x) - 2$	$-\lvert C_{T \to T'} \rvert - 1$	$\lvert C_{T \to T'} \rvert$
3(b)	$\deg_G(x) + 4$	$\deg_G(x) - 1$	$-\lvert C_{T \to T'} \rvert - 1$	$\lvert C_{T \to T'} \rvert$
3(c)	5	1	-1	2
3(d)	5	1	0	1
3(e)	$\deg_G(p) + 5$	$\deg_G(p) - 1$	$-\lvert C_{T \to T'} \rvert - 1$	$\lvert C_{T \to T'} \rvert + 1$

Fig. 3. Tree augmentations. Refer to Figure 2 for the semantics of the images.

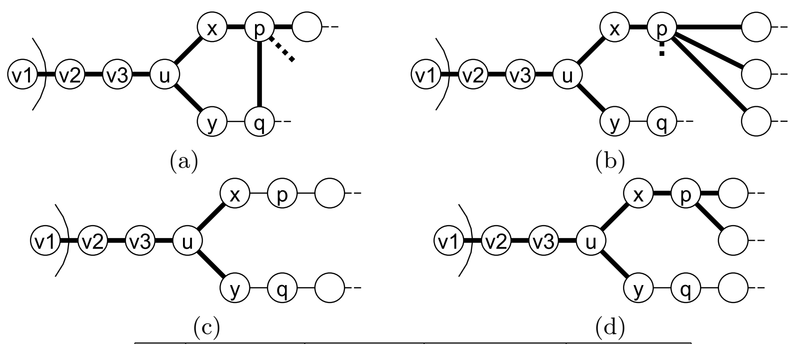
When none of the earlier operations are applicable, the boundary of the tree has a very specific structure. We will use the following lemma to capture this structure before introducing the remaining augmentation operations.

Lemma 10. *Assume none of the earlier operations are applicable and consider a live leaf* $v_1 \in \text{LiveLeaves}_G(T)$*, which must be a split vertex by Lemma 9. Let* $N_G(v_1) \setminus V(T) = \{v_2\}$*. By the definition of a split vertex we know that* $\deg_G(v_2) = 2$*. Consider the maximum path in* G *that is formed by* $P = \langle v_1, v_2, \ldots, v_q \rangle$ *where* $\deg_G(v_i) = 2$ *for all* $1 \leq i \leq q$*. Let* $\{u\} = N_G(v_q) \setminus \{v_{q-1}\}$*. The following must hold:*

1. $q \leq 3$,
2. $v_q \notin V(T)$,
3. $\deg_G(u) \geq 3$,
4. $u \notin V(T)$.

Proof. Assume the conditions in the Lemma hold. The size of a path component in G is bounded by 3 by Property (iii) of Definition 1; this establishes (1) since the v_i form a path component. If $q = 2$ then $v_q \notin V(T)$ follows from the definition of v_2. If $q = 3$ and $v_q = v_3 \in V(T)$ then the vertex v_2 has two neighbors $v_1, v_3 \in V(T)$ which implies that Operation 3 must be applicable - a contradiction. Together these statements prove (2). Since the degree of u must be unequal to 2 (by definition of the vertices v_i as a maximum path of degree-2 vertices) and since no degree-2 vertex is adjacent to a degree-1 vertex (by Property (iv) of Definition 1) we obtain (3). Because u has degree at least 3 we have by Definition 1 that u cannot be a split vertex and therefore by Lemma 9 it is not a leaf of T. Since $v_q \notin V(T)$ the vertex u cannot be an internal vertex of T since a vertex can only become internal by expanding it, and the expansion of u would have added v_q to T. Because u is not a leaf of T and not internal to T, it cannot be in T at all; this establishes (4) and finishes the proof. □

Augmentation Operations 7-26 We resume the description of the augmentation operations using the structural information of Lemma 10. The augmentation that we perform depends on the local situation of the vertex u. Since the various structural possibilities give rise to a multitude of different augmentation operations, we will not describe each of these operations individually in text; rather we will present these operations graphically to keep the proof as intuitive as possible. The remaining 20 augmentations are described by the illustrations in Figure 2-5: each subfigure corresponds to an augmentation operation. For these augmentations we do not give explicit proofs that they satisfy the incremental inequality, but instead we give bounds on the Δ values in the tables below the figures; these bounds imply that the operations satisfy the inequality. In the



Case	$\Delta N =$	$\Delta L =$	$\Delta S \geq$	$\Delta D \geq$
4(a)	$\deg_G(p) + 5$	$\deg_G(p) - 1$	$-\lvert C_{T \to T'} \rvert - 1$	$\lvert C_{T \to T'} \rvert + 1$
4(b)	$\deg_G(p) + 5$	$\deg_G(p) - 1$	$-\lvert C_{T \to T'} \rvert - 1$	$\lvert C_{T \to T'} \rvert$
4(c)	5	1	1	0
4(d)	8	2	$-\lvert C_{T \to T'} \rvert$	$\lvert C_{T \to T'} \rvert$

Fig. 4. Tree augmentations. Refer to Figure 2 for the semantics of the images.

presentation of these operations we will assume that $q = 3$. The case $q = 2$ can be handled in the same structural way, except that the increase in N is not as high as for $q = 3$. Therefore the situations of $q = 2$ satisfy the incremental inequality whenever the $q = 3$ case does. In the case analysis of the remaining situations we start from the structure of the boundary that is described in Lemma 10 and refine this structure step by step. We first deal with some easy cases.

- By the previous Lemma we have $u \notin V(T)$ and $\deg_G(u) \geq 3$, which implies $u \notin N_G(V(T))$ by Observation 2. If $\deg_G(u) \geq 4$ then expand as in Figure 2(a). Otherwise we can assume that $\deg_G(u) = 3$ in the remainder of the situations.
- Assuming $\deg_G(u) = 3$, let $\{x, y\} = N_G(u) \setminus \{v_q\}$. If one of x, y is adjacent to T then assume w.l.o.g. that this is x, and expand as in Figure 2(b); in the remainder we assume that x, y are not adjacent to T, and we may also assume that $\deg_G(x) \geq \deg_G(y)$.

We now proceed with a more careful case analysis.

1. If $\deg_G(x) \geq 3$:
   (a) If edge $xy \in E(G)$ then the vertices u, x, y form a triangle in G. By Property (v) of Definition 1, one of the vertices u, x, y must have a degree of at least 4. Since $\deg_G(u) = 3$ by the case distinctions made so far, and because of our assumption that $\deg_G(x) \geq \deg_G(y)$ we find that $\deg_G(x) \geq 4$. We proceed as in Figure 3(a).
   (b) If edge $xy \notin E(G)$ then proceed as in Figure 3(b).
2. If $\deg_G(y) = 1$, then we look at the degree of x. By the existence of the above case we may assume $\deg_G(x) \leq 2$.
   (a) If $\deg_G(x) = 1$ then proceed as in Figure 3(c).
   (b) If $\deg_G(x) = 2$ then let $N_G(x) \setminus \{u\} = \{p\}$. We consider the status of p. Observe that $\deg_G(p) > 1$ by Property (iv) of Definition 1.
      i. If $\deg_G(p) = 2$ then proceed as in Figure 3(d); note that vertex x is a split vertex after the augmentation.
      ii. If $\deg_G(p) \geq 3$ then vertex p is not adjacent to T by Observation 2. Proceed as in Figure 3(e).

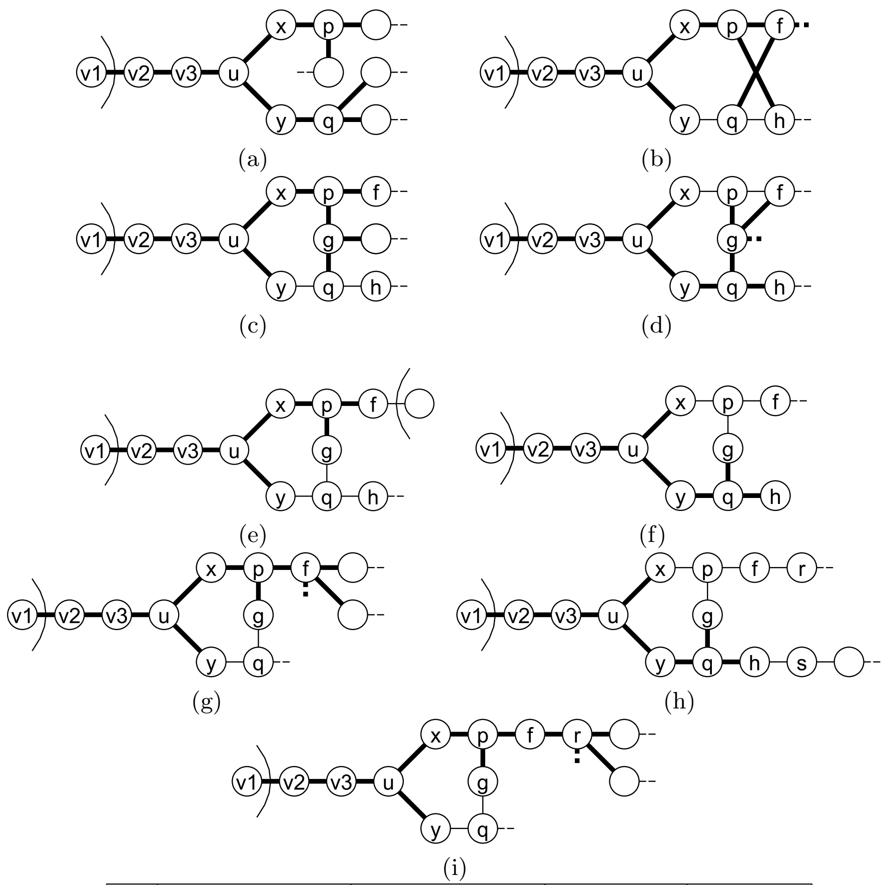
Since we assumed $\deg_G(x) \geq \deg_G(y)$ and we handled all cases with $\deg_G(y) = 1$ and $\deg_G(x) \geq 3$, we may assume in the remainder that $\deg_G(x) = \deg_G(y) = 2$. By Property (vi) we know that $N_G(x) \neq N_G(y)$ and hence that x must have some neighbor $p \neq u$, and y must have some neighbor $q \neq u$, such that $p \neq q$. Assume without loss of generality that $\deg_G(p) \geq \deg_G(q)$. We will find that the case $\deg_G(p) = \deg_G(q) = 3$ is the most complex; therefore we will first deal with the remaining cases, finishing with $\deg_G(p) = \deg_G(q) = 3$ at the end of the proof. Note that $\deg_G(p) \geq \deg_G(q) \geq 2$, since no degree-1 vertex can be adjacent to a degree-2 vertex by Property (iv).

1. If edge $pq \in E(G)$ then the degree of p must be at least 3. To see this, assume that p and q both have a degree of 2. Since they are connected to x and y respectively, which also have a degree of 2, it follows that x, p, q, y is then a path component of length at least 4. Since the length of path components in G is bounded by 3 by Property (iii) of Definition 1, this is not possible. Therefore at least one of p, q must have a degree unequal to 2. Note that p and q cannot have a degree of 1 by Property (iv), since they are adjacent to degree-2 vertices. Therefore at least one of p, q must have degree at least 3. Since we assumed $\deg_G(p) \geq \deg_G(q)$, we may conclude that $\deg_G(p) \geq 3$. We expand the tree as illustrated in Figure 4(a).
2. If edge $pq \notin E(G)$, we consider the local situation.
   (a) If $\deg_G(p) \geq 4$ then we proceed as in Figure 4(b).
   (b) If $\deg_G(q) = 2$ then we consider the degree of p. By the existence of the previous case, we may assume that $\deg_G(p) \leq 3$. Since p is adjacent to a degree-2 vertex, we know by Property (iv) that $\deg_G(p) \geq 2$.
      i. If $\deg_G(p) = 2$ then proceed as in Figure 4(c). (Note that the un-marked vertices adjacent to p and q might actually represent the same vertex, but this does not affect the augmentation in any way.)
      ii. If $\deg_G(p) = 3$ then proceed as in Figure 4(d). By Observation 2 the vertex p is not adjacent to T in this case, and hence all but one of its neighbors will become leaves. The vertex y will be a split vertex after the augmentation.

Since we assumed $\deg_G(p) \geq \deg_G(q)$, the only remaining situation is when edge $pq \notin E(G)$ and $\deg_G(p) = \deg_G(q) = 3$, which implies by Observation 2 that p and q are not adjacent to T. Note that $\deg_G(q) = 1$ is not possible by Property (iv). We now conclude with an analysis of this final case. For the analysis we consider the set $N_G(p) \cap N_G(q)$, which can be seen to have cardinality at most two by the structure of the local situation.

1. If $N_G(p) \cap N_G(q) = \emptyset$ then expand as in Figure 5(a).
2. If $N_G(p) \cap N_G(q) = \{f, h\}$ for some vertices f, h then expand as in Figure 5(b). Vertices f, h cannot be adjacent to T: if (for example) f had some neighbor in T then $\deg_G(f) \geq 3$ since f also has p, q as neighbors (which are not in T by the earlier observations), which would imply by Observation 2 that f is not adjacent to T.
3. If $N_G(p) \cap N_G(q) = \{g\}$ for a vertex g then we make a further distinction on the situation of g. Let $\{f\} = N_G(p) \setminus \{q, g\}$ and let $\{h\} = N_G(q) \setminus \{p, g\}$.
   (a) If $\deg_G(g) \geq 3$:
      i. If $N_G(g) \cap \{f, h\} = \emptyset$ then expand as in Figure 5(c).
      ii. Otherwise assume by symmetry that $f \in N_G(g)$ and expand as in Figure 5(d).
   (b) Otherwise we must have $\deg_G(g) = 2$, since $pg, qg \in E(G)$. Assume without loss of generality (by symmetry) that $\deg_G(f) \geq \deg_G(h)$.
      i. If one of f, h is adjacent to T then assume w.l.o.g that this is f and expand as in Figure 5(e).
      ii. If f, h are not adjacent to T:
         – If $\deg_G(h) = 1$ then expand as in Figure 5(f).
         – If $\deg_G(f) \geq 3$ then expand as in Figure 5(g).
         – Otherwise we must have $\deg_G(h) = \deg_G(f) = 2$, since we assumed that $\deg_G(f) \geq \deg_G(h)$. Let r be the neighbor of f that is not p, and let s be the neighbor of h that is not q. Assume w.l.o.g. that $\deg_G(r) \geq \deg_G(s)$.
            • If $\deg_G(s) = 2$ then expand as in Figure 5(h).
            • Otherwise $\deg_G(r) \geq \deg_G(s) \geq 3$; expand as in Figure 5(i). Once again we know that r is not adjacent to T by Observation 2.

Since this case analysis is exhaustive it shows that in every possible situation where T is not a spanning tree for G, there is some operation to augment T that satisfies the incremental inequality. By the earlier arguments this concludes the proof of Theorem 1. □



Case	$\Delta N =$	$\Delta L =$	$\Delta S \geq$	$\Delta D \geq$
5(a)	11	3	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert$
5(b)	$\lvert N_G(f) \setminus \{p,h\}\rvert + 8$	$\lvert N_G(f) \setminus \{p,h\}\rvert + 1$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert + 2$
5(c)	$\lvert N_G(g) \setminus \{f\}\rvert + 7$	$\lvert N_G(g) \setminus \{f\}\rvert$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert + 1$
5(d)	$\lvert N_G(g) \setminus \{h\}\rvert + 7$	$\lvert N_G(g) \setminus \{h\}\rvert$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert + 2$
5(e)	8	2	-2	2
5(f)	8	2	-1	1
5(g)	$\deg_G(f) + 7$	$\deg_G(f)$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert$
5(h)	8	2	0	0
5(i)	$\deg_G(r) + 8$	$\deg_G(r)$	$-\lvert C_{T\to T'}\rvert - 1$	$\lvert C_{T\to T'}\rvert$

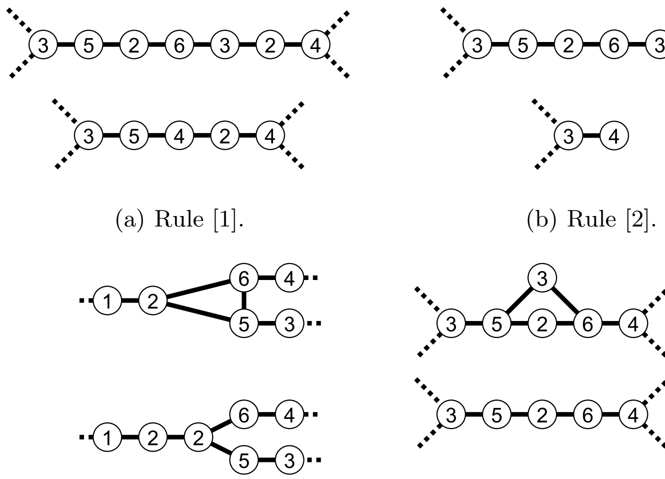
Fig. 5. Tree augmentations. Refer to Figure 2 for the semantics of the images.

4 Kernelization

In this section we describe the kernelization for Weighted Max Leaf. We first specify a set of reduction rules, then show that these can be applied efficiently and finally we analyze the resulting reduced instances.

4.1 Reduction Rules

We present reduction rules in a structured, 3-stage format. The reduction rules presented here transform an instance $\langle G, w, k\rangle$ into an instance $\langle G', w', k'\rangle$ such that G has a spanning tree with leaf weight at least k if and only if G' has a spanning tree with leaf weight at least k'. Any reduction rule that satisfies these properties is called *safe*. The reduction rules are illustrated by examples in Figure 6.



(a) Rule [1]. (b) Rule [2].

(c) Rule [3]. (d) Rule [4].

Fig. 6. Examples of the reduction rules. The reduced structure is shown below the original structure. The numbers represent vertex weights.

Reduction Rule 1 Shrink large path components

Structure: A path component $P = \langle u, s_1, s_2, \ldots, s_q, v\rangle$ of size $q \geq 4$ with endpoints u, v of degree at least 3, such that $w(s_1) \geq w(s_q)$.
Operation: Remove $s_2, \ldots, s_{q-1}$ and their incident edges from G. Add a new vertex s' with $N(s') := \{s_1, s_q\}$, and set $w'(s') := \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}) - w(s_1))$.
Justification: If a spanning tree avoids an edge with both endpoints inside the path component, it is always optimal to avoid an edge that maximizes the combined weight of its endpoints. The reduced graph offers the same connection and weighting possibilities as the original.

Lemma 11. *Rule [1] is safe:* $\langle G, w, k\rangle$ YES-*instance* $\Leftrightarrow$ $\langle G', w', k'\rangle$ YES-*instance.*

Proof. ($\Rightarrow$) Suppose G has a spanning tree T with $\textsc{lw}(T) \geq k$. Initialize T' as $T - \{s_1, \ldots, s_q\}$. Let $P' = \langle u, s_1, s', s_q, v\rangle$ be the reduced path component in G'. We show how to augment T' to a spanning tree, based on the topology of T. In cases where it is obvious that the resulting spanning tree T has sufficient leaf weight, we will not explicitly state this.

- If T uses all edges on P, then add all edges on P' to T'.
- If T avoids at least one edge on P then to be spanning it must avoid exactly one edge. If T avoids an edge e on P that is incident on an endpoint of the path component, then add all edges on P' to T' except e.
- If the edge that T avoids on P is not incident to an endpoint of the path component, then add all edges on P' to T' except for edge s_1, s'. The leaf weight achieved by T on component P is at most $\max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}))$. The leaf weight achieved by T' on component P' is $w'(s') + w'(s_1) = \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}))$, from which we can conclude $\textsc{lw}(T') \geq \textsc{lw}(T)$.

So we can convert each tree T' into a spanning tree $T \subseteq T'$ with at least as much leaf weight.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\textsc{lw}(T') \geq k'$. Initialize T as $T' - \{s_1, s', s_q\}$. Once again we show how to augment T to a spanning tree based on the topology of T'.

- If T' does not avoid an edge on the reduced path component, then add all edges on P to T.
- If T' avoids edge us_1 or $s_k v$, then add all edges on P to T except the one that is avoided by T'.

– So in the only remaining case tree T' avoids exactly one edge in the reduced path component, and it is an edge e incident on s'. Both vertices adjacent to s' have degree 2 in G', so both endpoints of e are leaves in T' and they contribute at most $w'(s') + w'(s_1)$ to $\text{LW}(T')$ (since $w(s_1) \geq w(s_q)$ by assumption, and their weights do not change). By our choice of $w'(s')$ there must be an edge $s_i s_{i+1}$ on P such that $w(s_i) + w(s_{i+1}) = w'(s') + w'(s_1)$. Let e^* be such an edge. We extend T to a spanning tree by adding all the edges on P except e^* to the tree. Now T is a spanning tree with at least as much leaf weight inside P as T' achieves inside the reduced path component. The spanning trees are equal outside of the path component under consideration, so $\text{LW}(T) \geq \text{LW}(T')$ which proves that G has a spanning tree with leaf weight at least $k' = k$.

Since this shows that the other direction of the equivalence also holds, the proof is complete. □

Reduction Rule 2 Shrink paths leading to a degree-1 vertex

> **Structure:** Path component $P = \langle u, s_1, s_2, \ldots, s_q, v\rangle$ of length $q \geq 1$ where $\deg_G(v) = 1$.
> **Operation:** Replace $s_1, \ldots, s_q$ and their incident edges by a direct edge uv.
> **Justification:** Every vertex s_i is a cut vertex and will never be a leaf in a spanning tree.

Lemma 12. *Rule [2] is safe:* $\langle G, w, k\rangle$ YES-*instance* $\Leftrightarrow$ $\langle G', w', k'\rangle$ YES-*instance.*

Proof. ($\Rightarrow$) Suppose G has a spanning tree T with $\text{LW}(T) \geq k$. Since every s_i is a cut vertex (its removal separates u from v), every s_i is internal in T according to Lemma 1. Consider the spanning tree T' for G' that is obtained from T by removing $s_1, \ldots, s_q$ and adding the direct edge uv. Since all the s_i were internal in T we must have $\text{Leaves}(T') = \text{Leaves}(T)$ and therefore $\text{LW}(T') = \text{LW}(T) = k'$, which proves that G' has a spanning tree with leaf weight at least k'.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\text{LW}(T') \geq k'$. We obtain spanning tree T for G by removing the direct edge uv, adding the vertices $s_1, \ldots, s_q$ to the graph and connecting u to v in the spanning tree through these added vertices. We have $\deg_T(u) = \deg_{T'}(u)$ and $\deg_T(v) = \deg_{T'}(v)$ which implies $\text{LW}(T') = \text{LW}(T)$, so we have proven that G has a spanning tree with leaf weight at least $k = k'$. □

Reduction Rule 3 Reduce triangles with simple neighborhoods

> **Structure:** Triangle on three vertices x, y, z such that every vertex of the triangle has at most one neighbor not in the triangle, and the graph is not isomorphic to K_3. Let x have minimum weight among $\{x, y, z\}$.
> **Operation:** Remove all the edges between vertices x, y, z. Add a new vertex m with $N(m) := \{x, y, z\}$ and set $w'(m) = w(x)$.
> **Justification:** Any spanning tree must avoid at least one edge on the triangle; the decision can be encoded using fewer high-degree vertices by adding the vertex m to represent the connectivity.

Lemma 13. *Rule [3] is safe:* $\langle G, w, k\rangle$ YES-*instance* $\Leftrightarrow$ $\langle G', w', k'\rangle$ YES-*instance.*

Proof. ($\Rightarrow$) Suppose G has a spanning tree T with $\text{LW}(T) \geq k$. We distinguish based on the status of the vertices in the triangle.

– If all vertices x, y, z are leaves in T, we build a spanning tree for G' by adding the isolated vertex m to T. We connect to m from the vertex x of minimum weight. The resulting tree T' is a spanning tree for G' with the same leaf weight as T, since the new leaf m has the same weight as x that became internal to connect to m.
– If there is at least one vertex on the triangle that is internal, then denote this vertex by v. We build a spanning tree $T' \subseteq G'$ by removing the edges from v to the other vertices on the triangle, adding the vertex m and edge vm and finally adding edges um for every $u \in N_T(v) \cap \{x, y, z\}$. Note that this construction does not change the degrees of vertices that are leaves in T; hence $\text{Leaves}(T') \subseteq \text{Leaves}(T)$ and the claim follows.

(⇐) Suppose G' has a spanning tree T' with $\textsc{lw}(T') \geq k'$. By our assumption that G is not isomorphic to K_3 we find that the set $\{x, y, z\}$ is a cutset for G' since it separates m from the vertices outside the triangle. Hence by Lemma 1 there must be at least one vertex among $\{x, y, z\}$ that is internal in T; let v be such a vertex. To build the spanning tree $T \subseteq G$ we distinguish based on the status of m in T'.

- If m is internal in T', we obtain T by removing m and its incident edges from T' and adding edges vu for every $u \in N_{T'}(m) \setminus \{v\}$. The resulting tree is a spanning tree for G with the same set of leaves.
- If m is a leaf in T' then we obtain T from T' by simply deleting m and its single incident edge. We now claim that the vertex denoted by v (which was internal in T') has become a leaf in T. To see this, observe that v must have had a degree of 2 in G'; it has one edge to m, and by the structure of the reduction rule it has at most one neighbor that is not in the set $\{x, y, z, m\}$. Since v is internal in T' (by assumption) it must have a degree of 2 in T'. By deleting m and its incident edge the degree of v becomes 1 in T, and hence it is a leaf. Since $w(v) = w'(v) \geq w'(m)$ by definition of $w'(m)$ we find that $\textsc{lw}(T) \geq \textsc{lw}(T')$. □

Reduction Rule 4 Reduce degree-2 vertices with identical neighborhoods

> **Structure:** Two vertices u, v with $w(u) \geq w(v)$ and $N_G(u) = N_G(v) = \{x, y\}$ such that $V \setminus \{u, v, x, y\} \neq \emptyset$.
> **Operation:** Remove u and its incident edges, set $k' := k - w(u)$.
> **Justification:** There is always an optimal spanning tree in which u is a leaf.

Lemma 14. *Rule [4] is safe:* $\langle G, w, k\rangle$ YES-*instance* ⇔ $\langle G', w', k'\rangle$ YES-*instance.*

Proof. (⇒) Suppose G has a spanning tree T with $\textsc{lw}(T) \geq k$. We first show that there is always an optimal spanning tree for G in which u is a leaf. Observe that u and v cannot both be internal in T, as this would imply T contains a cycle. If u is internal and v is a leaf, then we can also make v internal and u a leaf; since the weight of u is at least as much as that of v, this does not decrease the leaf weight of the spanning tree. This shows that there is always an optimal spanning tree in which u is a leaf. From such a spanning tree, we obtain a spanning tree for G' with leaf weight at least $k' = k - w(u)$ by removing u and its incident edges.

(⇐) Suppose G' has a spanning tree T' with $\textsc{lw}(T') \geq k'$. The vertices x, y from a cutset for G' since they separate v from the remainder of the graph. By Lemma 1 we know that one of the vertices x, y is internal in T'. We create a spanning tree $T \subseteq G$ by copying T', adding the vertex u and connecting to it from a vertex of x, y that is internal. Since u becomes a leaf in T we know that $\textsc{lw}(T) = \textsc{lw}(T') + w(u)$, which proves the claim. □

Although Rule [3] increases the number of vertices, it still effectively simplifies the instance. If the vertex v is in a triangle that is reduced by this rule then the neighborhood of v is simplified by the reduction: if v had degree 3 then its degree is reduced to 2, and if it had degree 2 then its degree is reduced to 1.

Since a kernelization is a self-reduction of a problem, it is important that the output instance of a kernelization algorithm satisfies the same restriction on the allowed weight range as the input problem. The following lemma will turn out to be useful to prove this.

Lemma 15. *Let* $\langle G, w, k\rangle$ *be an instance of* WEIGHTED MAX LEAF *such that Rule [1] can be applied, and let* $\langle G', w', k'\rangle$ *be the resulting instance after application of Rule [1]. Then it holds that* $\min\{w'(v) \mid v \in V(G')\} \geq \min\{w(v) \mid v \in V(G)\}$.

Proof. All vertices in G' that also exist in G have the same weight under w as under w'. The only vertex in G' that does not exist in G is the new vertex s' that is created by application of the reduction rule; we will show that its weight under w' is not less than the minimum weight of a vertex in G under w. The weight of s' is defined as $w'(s') := \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}) - w(s_1))$. Since the edge $s_1 s_2$ is considered in the maximum, it follows that $w'(s') \geq w(s_1) + w(s_2) - w(s_1) \geq w(s_2) \geq \min\{w(v) \mid v \in V(G)\}$, which proves the claim. □

4.2 Structure of a Reduced Instance

If no reduction rules are applicable to an instance, then such an instance is called *reduced*. The structure of reduced instances is captured by the class $\mathcal{C}$ (see Definition 1), which is proven in the following theorem.

Theorem 2. *If $\langle G', w', k' \rangle$ is a reduced instance of* WEIGHTED MAX LEAF *that is not isomorphic to a simple cycle and $|V(G')| \geq 3$, then $G' \in \mathcal{C}$.*

Proof. Suppose $\langle G', w', k' \rangle$ is a reduced instance with $|V(G')| \geq 3$ that is not isomorphic to a simple cycle. We will prove that G' satisfies all properties of Definition 1, in the order in which the properties are listed in the definition.

- (i) The input graph is simple and connected by definition. It is easy to verify that the reduction rules preserve these properties.
- (ii) By assumption the reduced graph G' is not isomorphic to a simple cycle.
- (iii) By Rule [1] the reduced graph G' does not have path components of length larger than 3.
- (iv) By Rule [2] the reduced graph G' does not have degree-1 vertices that are adjacent to degree-2 vertices. If the neighbor of a degree-1 vertex also has degree-1 then the graph is isomorphic to K_2 and hence $|V(G')| < 3$; otherwise every degree-1 vertex must be adjacent to a vertex of degree at least 3.
- (v) Suppose the reduced graph has a triangle x, y, z such that each vertex on the triangle has degree at most 3. If G' is isomorphic to K_3 then it is isomorphic to a simple cycle, which contradicts the assumption in the theorem. But if G' is not isomorphic to K_3 then Rule [3] is applicable, which is also a contradiction. So the reduced graph cannot have a triangle on vertices of degree at most 3.
- (vi) Suppose there are two degree-2 vertices u, v such that $N_G(u) = N_G(v) = \{x, y\}$. If $V \setminus \{u, v, x, y\} \neq \emptyset$ then Rule [4] is applicable; on the other hand if $V = \{u, v, x, y\}$ then either G' is a simple cycle on 4 vertices (which contradicts our assumption) or G' is isomorphic to K_4 minus one edge, which must contain a triangle to which Rule [3] is applicable.

Since G' satisfies all the required properties we may conclude that $G' \in \mathcal{C}$. □

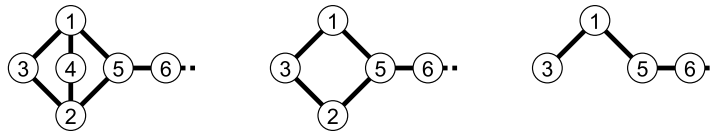
4.3 Reduction Algorithm

In this section we consider how the reduction rules can be realized by an algorithm. It is easy to verify that we can test in polynomial time whether a reduction rule is applicable to the graph, and to apply the reduction if necessary. All reduction rules except Rule [3] reduce the number of vertices, and for Rule [3] it can be verified that once the rule has been applied to a triangle, it will never be applied to any vertex of that triangle again. These easy observations show that we can obtain a reduced instance in polynomial time. By a more careful and somewhat lengthy analysis it is actually possible to show that a reduced instance can be computed in $O(|V| + |E|)$ time, which is the subject of the next pages.

Theorem 3. *Given an instance $\langle G, w, k \rangle$ of* WEIGHTED MAX LEAF *with $G = (V, E)$ encoded as an adjacency-list, it is possible to exhaustively apply the reduction rules in $O(|V| + |E|)$ time to obtain an equivalent* reduced *instance $\langle G', w', k' \rangle$.*

Proof. We will give a high-level overview of a linear-time algorithm to compute a reduced instance. We assume that the adjacency-list datastructure uses cross-pointers between the edge entries to allow a vertex v to be deleted in $\deg(v)$ time. We also assume that the weight of a vertex is stored as a field in the vertex object.

The main idea behind the reduction algorithm is to process one reduction rule at a time: take a reduction rule and exhaustively apply it to the graph wherever possible, before continuing to the next reduction rule. If the order in which we process the reduction rules ensures that a reduction will never introduce a structure that needs to be reduced by an earlier rule in the ordering, then



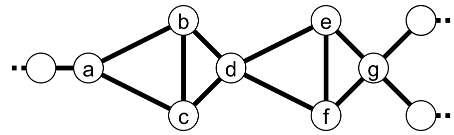
(a) Original graph. (b) After application of Rule [4]. (c) After another application of Rule [4].

Fig. 7. Example of an application of Rule [4] creating another structure that has to be reduced by the same rule. The drawn numbers represent vertex weights. Initially the vertices with weight 1 and 2 have degree 3, so that Rule [4] does not require them to be deleted. But after the required deletion of the vertex of weight 4, the vertices of weight 1 and 2 become degree-2 vertices with identical neighborhoods; the one with highest weight must now be deleted.

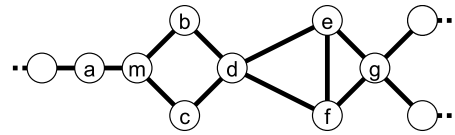
this strategy will clearly work. Unfortunately it seems to be impossible to devise an ordering that enforces this. Nevertheless we will use this as the main structure for the algorithm; we show that we can easily detect when reducible structures are created by other reductions, which allows us to ensure that we still find all reducible structures in the graph. The algorithm consists of one phase for each reduction rule. As an invariant we use that after the phase corresponding to reduction rule R, the graph is not reducible by rule R or by any rules of earlier phases. We now present the phases of the algorithm.

Rule [4] Given a vertex $s \in V(G)$ we can apply Rule [4] to all pairs u, v for which there is a vertex $t \in V(G) \setminus \{u, v\}$ such that $N_G(u) = N_G(v) = \{s, t\}$, in $O(\deg_G(s))$ time. This works as follows. We give every vertex object two new fields testSource and minWeight. Given the vertex s we loop over all its neighbors $u \in N_G(s)$. If $N_G(u) = \{s, t\}$ for some vertex t, then we determine whether this is the first degree-2 vertex with $\{s, t\}$ as its neighborhood, by checking the testSource variable. If t.testSource is not set to s then we have not encountered a degree-2 vertex with this neighborhood before; we set t.testSource to s and t.minWeight to u. If t.testSource was already set to s then we had already encountered the degree-2 vertex t.minWeight with the same neighborhood; now we compare the weight of t.minWeight to the weight of u to determine which vertex to delete. We delete the appropriate vertex and decrease k accordingly. The deletion takes constant time since the deleted vertex has a degree of 2. Therefore this test runs in $O(\deg_G(s))$ time.
Using this test we can exhaustively apply Rule [4] by running the test for each vertex $v \in V(G)$ (skipping vertices that are already deleted during previous tests, of course); this will ensure that we find and reduce all reducible structures that were present in the original input graph. But what if the deletion of some vertex triggers a new reducible structure? To ensure that we do not miss these reducible structures we use Lemma 16 which shows that for every vertex we delete, it is possible to test in constant time whether this introduces a new reducible structure. Combining these ideas yields an algorithm that takes $\sum_{v \in V(G)} O(\deg_G(v))$ time for testing the neighborhood of each vertex, and additionally a constant amount of time per reducible structure that is triggered by the deletion of some other vertex. This sums up to $O(|V| + |E|)$ time for this phase.

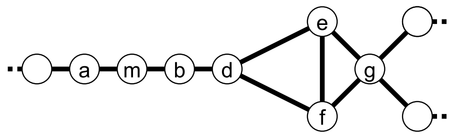
Rule [3] The crucial observation for the implementation of this rule is that reducible triangles consist of three vertices of maximum degree 3. This implies that for a given vertex $v \in V(G)$ we can test in constant time whether it is part of a reducible triangle; we first test if the degree of v is two or three, and if so we check if there is a triangle among the neighbors of degree at most 3. This test runs in constant time because the vertices involved have bounded degree. The bound on the degrees of the involved vertices also implies that we can insert the new vertex m and update the other edges in constant time.
We exhaustively apply this rule by successively testing for each vertex whether it is in a reducible triangle, and performing the reduction if necessary. Observe that an application of Rule [3] to a triangle $\{x, y, z\}$ cannot cause a different triangle in the graph to become reducible



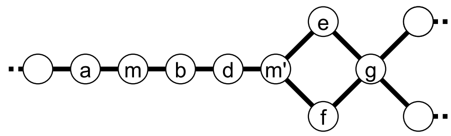
(a) Input graph; Rule [3] applies to triangle $\{a, b, c\}$ but not to $\{b, c, d\}$ or $\{d, e, f\}$ because d has two neighbors outside those triangles.



(b) The situation after reducing triangle $\{a, b, c\}$; Rule [4] is now applicable. Assume $w(b) < w(c)$, which implies that c must be deleted.



(c) After applying Rule [4] vertex d has only one neighbor outside triangle $\{d, e, f\}$ so Rule [3] can be applied.



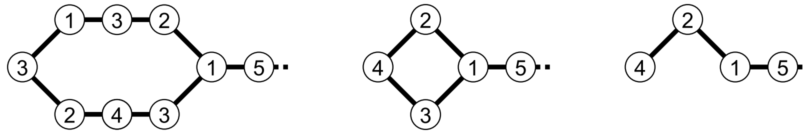
(d) The situation after reducing triangle $\{d, e, f\}$; once again Rule [4] is applicable.

Fig. 8. Situation in which Rule [3] triggers Rule [4], which in turn triggers Rule [3].

if it was not reducible before, because the rule only changes the neighborhoods of the vertices $\{x, y, z\}$ (which cease being a triangle) and does not affect the other vertices in any way. Unfortunately there are two other subtle issues here: the reduction of a triangle may create structures that have to be reduced by Rule [4], and the application of that rule may in turn trigger Rule [3] on a structure to which it was previously not applicable. An example of such a cascade is shown in Figure 8. Lemma 17 shows that a structure reducible by Rule [4] can only be created among the vertices $\{x, y, z\}$ that form the reducible triangle; therefore we can detect such a structure in constant time when we reduce the triangle, by pairwise comparing the constant-sized neighborhoods of x, y and z - we apply Rule [4] if needed. As in the previous phase we test whether Rule [4] triggers itself on a new structure. If Rule [4] deletes a vertex u with $N_G(u) = \{x, y\}$ and thereby causes Rule [3] to become applicable where it previously was not applicable, then one of x, y must be contained in the reducible triangle. We can test in constant time whether this is the case, and reduce if necessary. This ensures that after executing the steps in this phase we obtain an instance to which Rule [3] and Rule [4] are not applicable.

Now let us consider the running time for this phase. We need constant time per vertex (to check its degree, and to check whether its neighbors form a reducible triangle) plus constant time per application of Rule [3] or Rule [4]. Since every application of Rule [4] deletes a vertex, the total number of applications is bounded by the initial number of vertices plus the number of vertices created by Rule [3]; observe that every application of that rule creates exactly one new vertex. Lemma 18 shows that the number of applications of Rule [3] is bounded linearly by the number of vertices that are in the graph at the start of this phase; once Rule [3] has been applied to a triangle $\{x, y, z\}$ (creating a new vertex m), the rule will not become applicable again to one of the vertices $\{x, y, z, m\}$. The number of vertices in the graph at the beginning of this phase is not higher than the number of vertices in the input to the reduction algorithm, which allows us to conclude that this phase can be executed in $O(|V| + |E|)$ time.

Rule [1] It is straight-forward to exhaustively apply this reduction rule to the graph by performing a depth-first search starting in a vertex of degree unequal to 2, using some state information to keep track of the length of the currently traversed path component. Since this rule decreases the length of path components down to 3, it does not introduce triangles that have to be reduced by Rule [3]. Since an application of Rule [1] only affects the neighborhoods of inner vertices of the path component, it can only trigger Rule [4] if the first and last inner vertex of the path component have identical neighborhoods after the reduction - an example of this is shown in Figure 9. This situation can be recognized and further reduced in constant



(a) Original graph to which Rule [1] applies.

(b) After the application of Rule [1].

(c) Structure after reduction by Rule [4].

Fig. 9. Situation in which Rule [1] triggers Rule [4], which in turn triggers Rule [2]. The numbers represent vertex weights.

time, which implies that we can execute this phase in $O(|V| + |E|)$ time to obtain a graph to which Rule [1], Rule [3] and Rule [4] are not applicable.

Rule [2] Similarly to the previous approach we can exhaustively apply Rule [2] by performing a depth-first traversal through the graph starting from some vertex of degree unequal to 2. When path components are encountered that end in a degree-1 vertex we can immediately shrink these components. Such a traversal trivially takes $O(|V| + |E|)$ time. At the beginning of this phase the graph cannot be reduced by Rule [1], Rule [3] or Rule [4] and it can be verified that an application of Rule [2] preserves this property. Since an application of Rule [2] does not trigger itself on new structures, it is guaranteed that after a depth-first traversal of the entire graph we have handled all structures that are reducible by Rule [2]. This shows that at the end of this phase the instance is not reducible by any of the reduction rules. Since the instance was obtained by repeated application of safe reduction rules it must be equivalent to the input instance, and hence we can output the obtained reduced instance as $\langle G', w', k' \rangle$.

We have shown that we can compute a reduced instance in a constant number of phases that each take $O(|V| + |E|)$ time, which concludes the proof. □

Lemma 16. *Let $\langle G, w, k \rangle$ be an instance of* Weighted Max Leaf *to which Rule [4] is applicable because there are distinct degree-2 vertices u, v with $N_G(u) = N_G(v) = \{x, y\}$ and $w(u) \geq w(v)$. It can be detected in constant time whether the application of Rule [4] to this structure creates a new structure that can be reduced by Rule [4], i.e. whether the rule creates a reducible structure that was not reducible in G.*

Proof. Let $\langle G, w, k \rangle$ be an instance of Weighted Max Leaf that contains two degree-2 vertices u, v with $N_G(u) = N_G(v) = \{x, y\}$. Let $\langle G', w', k' \rangle$ be the the instance obtained by application of the rule; by definition G' is obtained from G by deleting vertex u. The only way in which deletion of u can cause Rule [4] to become applicable to vertices in G' to which it was not applicable in G, is when the degree of one of the neighbors x, y of u is decreased to 2. So suppose that one of x, y gets a degree of 2 by deleting u; assume without loss of generality that this is x. If x becomes part of a reducible structure, then there must be some other vertex $x' \in V(G')$ such that $N_{G'}(x) = N_{G'}(x')$. But since $N_G(v) = N_{G'}(v) = \{x, y\}$ the only vertex that can have the same neighborhood in G' as x is the vertex y, since no other vertex has v as its neighbor. Therefore the deletion of u can only trigger a new occurrence of Rule [4] if afterwards it holds that $N_{G'}(x) = N_{G'}(y)$ and x, y have degree 2. When we apply the reduction we have access to x and y, so we can test in constant time whether the reduction creates a new reducible structure. □

Lemma 17. *Let $\langle G, w, k \rangle$ be an instance of* Weighted Max Leaf *containing a triangle on vertices $\{x, y, z\}$ to which Rule [3] is applicable, such that Rule [4] is not applicable to $\langle G, w, k \rangle$. Let $\langle G', w', k' \rangle$ be the instance obtained by applying Rule [3]. If $\langle G', w', k' \rangle$ contains two distinct vertices u, v such that $N_{G'}(u') = N_{G'}(v') = \{x', y'\}$ for distinct vertices x', y' then it must hold that $\{u', v'\} \subseteq \{x, y, z\}$.*

Proof. Assume the conditions stated in the lemma hold. Observe that the application of Rule [3] does not change the neighborhoods of vertices in $V(G) \setminus \{x, y, z\}$, and that the newly introduced vertex m has a degree of 3. By the assumption that Rule [4] is not applicable to $\langle G, w, k\rangle$ we know that G contains no two degree-2 vertices with identical neighborhoods; therefore if G' contains such a structure then one of the vertices involved is one of x, y, z. But since x, y, z have m as a neighbor, and none of the other vertices in G' have m as a neighbor, it follows that if G' contains two degree-2 vertices with identical neighborhoods then this must be two vertices from the set $\{x, y, z\}$. ◻

Lemma 18. *Let $\langle G, w, k\rangle$ be an instance of* Weighted Max Leaf *that contains a triangle on the vertices $\{x, y, z\} \subseteq V(G)$ to which Rule [3] is applicable. Let $\langle G', w', k'\rangle$ be the instance after the reduction, and let m be the new vertex that was created by the reduction. Let $\langle G'', w'', k''\rangle$ be an instance obtained from $\langle G', w', k'\rangle$ by applying Rule [3] or Rule [4] zero or more times. Then Rule [3] is not applicable in G'' to a triangle that contains one or more vertices from the set $\{x, y, z, m\}$.*

Proof. Let $\langle G', w', k'\rangle$ and $\{x, y, z, m\}$ be as described in the lemma. Since $\langle G', w', k'\rangle$ was obtained by applying Rule [3] we know that $N_{G'}(m) = \{x, y, z\}$ and that none of the direct edges between x, y, z are present in G', which implies that the vertices $\{x, y, z, m\}$ are not contained in a triangle in G'. By the preconditions for Rule [3] we know that the vertices x, y, z have degree 1 or 2 in G'. Now note that Rule [3] only affects the neighborhoods of vertices in a triangle. Since x, y, z, m are not in a triangle, their neighborhoods cannot be affected by Rule [3] and hence no edges are added that can cause them to start forming a triangle. Since Rule [4] does not add any edges to the graph, this rule does not cause $\{x, y, z, m\}$ to become part of a triangle either. From this we can conclude that zero or more applications of Rule [3] or Rule [4] cannot cause $\{x, y, z, m\}$ to become part of a triangle, which implies Rule [3] is not applicable to them in G''. ◻

4.4 Kernelization Theorem

Theorem 4. *The* Weighted Max Leaf *problem has a kernel with at most $5.5k$ vertices: there is an algorithm that takes an instance $\langle G, w, k\rangle$ of* Weighted Max Leaf *as input, and computes an equivalent instance $\langle G', w', k'\rangle$ such that $|V(G')| \leq 5.5k$ and $k' \leq k$ in linear time.*

Proof. We sketch the kernelization procedure. When supplied with an input $\langle G, w, k\rangle$ the algorithm first computes a reduced instance $\langle G', w', k'\rangle$ in linear time using Theorem 3. The safety of the reduction rules ensures that $\langle G, w, k\rangle$ is a yes-instance if and only if $\langle G', w', k'\rangle$ is a yes-instance. By the definitions of the reduction rules we know that $k' \leq k$. If the graph G' is isomorphic to a simple cycle then the problem can be decided in linear time: the maximum leaf weight that can be obtained by a spanning tree in a simple cycle is equal to the maximum over all edges of the combined weight of the edge endpoints. So when G' is isomorphic to a simple cycle we can decide the problem and output a trivial 1-vertex instance that yields the same answer.

Assume from now on that G' is not a simple cycle. If $k' \leq 1$ then the algorithm outputs a trivial yes instance. If $|V(G')| \geq 5.5k' \geq 5.5$ then the answer to the decision problem must be yes: since G' is not isomorphic to a cycle Theorem 2 shows that $G' \in \mathcal{C}$, which implies by Theorem 1 that there is a spanning tree $T' \subseteq G'$ with at least k' leaves. Noting that every vertex has a weight not smaller than 1, the leaf weight of T' must be at least k' which implies that $\langle G', w', k'\rangle$ is indeed a yes instance. We now output a trivial 1-vertex instance that yields this answer. If $|V(G')| < 5.5k'$ then the size of the output graph is bounded as required since $k' \leq k$, and we output $\langle G', w', k'\rangle$. Lemma 19 shows that a reduced instance is a valid input for Weighted Max Leaf, which verifies that the kernelization is indeed a self-reduction and concludes the proof. ◻

Lemma 19. *Let $\langle G', w', k'\rangle$ be obtained by applying reduction rules to an instance $\langle G, w, k\rangle$ of* Weighted Max Leaf. *If $k' \geq 1$ then $\langle G', w', k'\rangle$ is also a valid instance of* Weighted Max Leaf*: the graph G' is simple and connected and the weight of every vertex is not smaller than 1.*

Proof. By the specification of Weighted Max Leaf the input graph G must be simple and connected in a valid instance. It is easy to verify that the reduction rules preserve these properties. Therefore it only remains to show that the weights assigned to vertices by the new weight function

w' are all at least 1. The only reduction rule that changes vertex weights is Rule [1]. Lemma 15 proves that the minimum vertex weight is not decreased by an application of that rule, which shows that the vertex weights under w' are not smaller than the vertex weights under w. Since $\langle G, w, k\rangle$ is a valid instance with weights not smaller than 1, this proves the claim. □

5 Hardness of Approximation

Although the focus of this work is on parameterized complexity, we briefly change the perspective in this section and consider Weighted Max Leaf as an optimization problem. In this case an instance consists only of a weighted graph, and the goal is to find a spanning tree with maximum leaf weight. This point of view allows us to study the approximability of the problem, and contrasts the parameterized view of Weighted Max Leaf which treats it as a decision problem. It is not hard to see that the optimization problem related to Weighted Max Leaf is in fact an NP-optimization problem since the value of a potential solution can be computed in polynomial time. There is an extensive literature [18] devoted to NP-optimization problems and reductions between them, with various notions of approximation-preserving reductions. To simplify the exposition we will *not* use the rigorous formal framework of NP-optimization problems, but rather present the results in an informal manner. It is easy to verify that the claims made here can be formalized in a straight-forward way.

Consider a NP-maximization problem Π, and let f be a function that maps instances x of Π to non-negative numbers. The function f will often only depend on the optimum value of x and on the size of x. We say that an algorithm A is a polynomial time f-approximation algorithm for Π if algorithm A always computes a feasible solution in polynomial time, such that value v of the resulting solution satisfies $v \geq \text{OPT}/f(x)$, which means that the value that is found is at most a factor $f(x)$ smaller than the optimum. When f is a constant function then this yields a *constant-factor* approximation algorithm, but we may also allow f to be a function that depends on the instance size (for example a $\frac{n}{\log n}$-approximation), or a function that depends on the optimum value (e.g. a $\sqrt{\text{OPT}}$-approximation). If Π has a f-approximation and $f \in O(g(x))$ then we say that Π has a $O(g(x))$-approximation.

In this section we will show that Weighted Max Leaf is hard to approximate, using a reduction from Independent Set. The approximability of Independent Set has been studied intensively, leading to the following result:

Proposition 1 ([19,20]). *The* Independent Set *problem has no polynomial time $O(n^{1-\epsilon})$-approximation algorithm for any $\epsilon > 0$ unless $P = NP$.*

Recall that an NP-optimization problem is *polynomially bounded* if the optimum solution value is bounded by a polynomial in the instance size. In some settings it is harder to approximate a problem if it is not polynomially bounded. We will prove that Weighted Max Leaf is hard to approximate, even when its optimum is polynomially bounded and when the input graphs are required to be 2-degenerate (see Section 2). For ease of notation we define the problem *Pb*-Weighted Max Leaf as the restriction of Weighted Max Leaf where the graph G is 2-degenerate and the weights are positive integers not exceeding $|V(G)|^2$. It is easy to verify that under this definition, *Pb*-Weighted Max Leaf is a polynomially bounded optimization problem.

Theorem 5. *We can use an approximation algorithm for* Pb-Weighted Max Leaf *to approximate* Independent Set.

1. *If* Pb-Weighted Max Leaf *has a polynomial-time $O(n^c)$-approximation algorithm then* Independent Set *has a $O(n^{2c})$ approximation algorithm.*
2. *If* Pb-Weighted Max Leaf *has a polynomial-time $O(\text{OPT}^c)$-approximation algorithm then* Independent Set *has a $O(n^{3c})$ approximation algorithm.*

Proof. The proof is based on a reduction from Independent Set to *Pb*-Weighted Max Leaf that preserves approximation properties. We will show that using this reduction, we can use the existence of approximation algorithms for *Pb*-Weighted Max Leaf to construct approximation algorithms for Independent Set.

Consider an instance G of Independent Set. We will give an approximation preserving reduction that transforms G into an instance $\langle G', w_{G'} \rangle$ of *Pb*-Weighted Max Leaf. For the sake of analysis we will partition the vertices of the resulting graph G' into two types: *heavy* vertices that correspond to vertices in graph G, and *light* vertices that are added by the reduction. The reduction proceeds as follows.

1. Initialize G as a copy of G'. All vertices of G' at this stage are *heavy*.
2. Replace every edge uv between heavy vertices by a new light vertex x with edges $\{ux, vx\}$.
3. Add a new light root vertex r and give r edges to all heavy vertices.

The graph that results from the above three steps is used as G'. For ease of notation define $n := |V(G)|$ and $n' := |V(G')|$. The weight function is simple, and corresponds to the intuition. We set the weight of all heavy vertices to n^2, and we set the weight of all light vertices to 1. Figure 10 shows an example of this reduction.

From the definition of the graph G' it follows that $n' = |V(G)| + |E(G)| + 1 \leq n + \binom{n}{2} + 1$ which implies that $n' \leq n^2$ for $n \geq 2$. Since we may assume without loss of generality that $n \geq 2$ (the $n = 1$ case can be solved trivially in polynomial time) we have $n' \leq n^2$. Using the fact that the weight of a vertex in G' is at most $n^2 < (n')^2$ it easily follows that the vertex weights satisfy the restrictions placed on the problem *Pb*-Weighted Max Leaf. It is not hard to verify that G' is a 2-degenerate graph. Any vertex-induced subgraph that has minimum degree 3 or higher cannot contain any of the light vertices that were used to subdivide edges. But if all such light subdivider vertices are excluded from a vertex-induced subgraph, then such a subgraph must in fact be a star and hence it contains a vertex of degree 1. Therefore all vertex-induced subgraphs of G' have minimum degree at most 2 and hence G' is 2-degenerate.

The following lemma shows the relationship between solutions of the Independent Set instance G and the *Pb*-Weighted Max Leaf instance $\langle G', w_{G'} \rangle$.

Lemma 20. *Instance G has an independent set S of size at least k $\Leftrightarrow$ instance $\langle G', w_{G'} \rangle$ has a spanning tree $T' \subseteq G'$ with at least k heavy leaves.*

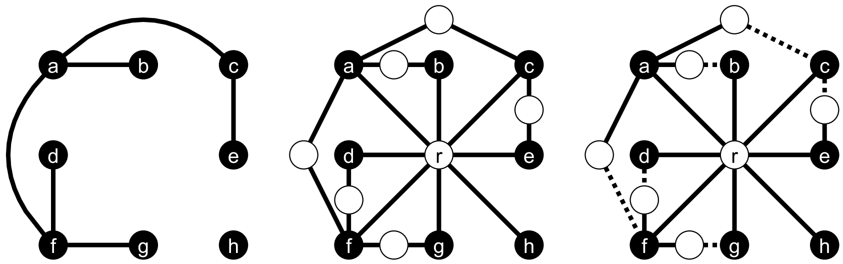
Proof. We shall use the fact that the vertex set of G corresponds to the heavy vertices in G'.

($\Rightarrow$) Suppose S is an independent set for G of size $|S| \geq k$. We will show that G' has a spanning tree with at least $|S|$ heavy leaves. We initialize a tree $T' \subseteq G'$ by taking the light vertex r as the root, and adding edges to every heavy vertex in G'. Every heavy leaf in T' now corresponds to a vertex of G. The only vertices that this tree does not reach are the light vertices that were added when subdividing edges of G. Since S is an independent set, we know $V(G) \setminus S$ forms a vertex cover for G. By the equivalence between the light subdivider vertices in G' and edges in G, we can augment tree T to a spanning tree by connecting to the light subdivider vertices from the vertices in $V(G) \setminus S$. If we augment T' to a spanning tree in this way, every heavy vertex in G' that is a member of the independent set S in G remains a leaf. So the augmented T' is a spanning tree for G' with at least $|S| \geq k$ heavy leaves.

($\Leftarrow$) Suppose $T' \subseteq G'$ is a spanning tree for G' and S are its heavy leaves with $|S| \geq k$. By the correspondence between heavy vertices in G' and vertices of G, we can also interpret S as a subset of the vertices of G. We will prove that S forms an independent set in G, by showing that if there is an edge uv in G, then u and v are not both heavy leaves in any spanning tree for G'.

So assume G has an edge uv. By the definition of the reduction, this edge was subdivided by some light vertex x when forming G'. Since $N_{G'}(x) = \{u, v\}$ it follows that $\{u, v\}$ is a cutset for G' since it separates w from the light root vertex r. By Lemma 1 we may conclude that at least one of u and v is not a leaf in a spanning tree for G'.

Using this fact we obtain by contraposition that the set S of heavy leaves is an independent set in G, and hence this establishes that G has an independent set of size $|S| \geq k$. ◻



(a) Graph G. (b) Resulting $\langle G', w_{G'}\rangle$. (c) Tree T'.

Fig. 10. Example of the reduction from Independent Set to *Pb*-Weighted Max Leaf. (a) An instance G of Independent Set. (b) The corresponding instance $\langle G', w_{G'}\rangle$ of *Pb*-Weighted Max Leaf that results from the reduction. Heavy vertices with weight n^2 are drawn in black, and light vertices with weight 1 are drawn in white. (c) The spanning tree $T' \subseteq G'$ that corresponds to the independent set $\{b, c, d, g, h\}$ in G. Thick lines represent edges in $E(G') \cap E(T')$ and broken lines represent edges in $E(G') \setminus E(T')$.

The correspondence between the two instances allows us to use an approximation algorithm for *Pb*-Weighted Max Leaf to build an approximation algorithm for Independent Set. Suppose we have a polynomial time $O(n^c)$-approximation (or $O(\text{OPT}^c)$ approximation) algorithm A for *Pb*-Weighted Max Leaf. We construct a polynomial time $O(n^{2c})$-approximation (resp. $O(n^{3c})$-approximation) algorithm for Independent Set by applying the reduction from Independent Set to *Pb*-Weighted Max Leaf and running algorithm A on the instance $\langle G', w_{G'}\rangle$. Let $T' \subseteq G'$ denote the spanning tree that is found. By Lemma 20 the heavy leaves of spanning tree T' form an independent set in G; we use this independent set as the output of algorithm B. Since the reduction can be applied in polynomial time, it follows that B runs in polynomial time if the algorithm A exists. The vertex weights that are used in the reduction allow us to easily establish a relationship between the leaf weight of a spanning tree, and the number of heavy leaves that it contains. The total weight of the light vertices is less than n^2, since each light vertex has weight 1 and there at most $\binom{n}{2} + 1$ of such vertices. We can consider the leaf weight of a spanning tree to consist of two parts: the weight of heavy leaves plus the weight of light leaves. Consider the spanning tree $T' \subseteq G'$ that is found by running algorithm A on the instance resulting from the reduction, and let k' denote the number of *heavy* leaves in T'. Since light leaves contribute at most n^2 to the total leaf weight, and because the weight of a heavy vertex is exactly n^2, we find the following:

$$k' \geq \frac{1}{n^2}(\underset{w_{G'}}{\text{LW}}(T') - n^2) \geq \frac{1}{n^2}\underset{w_{G'}}{\text{LW}}(T') - 1. \tag{4}$$

It remains to prove the approximation guarantee for algorithm B. The following observation will be important. Let k be the size of a maximum independent set in G, and let $S \subseteq V(G)$ be an independent set in G of size k. By the correspondence between G and the instance $\langle G', w_{G'}\rangle$ that was derived in Lemma 20 we know that the vertices in S correspond to heavy vertices in G', and that there is a spanning tree $T'' \subseteq G'$ where all the vertices in S are heavy leaves; hence such a spanning tree has leaf weight at least $|S|n^2 = kn^2$. Let $T^* \subseteq G'$ be a spanning tree for G' with maximum leaf weight. Since the leaf weight of T^* must be at least as large as the leaf weight of T'', we find:

$$\underset{w_{G'}}{\text{LW}}(T^*) \geq kn^2. \tag{5}$$

Using these ingredients are are now ready to prove the approximation guarantees for the resulting algorithm B.

Size-based Approximation We will first prove that B is a $O(n^{2c})$-approximation algorithm if A is a $O(n^c)$-approximation algorithm. By the definition of a $O(n^c)$-approximation algorithm we know that there are constants n_1 and c_1 such that for all instances of *Pb*-Weighted Max Leaf with size $n' \geq n_1$ the value found by the approximation algorithm is at least $\text{OPT}/c_1(n')^c$. We may assume without loss of generality that $n' \geq n_1$ for the instance of *Pb*-Weighted Max Leaf that results from the reduction, because for all instances smaller than n_1 we can solve the problem optimally in polynomial time since n_1 is a constant. Recall that T' is the spanning tree found by running algorithm A on the instance resulting from the reduction. By the assumption on the approximation quality of A we find that:

$$\underset{w_{G'}}{\text{LW}}(T') \geq \underset{w_{G'}}{\text{LW}}(T^*)/(c_1(n')^c) \geq \underset{w_{G'}}{\text{LW}}(T^*)/(c_1 n^{2c}), \tag{6}$$

where the last step follows from the fact that $n' \leq n^2$ for $n \geq 2$ by definition of the reduction. Combining (6) with (5) we find:

$$\underset{w_{G'}}{\text{LW}}(T') \geq kn^2/(c_1 n^{2c}). \tag{7}$$

Combining (7) with (4) we find that:

$$k' \geq \frac{1}{n^2} kn^2/(c_1 n^{2c}) - 1 = k/(c_1 n^{2c}) - 1. \tag{8}$$

So the number k' of heavy leaves in the approximate solution T' is at least $k/(c_1 n^{2c}) - 1$, where k is the optimum of the Independent Set instance. Since the heavy leaves in T' correspond to the independent set that is output by algorithm B, we may conclude from (8) that algorithm B is indeed a $O(n^{2c})$-approximation for Independent Set since $k/(c_1 n^{2c}) - 1 \geq k/O(n^{2c})$.

Value-based Approximation We now prove that B is a $O(n^{3c})$-approximation algorithm if A is a $O(\text{OPT}^c)$-approximation algorithm. By definition of this type of approximation guarantee we find that there are constants c_2, n_2 such that for all sufficiently large instances with $n' \geq n_2$ it holds that:

$$\underset{w_{G'}}{\text{LW}}(T') \geq \underset{w_{G'}}{\text{LW}}(T^*)/(c_2(\underset{w_{G'}}{\text{LW}}(T^*))^c) = \frac{1}{c_2}(\underset{w_{G'}}{\text{LW}}(T^*))^{1-c}. \tag{9}$$

Combining (9) with (5) yields:

$$\underset{w_{G'}}{\text{LW}}(T') \geq \frac{1}{c_2}(kn^2)^{1-c}. \tag{10}$$

Combining (10) with (4) we find that:

$$k' \geq \frac{1}{n^2}\frac{1}{c_2}(kn^2)^{1-c} - 1 = \frac{k}{c_2 k^c n^{2c}} - 1 \geq \frac{k}{c_2 n^c n^{2c}} - 1 = \frac{k}{c_2 n^{3c}} - 1, \tag{11}$$

where the last transformation step follows from the fact that $k \leq n$ since the optimum k of the Independent Set instance is at most the size n of the graph G. Hence B is a polynomial time $O(n^{3c})$-approximation algorithm for Independent Set. This concludes the proof of Theorem 5. □

Using Proposition 1 we obtain the following corollary.

Corollary 1. Pb-Weighted Max Leaf *does not have a polynomial-time $O(n^{\frac{1}{2}-\epsilon})$-approximation algorithm or $O(\text{OPT}^{\frac{1}{3}-\epsilon})$-approximation algorithm for any $\epsilon > 0$ unless P = NP.*

The corollary shows that the introduction of vertex weights makes Max Leaf much harder to approximate: the unweighted Max Leaf problem has a constant factor 2-approximation [21] and is in fact APX-complete [22], but even the restriction *Pb*-Weighted Max Leaf of the weighted version is not in APX unless P = NP. It is uncommon for optimization problems to exhibit structural differences in hardness of approximation when comparing unweighted versions to polynomially-bounded weighted versions. Crescenzi et al. [5] investigated weighted and unweighted variants of optimization problems with respect to their approximation properties, and found that weights do not structurally alter the hardness of approximation for Minimum Vertex Cover, Minimum Satisfiability, Maximum Cut, Maximum DiCut, Maximum 2-Satisfiability and Maximum Exact k-Satisfiability.

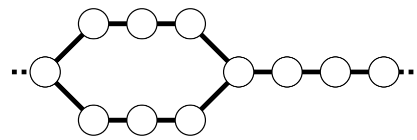


Fig. 11. Component that shows that the factor 5.5 is best-possible: the graph obtained by replacing every vertex of a simple cycle of length n by this subgraph is a member of $\mathcal{C}$. It has $11n$ vertices and at most $2n + 2$ leaves in any spanning tree.

6 Conclusion

We have presented a simple problem kernel with $5.5k$ vertices for Weighted Max Leaf, using new weight-based reduction rules. This serves as an example of how to obtain effective and efficient data reduction on problems with vertex weights. These weights can be used to model real-world problems more accurately. A large part of this work was devoted to the proof of a combinatorial result that graphs excluding some simple substructures always have spanning trees with many leaves; using this result the kernelization effort reduces to eliminating those substructures in the input graph without blowing up the parameter value.

The use of this kernelization algorithm is not restricted to solving the decision variant of Weighted Max Leaf, but it can also be used to construct a spanning tree with the desired leaf weight if one exists. This stems from the fact that all the reduction rules can be reversed to lift a spanning tree for the reduced graph back to the original graph, and from the fact that the combinatorial proof of the extremal result is constructive. When Theorem 1 assures there is a spanning tree with at least k leaves (and hence leaf weight at least k), then such a spanning tree can be found in polynomial time by executing the augmentation operations.

The size of the resulting problem kernel directly corresponds to the extremal bound from Theorem 1. The factor 5.5 in this theorem is best-possible as shown by the construction in Figure 11, which implies that the analysis of the kernel size is tight. It can be shown that Rule [1] and Rule [2] are sufficient to obtain a kernel of $7.5k$ vertices, and that successively adding Rule [3] and Rule [4] leads to kernels with $7k$ and $5.5k$ vertices, respectively. We found a reduction rule to eliminate the structure shown in Figure 11, but we chose not to incorporate this rule in the presentation since it does not lead to a kernel with less than $5.25k$ vertices while it significantly complicates the proof of the required strengthening of Theorem 1. Some illustrations supporting these claims can be found in the appendix.

The proof of the extremal result of Theorem 1 uses an extension of the method of amortized analysis by keeping track of dead leaves; we extended the method by incorporating a new term ΔS in the incremental inequality, which allows us to exploit the fact that the considered graphs do not have long path components. We believe that this technique may be of independent interest since it can be used to prove similar results about leafy spanning trees in graph classes that avoid other subgraphs.

The kernelization procedure for Weighted Max Leaf raises the question whether the existing FPT algorithms for Max Leaf can be converted to the weighted setting. We have verified that this is indeed the case for the $O(6.75^k \cdot n^{O(1)})$ algorithm by Bonsma and Zickfeld [16]; their algorithm uses reduction rules to eliminate "diamonds" and "blossoms" in the input graph, and then guesses the leaf status of the vertices of degree at least 3 in the remaining graph. Given the status of the high-degree vertices the optimal number of leaves can be computed by finding a minimum *edge weighted* spanning tree. This approach carries over to the weighted setting by building weight-aware reduction rules that eliminate the diamonds and blossoms. It is an open question whether the $O(4^k n^{O(1)})$ FPT algorithm of Kneis et al. [10] can be adapted to solve the weighted problem.

We have shown that Weighted Max Leaf is hard to approximate: there is no polynomial-time $O(n^{\frac{1}{2}-\epsilon})$-approximation algorithm or $O(\textsc{opt}^{\frac{1}{3}-\epsilon})$-approximation algorithm for any $\epsilon > 0$ unless P = NP. This shows that Weighted Max Leaf is an example of a problem for which the natural parameterization admits a linear-vertex kernel, but where the associated optimization problem does not have a constant-factor approximation algorithm unless P = NP. To our knowledge this is the first problem that shows this kind of behavior; we often find that problems that have a linear-vertex kernel also admit constant-factor approximation algorithms, with Vertex Cover being a notable example. We expect that further study into the parameterized complexity of weighted graph problems could shed more light on the connection between approximation algorithms and kernels. The effects of different parameterizations and weights of 0 on the fixed parameter complexity of Weighted Max Leaf will be reported elsewhere.

Acknowledgements I would like to thank Hans Bodlaender for his guidance and advice, Peter Rossmanith for a brief but inspiring discussion about rational-valued vertex weights during TACO day '09 and finally Marinus Veldhorst for inspiring me to study spanning tree problems.

References

1. Downey, R., Fellows, M.R.: Parameterized complexity. Monographs in computer science. Springer, New York (1999)
2. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: MFCS. (2006) 238–249
3. Fujito, T., Nagamochi, H.: A 2-approximation algorithm for the minimum weight edge dominating set problem. Discrete Appl. Math. **118** (2002) 199–207
4. Halldrsson, M.M.: Approximations of weighted independent set and hereditary subset problems. Journal of Graph Algorithms and Applications **4** (2000) 2000
5. Crescenzi, P., Silvestri, R., Trevisan, L.: On weighted vs unweighted versions of combinatorial optimization problems. Inf. Comput. **167** (2001) 10–26
6. Zimand, M.: Weighted NP optimization problems: Logical definability and approximation properties. SIAM J. Comput. **28** (1999) 36–56
7. Böcker, S., Briesemeister, S., Bui, Q.B., Truss, A.: A fixed-parameter approach for weighted cluster editing. In: Series on Advances in Bioinformatics and Computational Biology. Volume 5., Imperial College Press (2008) 211–220
8. Lemke, P.: The maximum leaf spanning tree problem for cubic graphs is NP-complete. IMA publication no. 428 (1988)
9. Estivill-Castro, V., Fellows, M., Langston, M., Rosamond, F.: FPT is P-time extremal structure I. In: Proc. 1st ACiD. (2005) 1–41
10. Kneis, J., Langer, A., Rossmanith, P.: A new algorithm for finding trees with many leaves. In: Proc. 19th ISAAC. (2008) 270–281
11. Raible, D., Fernau, H.: An amortized search tree analysis for k-leaf spanning tree. In: Proc. 36th SOFSEM. (2010) 672–684
12. Fernau, H., Kneis, J., Kratsch, D., Langer, A., Liedloff, M., Raible, D., Rossmanith, P.: An exact algorithm for the maximum leaf spanning tree problem. In: Proc. 4th IWPEC. (2009) 161–172
13. Storer, J.A.: Constructing full spanning trees for cubic graphs. Information Processing Letters **13** (1981) 8–11
14. Griggs, J.R., Kleitman, D., Shastri, A.: Spanning trees with many leaves in cubic graphs. J. Graph Theory **13** (1989) 669–695
15. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discret. Math. **4** (1991) 99–106
16. Bonsma, P.S., Zickfeld, F.: Spanning trees with many leaves in graphs without diamonds and blossoms. In: Proc. 8th LATIN, Búzios, Brazil (2008) 531–543
17. Bollobas, B.: Extremal Graph Theory. Courier Dover Publications (2004)
18. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer (2000)
19. Hastad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Annual IEEE Symposium on Foundations of Computer Science (1996) 627
20. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing **3** (2007) 103–128
21. Solis-oba, R.: 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. In: Lect. Notes Comput. Sci, Springer (1998) 441–452

22. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Information Proccessing Letters (1994)
23. Prieto, E.: Systematic Kernelization in FPT Algorithm Design. PhD thesis, University of Newcastle, Australia (2005)

A Spanning Trees with Many Leaves in Graphs without Longs Paths: A Simple Proof

In this section we give a relatively simple proof of the existence of spanning trees with many leaves in graphs that do not have long paths of degree-2 vertices. To make the proof self-contained, we repeat some material that was already present in the main text.

Definition 5. *Let $\mathcal{C}^{\mathcal{P}}$ be the class of graphs $G = (V, E)$ that satisfy the following properties:*

(i) Graph G is simple and connected.
(ii) The graph is not isomorphic to a simple cycle.
(iii) The maximum size of a path component in G is at most $\mathcal{P}$.
(iv) Every vertex $v \in V$ with $\deg(v) = 1$ is adjacent to a vertex of degree at least 3.

Theorem 6. *Every graph $G = (V, E) \in \mathcal{C}^{\mathcal{P}}$ for $\mathcal{P} \geq 2$ has a spanning tree with at least $|V|/(3 + 1.5\mathcal{P})$ leaves.*

Proof. Consider $G = (V, E) \in \mathcal{C}$. Observe that connected graphs with fewer than 3 vertices do not satisfy Property (iv), so any graph $G \in \mathcal{C}$ must have $|V| \geq 3$. Our proof uses the method of amortized analysis by keeping track of dead leaves, as introduced by Griggs et al. [14]. The proof is constructive and consists of a series of operations that can be used to initialize a tree $T \subseteq G$, and to augment T if it does not yet span G. We will prove that the resulting tree T has sufficient leaves by showing for every augmentation step that the increase in the total size of the tree is balanced against the increase in the number of leaves of the tree. To analyze the number of leaves in the resulting spanning tree we use the notion of *dead leaves*. A leaf $v \in \textsc{Leaves}(T)$ is called *dead* if all its neighbors in G are also in T. More formally, leaf v is dead iff. $N_G(v) \subseteq V(T)$. Every leaf vertex that is not dead, is alive. We define the following abbreviation for the set of live leaves:

$$\underset{G}{\textsc{LiveLeaves}}(T) := \{v \in \textsc{Leaves}(T) \mid N_G(v) \setminus V(T) \neq \emptyset\}. \tag{12}$$

If vertex $v \in V$ has a neighbor $u \in N_G(v)$ but the vertex u is not in T, then we say that v has a neighbor u outside the tree. If $u \in N_G(v)$ and $u \in V(T)$ then vertex v has a neighbor u inside the tree. A vertex $x \in N_G(V(T))$ is said to be *adjacent* to the tree T. Our analysis uses the following properties of the tree T that is under construction:

- L, the number of *leaves* of T,
- D, number of *dead* leaves of T,
- N, the total number of vertices of T.

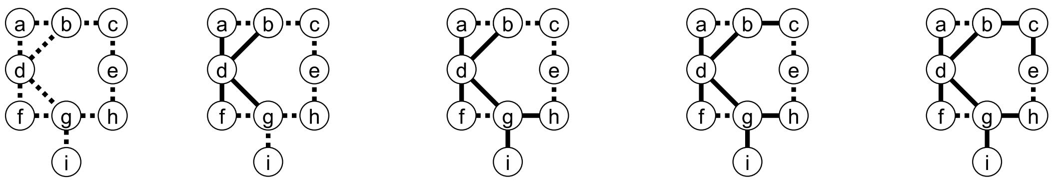
We will give a series of augmentation operations that satisfy the following *incremental inequality*:

$$(3 + \mathcal{P})\Delta L + (\mathcal{P}/2)\Delta D \geq \Delta N \tag{13}$$

The Δ values in this inequality represent the changes in the respective quantities in the new tree compared to the old tree. For example, if the tree had 5 leaves before the augmentation operation and it has 7 leaves after the operation then $\Delta L = 7 - 5 = 2$.

Observe that in a spanning tree there can be no live leaves: all neighbors to all vertices must be in the tree. This means that all leaves must be dead and hence $L = D$. It follows that if we grow a spanning tree such that every augmentation step satisfies the incremental inequality, then by summing the inequalities over the individual augmentation steps we find that the final tree satisfies $(3 + \mathcal{P})L + (\mathcal{P}/2)D \geq N$; using the fact that $L = D$ on spanning trees we can then conclude that $|\textsc{Leaves}(T)| \geq |V|/(3 + 1.5\mathcal{P})$ for a tree T that is grown by operations that respect the incremental inequality. So to prove the claim all that remains is to show that there exists a set of augmentation operations that can grow a spanning tree while respecting the incremental inequality.

The initialization of the tree T is simple: we just pick a vertex v of maximum degree in G as the root of the tree, and we add edges to all neighbors of v to the tree. So after initialization we



(a) (b) $(4,2,4)$ (c) $(1,1,2)$ (d) $(0,0,1)$ (e) $(0,2,1)$

Fig. 12. This sequence shows the effect of successive vertex expansions. Dotted lines indicate edges in $E(G)\setminus E(T)$, and solid lines represent edges in $E(G)\cap E(T)$. Each expansion is labeled with the corresponding vector $(\Delta L, \Delta D, \Delta N)$ of changes in the measured quantities. 12(a): the graph G without a tree subgraph. 12(b): initialized T as the star around vertex d, causing a, f to become *dead* leaves and b, g to become *live* leaves. 12(c): expanded g, adding i, h to T and immediately making i a dead leaf. 12(d): expanded b. 12(e): expanded c, which causes vertices e and h to become dead leaves.

have a tree with one internal vertex v and leaf set $\textsc{Leaves}(T) = N_G(v)$. For this operation we have $\Delta N = 1 + |N_G(v)|$ since the tree is a star rooted at v, and $\Delta L = |N_G(v)|$ because all neighbors of v have become leaves. The value ΔD cannot be negative because there were no leaves before the tree was initialized. With this information it can easily be seen that the initialization of the tree satisfies the incremental inequality. We need the following definition to describe the operations that augment the tree.

Definition 6 (Vertex Expansion). *Let $T \subseteq G$ be a tree. The* expansion *of a vertex $v \in V(T)$ yields an augmented tree $T' \subseteq G$, where T' is obtained by adding all edges $vu \in E(G)$ with $u \in N_G(v) \setminus V(T)$ to the tree T.*

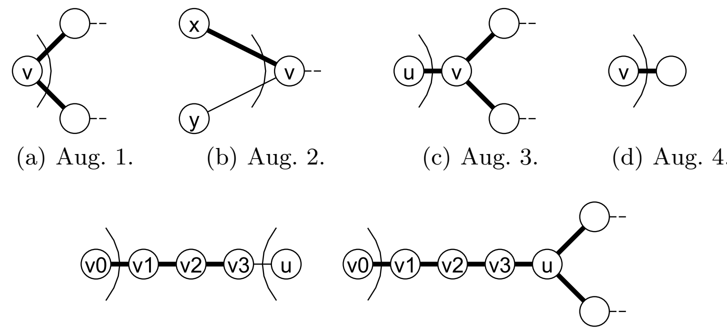
Figure 12 shows an example of vertex expansions. It follows from the definition that the expansion of a vertex can never decrease the number of leaves in the tree. If v has a neighbor $u \in N_G(v)\setminus V(T)$, then expansion of v will cause v to become internal (decreasing the number of leaves), but it will also cause u to become a new leaf so there is never a net decrease in the number of leaves. If v has no neighbors outside T, then the expansion has no effect and the number of leaves does not decrease either. The operations that we will present to augment the tree consist of series of vertex expansions. This means that $\Delta L \geq 0$ for every augmentation step. Since the expansion of a vertex can not change the fact that a leaf is dead, we also have $\Delta D \geq 0$ for all our augmentation operations. By growing the tree through expansion operations we maintain the invariant that for every internal vertex of T, all its neighbors in G are also in T. This implies that all vertices of T that have a neighbor outside T must be *leaves*; this will be important later on in the proof.

Let us now consider the operations that augment the tree. The operations are listen in a specific order - this order is relevant to the proof. When constructing a spanning tree, an operation should only be used if none of the operations that came earlier in the listing can be applied.

Augmentation Operation 1 If there is a live leaf $v \in \textsc{LiveLeaves}_T(G)$ that has at least two neighbors outside T, then expand v.

Let us verify that this augmentation satisfies the incremental inequality. Consider the set $N_G(v) \setminus V(T)$ with neighbors of v outside of T prior to the augmentation. All of these vertices will become leaves after the expansion of v. Since the expansion of v causes v to become an internal node of the tree, the net change in leaves is $\Delta L = |N_G(v) \setminus V(T)| - 1$. The only vertices that are added to T by this augmentation are the neighbors of v outside T, so we have $\Delta N = |N_G(v) \setminus V(T)|$. Combined with the fact that $\Delta D \geq 0$ for all operations that consist of vertex expansions, it is easy to see that this operation satisfies the incremental inequality.

Observation 1 If Augmentation 1 cannot be applied, then all live leaves have exactly one neighbor outside T.



(a) Aug. 1. (b) Aug. 2. (c) Aug. 3. (d) Aug. 4.

(e) Aug. 5. (f) Aug. 6.

Fig. 13. This figure gives graphical representations of the augmentation operations. Each subfigure represents the structure of the boundary of the tree to which its augmentation is applicable, i.e. each subfigure shows a substructure of G and the tree T near that substructure. Vertices of G that are also in T are drawn to the inside of an arc; the remaining vertices are in G, but not yet in T. The augmentation operation is visualized by drawing all edges that are added to T by the augmentation as thick lines. Edges of the graph that are not added to the tree by the augmentation are drawn as thin lines. A dotted line indicates a potential edge in the graph that is not added to T by the augmentation.

Augmentation Operation 2 If there is a vertex $v \in N_G(V(T))$ that has at least two neighbors x, y inside T, then expand x.

It is important to note that prior to this augmentation, both vertices x and y must have exactly one neighbor outside T by Observation 1 and hence they must be leaves of the tree. Since the vertex v is added to T by the expansion of x, it follows that afterwards the leaf y has no neighbors outside T and it has become a dead leaf; the same holds for any other neighbors that v might have inside T. The leaf x turns into an internal node by this operation, but in return we get v as a new leaf. Therefore we have $\Delta L = 0$, $\Delta D \geq 1$ and $\Delta N = 1$ which satisfies the incremental inequality.

Observation 2 If Augmentation 2 cannot be applied, then all vertices in $N_G(V(T))$ have exactly one neighbor in T.

Augmentation Operation 3 If there is a vertex $v \in N_G(V(T))$ that has at least two neighbors outside T, and vertex v is adjacent in G to vertex $u \in V(T)$, then first expand u and then expand v.

All neighbors of v outside T will become leaves after this augmentation. The only vertices added to T by the augmentation are v and its neighbors outside T. The leaf u becomes an internal node through this operation. Hence we have $\Delta L = |N_G(v) \setminus V(T)| - 1$, $\Delta D \geq 0$ and $\Delta N = |N_G(v) \setminus V(T)| + 1$ which satisfies the incremental inequality.

Observation 3 If Augmentation 3 cannot be applied, then all vertices in $N_G(V(T))$ have exactly one neighbor outside T.

Observation 4 If Augmentation 2 and Augmentation 3 cannot be applied, then all vertices in $N_G(V(T))$ have degree at most 2.

Augmentation Operation 4 If there is a vertex $v \in V(T)$ that has a degree-1 neighbor outside T, then expand v.

If this operation is applicable and none of the earlier operations are, then by Observation 1 vertex v has only one neighbor outside T. The expansion of v causes it to become an internal node, but in return its degree-1 neighbor that was outside T before the expansion now becomes a leaf; and in fact it must become a dead leaf since its only neighbor is already in the tree. Therefore we have $\Delta L = 0$ and $\Delta D = \Delta N = 1$ which satisfies the incremental inequality.

Observation 5 If Augmentation 4 cannot be applied, then all vertices in $N_G(V(T))$ have degree at least 2.

Observation 6 If no earlier augmentation can be applied, then all vertices in $N_G(V(T))$ have degree 2 and have exactly one neighbor inside T and one neighbor outside T.

These observations show that when none of the earlier augmentation operations can be applied, the boundary of the tree must have a very specific structure. We will make use of this by showing that there are only two structurally different ways the graph can behave near the boundary of the tree, and that for both cases there is a valid augmentation operation.

So assume the earlier operations are not applicable and consider a vertex $v_1 \in N_G(V(T))$. If there is no such vertex then T is already a spanning tree, and we are done. We know that v_1 has exactly one neighbor outside T; let this be v_2. Consider the path that starts with v_1, v_2 and continues until it encounters a vertex u which is in T, or has degree unequal to 2. The vertices on this path are $v_1, \ldots, v_q, u$ and by definition we know that $\deg_G(v_i) = 2$ and $v_i \notin V(T)$ for $1 \leq i \leq q$. Let v_0 be the unique neighbor of v_1 that is inside T. Observe that all vertices v_i $(1 \leq i \leq q)$ lie on the same path component, and the length of this component is at least q; hence by Property (iii) of Definition 5 we know that $q \leq \mathcal{P}$. There are two possibilities for the structure of this path; we can have $u \in V(T)$, and we can have $u \notin V(T)$ and $\deg_G(u) \neq 2$. We will give separate augmentation operations for these possibilities.

Augmentation Operation 5 If the graph G contains a simple path $P = \langle v_0, v_1, \ldots, v_q, u \rangle$ such that $\deg_G(v_i) = 2$ and $v_i \notin V(T)$ for $1 \leq i \leq q$, and $v_0, u \in V(T)$ then expand $v_0, v_1, \ldots, v_{q-1}$.

The vertex v_q is added to T by these expansions. The degree of v_q is 2 by definition, its neighbor u is inside T by assumption and its neighbor v_{q-1} is in T because we expand $v_0, v_1 \ldots, v_{q-1}$. Hence after the expansions the vertex v_q is a leaf whose neighbors are all inside T, which means it is a dead leaf. Before the expansions the vertex u has a neighbor outside T and must therefore be a live leaf. By Observation 1 the vertex u has exactly one neighbor outside T before the expansions, which implies that after the augmentation it must have become a dead leaf. Taking all of this into account we find that $\Delta L = 0$, $\Delta D = 2$ and $\Delta N \leq \mathcal{P}$ which satisfies the incremental inequality.

Augmentation Operation 6 If the graph G contains a simple path $P = \langle v_0, v_1, \ldots, v_q, u \rangle$ such that $\deg_G(v_i) = 2$ and $v_i \notin V(T)$ for $1 \leq i \leq q$, and $v_0 \in V(T)$, $u \notin V(T)$ and $\deg_G(u) \neq 2$ then expand $v_0, v_1, \ldots, v_{q-1}, v_q, u$.

If $u \notin V(T)$ then by definition of u we know $\deg(u) \neq 2$, and since u is adjacent in G to the degree-2 vertex v_q we get from Property (iv) that $\deg_G(u) \geq 3$. By Observation 6 this implies that $u \notin N_G(V(T))$ and therefore none of the neighbors of u are in T. Therefore the expansion of the vertices $v_0, \ldots, v_q, u$ causes all neighbors of u except v_q to become leaves, and the existing leaf v_0 becomes an internal node; we have $\Delta L = |N_G(u) \setminus \{v_q\}| - 1$. The number of dead leaves cannot decrease so we have $\Delta D \geq 0$, and finally since the value of q is bounded by the length $\mathcal{P}$ of the largest path component we have $\Delta N \leq \mathcal{P} + |N_G(u)|$. It is a simple exercise to show that these values must satisfy the incremental inequality since $|N_G(u)| = \deg_G(u) \geq 3$. This concludes the description of the augmentation operations.

We have shown that there are only two possibilities for the structure of the tree near the boundary of the graph when Augmentation 4 or earlier augmentations cannot be applied, and that there are augmentation operations to handle both possibilities. This shows that whenever T is not a spanning tree we can apply some augmentation operation that satisfies the incremental inequality. By our earlier analysis of the implications of the incremental inequality this completes the proof of Theorem 6. □

A.1 Implications

Theorem 6 can be exploited to give simple kernels for Weighted Max Leaf and Max Leaf. We briefly indicate how the kernels can be obtained, in the hope that the resulting simple kernels may be used in the classroom when teaching; both the reduction rules and the analyses are significantly simpler than the best-known kernelization results for these problems.

Kernel With 7.5*k* Vertices For Weighted Max Leaf Although Theorem 1 was tailored towards graphs that do not have path components of length larger than 3, the proof of Theorem 6 gives a lower bound on the number of leaves for *any* maximum size of path component $\mathcal{P}$. Using the value $\mathcal{P} = 3$ we can interpret Theorem 6 as a weaker form of Theorem 1: the simpler theorem yields a bound of $|V|/7.5$ leaves, whereas we obtained a bound of $|V|/5.5$ by the complicated theorem. Observe that the conditions (v) and (vi) of Definition 1 are not mentioned in Definition 5; these conditions are required to get the bound of 5.5, but not needed to obtain 7.5. This implies that there is a simple kernel with $7.5k$ vertices for Weighted Max Leaf that only uses Rule [1] and Rule [2].

Kernel With 5*k* Vertices For Max Leaf It is not hard to verify that the proof of Theorem 6 can be adapted for the class $\mathcal{C}^{\mathcal{P}}$ with $\mathcal{P} = 1$ to yield a lower bound of $|V|/5$ leaves; it suffices to change the $(\mathcal{P}/2)\Delta D$ term in the incremental inequality to $1\Delta D$. Through this observation we get a simple kernel with $5k$ vertices for the unweighted Max Leaf problem, and an elegant combinatorial proof of the size bound. It may be verified that we can reduce an instance of unweighted Max Leaf to a graph in $\mathcal{C}^1$ by applying Rule [2] in combination with the following rule that has been used in other kernels for Max Leaf [23, p. 42–44]: if there is an edge uv between vertices of degree 2, then contract this edge if it is a bridge and delete the edge if it is not a bridge.

B Extremal Graph Constructions

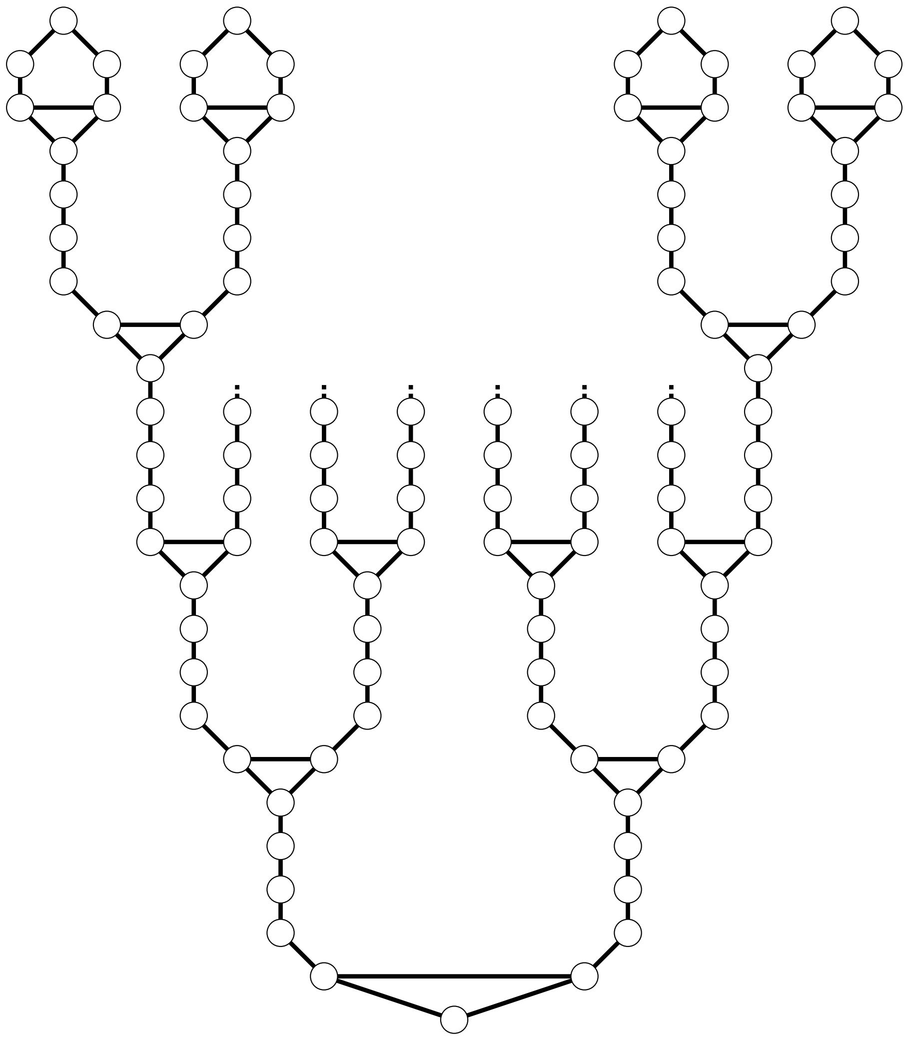


Fig. 14. Construction that shows that only using reduction rules [1]-[2] cannot lead to a kernel with ck vertices for $c < 7.5$. The given construction can be interpreted as a complete binary tree where every node is replaced by a substructure. A graph obtained in this way is not reducible by the mentioned set of reduction rules, and it can be verified that the graph obtained by transformation from a complete binary tree of height h has $6 \cdot 2^{h+1} + 3 \cdot 2^h - 4$ vertices, and at most $2 \cdot 2^h$ leaves in any spanning tree.

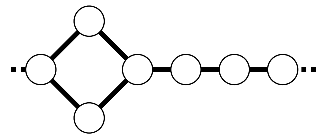


Fig. 15. Component that shows that only using reduction rules [1]-[3] cannot lead to a kernel with ck vertices for $c < 7$: the graph obtained by replacing every vertex of a simple cycle of length n by this subgraph is not reducible by the mentioned set of reduction rules. It has $7n$ vertices and at most $n + 2$ leaves in any spanning tree.

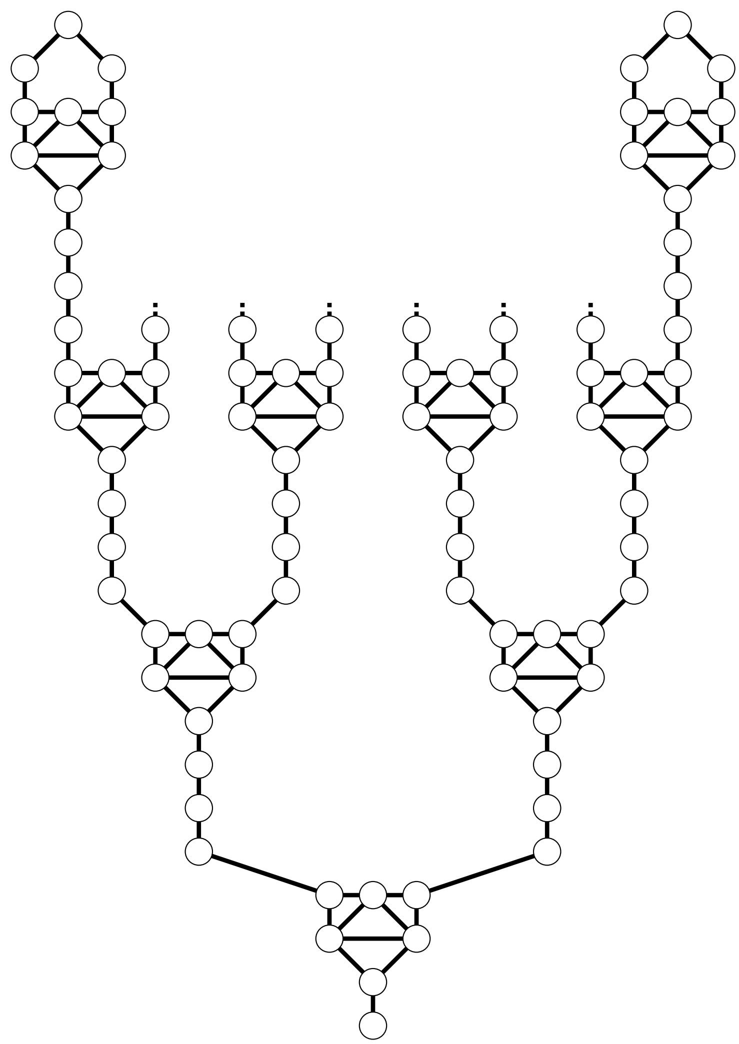


Fig. 16. Construction that shows that even when eliminating the substructure of Figure 11 we do not obtain a kernel with ck vertices for $c < 5.25$. The graph obtained by transforming from a complete binary tree of height h has $9 \cdot 2^{h+1} + 3 \cdot 2^h - 2$ vertices, and at most $1 \cdot 2^{h+1} + 2 \cdot 2^h$ leaves in any spanning tree.