Fixed Parameter Complexity of the Weighted Max Leaf Problem

by

Bart Jansen

Thesis

Submitted to the Utrecht University

for the degree of

Master of Science

Supervisors: dr. H. L. Bodlaender and dr. G. Tel

Thesis number Inf/Scr 08 75

Department of Computer Science

July 2009

Universiteit Utrecht

Fixed Parameter Complexity of the Weighted Max Leaf Problem

Bart Jansen

Utrecht University bmpjanse@cs.uu.nl
Inf/Scr_08_75

Abstract. In this paper we consider the fixed parameter complexity of the Weighted Max Leaf problem, where we get as input an undirected connected graph $G = (V, E)$, a set of numbers S and a weight function $w : V \mapsto S$ on the vertices, and are asked whether a spanning tree for G exists such that the combined weight of the leaves of the tree is at least k. The fixed parameter complexity of the problem strongly depends on whether we allow vertices to have a weight of 0. We show that if $S = \mathbb{N}$ then this problem admits a kernel with $7.5k$ vertices. We prove that the case $S = \{0\} \cup \mathbb{N}$ is hard for W[1] on general graphs, but that a kernel exists with $78k$ vertices on planar graphs, $O(\sqrt{\gamma}k+\gamma)$ vertices on graphs of oriented genus at most γ, and with $(1+4\delta)(1+\frac{\delta}{2})k$ vertices on graphs in which the degree of positive-weight vertices is bounded by δ. For the correctness proofs of the kernels we develop some extremal graph theory about spanning trees with many leaves in (bipartite) graphs without long paths of degree-2 vertices.

1 Introduction

Spanning tree problems frequently arise in the design of communication networks [1]. For example, the problem of finding a connected backbone network for communication such that a maximum number of network nodes only have to perform direct communication with a single peer, can be modeled as an instance of the Maximum Leaf Spanning Tree problem (Max Leaf). In its optimization variant, the problem is to find a spanning tree T of a given undirected connected graph, such that T maximizes the number of leaf vertices over all possible spanning trees. The decision version of Max Leaf is formally defined as follows:

Max Leaf
Instance: An undirected connected graph $G = (V, E)$; a positive integer k.
Question: Does G have a spanning tree with k or more leaves?

Besides applications in network design Max Leaf also has uses in circuit layout [2] and in determining the Vapnik-Chervonenkis of some special types of set systems [3]. The problem has a very long history and has not only been considered from a computational point of view, but also in the context of extremal graph theory. The Max Leaf problem was identified as NP-complete on planar graphs of maximum degree 4 by Garey and Johnson [4] in their famous work on computational complexity. Nine years later, Peter Lemke [5] showed that it is also NP-complete on 3-regular graphs. Even though the problem was proven to be APX-complete [6] - which denies the existence of a polynomial time approximation scheme (PTAS) - there have been several results concerning constant-factor approximation algorithms for the problem, culminating in a linear time 2-approximation by Solis-Oba [7]. James Storer [2] first considered the special case of cubic graphs, for which a $\frac{3}{2}$-approximation algorithm was recently presented [8].

The problem of finding spanning trees with many leaves is closely related to finding a small connected dominating set. For a simple connected graph G with $n \geq 3$ vertices, it can be shown that a spanning tree with at least k leaves exists if and only if there is a connected dominating set for G consisting of $n - k$ vertices.

In recent years people have started to investigate variants of the Max Leaf problem on bipartite graphs. The following problem was first posed by Rahman and Kaykobad [9]:

Bipartite Max Leaf
Instance: An undirected bipartite connected graph $G = (V, E)$ with one partite set marked *black* and the other set marked *white*; a positive integer k.
Question: Does G have a spanning tree with k or more *black* leaves?

Table 1. Complexity overview of the unweighted problems.

Problem	NP-complete for	Fixed parameter	Approximation
Max Leaf	planar max. deg. 4 [4] 3-regular [5]	$3.75k$ kernel [15] $O^*(4^k k^2)$ [16]	APX-complete [6] 2-approx. [7]
Bipartite Max Leaf	planar max. deg. 4 [10] d-regular $d \geq 4$ [11]	$W[1]$-hard	No c-approx.

The classical complexity of Bipartite Max Leaf was established through a publication by Li and Toulouse [10], who showed it to be NP-complete on planar graphs of maximum degree 4. One year later is was also shown [11] to be NP-complete on d-regular graphs for $d \geq 4$. To our knowledge we are the first to consider the approximability of Bipartite Max Leaf, by showing in this work that Bipartite Max Leaf is at least as hard to approximate as Independent Set. This implies that no constant-factor approximation algorithm exists unless P = NP [12, 13]. Fusco and Monti [11] have presented linear time constant-factor approximation algorithms for the special case of d-regular graphs, achieving an approximation factor of $2 - 2/(d-1)^2$.

Table 1 gives an overview of the complexity of Max Leaf and Bipartite Max Leaf. The main subject of this work is the following problem, which has received some attention from the Operations Research community [14]:

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

Since this weighted problem generalizes Max Leaf it is NP-complete on planar graphs of maximum degree 4 and for 3-regular graphs. When we allow weights to be elements of $\{0, 1\}$ then the weighted problem also generalizes Bipartite Max Leaf, from which we can also conclude NP-completeness on d-regular graphs for $d \geq 4$.

The subject of spanning trees has also received considerable attention from the perspective of extremal graph theory. In 1981 Storer initiated the research by giving an approximation algorithm for Max Leaf on graphs of maximum degree 3, analyzing the performance ratio of the algorithm by studying the structural properties of spanning trees in the graph. This formed the inspiration for Griggs, Kleitman and Shastri to analyze the structural properties of spanning trees in 3-regular graphs. They showed that every connected cubic graph with n vertices contains a spanning tree with at least $\lceil \frac{n}{4} + 2 \rceil$ leaves, and that this bound is tight. Kleitman and West proved in 1991 that this lower bound also holds when we consider graphs of *minimum* degree 3. The study of leafy spanning trees was continued by Bonsma and Zickfeld [17] with the purpose of aiding FPT algorithm design. They showed that every graph of minimum degree 3 that excludes two specific types of subgraphs called *diamonds* and *blossoms*, contains a spanning tree with at least $\lceil (n+4)/3 \rceil$ leaves.

In this work we go beyond the classical complexity analysis of these problems, and consider them from a *Fixed Parameter Complexity* [18] point of view. Fixed parameter complexity theory was developed by Downey and Fellows as a way to cope with the apparent intractability of NP-hard problems. Classical complexity theory considers the time required to solve a problem measured in a single parameter n, the size of the input. Fixed parameter complexity theory is basically a two-dimensional approach to complexity analysis; besides the instance size n we also take into account an integer k called the *parameter* of the problem which measures some characteristic of the problem instance. A formal parameterized decision problem is a set $P \subseteq \Sigma^* \times \mathbb{N}$, and an instance of a P is a tuple $\langle I, k \rangle$ where I is the "regular" problem instance and k denotes the parameter. We then ask whether the problem can be solved in $f(k)p(|I|)$ time, where f can be any arbitrary (exponential) function and p is a polynomial. Any problem that can be solved within such time bounds is called *fixed parameter tractable* (FPT). For FPT problems we can effectively restrict the combinatorial explosion that is seemingly inherent to solving NP-hard problems to the parameter of the problem. It is then hoped that this yields practical tractability of problems for which the instance size can be large, but the characteristic measured by the parameter is small. Problem instances occurring in practical situations often have internal structure that can be exploited through the parameter.

A technique that is often employed to obtain fixed parameter tractability is *problem kernelization*. We use the working definition as given in [19].

Table 2. Fixed parameter complexity of Weighted Max Leaf. A table entry either states hardness for a parameterized complexity class or gives the number of vertices in the kernel for the problem, implying that the problem is in FPT. The last column corresponds to the problem restricted to graphs where the degree of positive-weight vertices is bounded by δ. $H(\gamma)$ is the Heawood number of γ (Definition 2).

Weight range S	General	Planar	Oriented genus $\leq \gamma$	Degree-bound δ
$\{0\} \cup \mathbb{N}$	Hard for W[1]	$78k$	$(24H(\gamma) - 18)k + 10\gamma$	$(1 + 4\delta)(1 + \frac{\delta}{2})k$
$\{0\} \cup \mathbb{Q}_{\geq 1}$	Hard for W[1]	$O(k)$ kernel	$O(\sqrt{\gamma}k + \gamma)$ kernel	$O(\delta^2 k)$ kernel
$\mathbb{N}$ or $Q_{\geq 1}$	$7.5k$ kernel			
$\mathbb{Q}_{>0}$	NP-complete for fixed parameter value (Not in FPT)			

Definition 1 (Kernelization). *A kernelization algorithm, or in short, a* kernel *for a parameterized problem L is an algorithm that given $\langle I, k\rangle \in \Sigma^* \times \mathbb{N}$ outputs in $p(|I| + k)$ time a tuple $\langle I', k'\rangle \in \Sigma^* \times \mathbb{N}$ such that:*

- $\langle I, k\rangle \in L \Leftrightarrow \langle I', k'\rangle \in L$,
- $|I'|, k' \leq g(k)$,

where g is an arbitrary computable function, and p a polynomial. Any function g as above is referred to as the size *of the kernel.*

If $g(k) \in O(k)$ then we have a *linear* problem kernel. It can easily be seen that every decidable parameterized problem for which a kernelization algorithm exists is in FPT: we first compute a problem kernel of size $|I'| \leq g(k)$ and then decide the problem by brute force on the instance I', yielding a running time of $f(g(k)) + p(|I|)$.

The subject of this work is the parameterized complexity of the Weighted Max Leaf problem when parameterized by the target leaf weight k. When we refer to the fixed parameter variant of the regular Max Leaf problem, we use the target leaf number k as the parameter.

The Max Leaf problem and variants for directed graphs [20, 21] have received a lot of attention from the fixed parameter community. Two types of improvement races have been established: the "kernelization race" and the "$f(k)$" race. The first seeks to find a kernel of minimum size, whereas the $f(k)$ race aims to find a fixed parameter algorithm with the smallest growing function $f(k)$ in the combinatorial explosion of the running time. The fixed parameter tractability of Max Leaf has been studied extensively. The best kernelization result at the moment is a kernel with $3.75k$ vertices by Estivill-Castro et al. [15], whereas the $O(poly(|V|) + 4^k k^2)$ algorithm by Kneis, Langer and Rossmanith [16] currently achieves the lowest running time.

Although the FPT results regarding Max Leaf are numerous, to our knowledge there has never been a study into the fixed parameter complexity of Bipartite Max Leaf. In this work we show that Bipartite Max Leaf is hard for $W[1]$ through a parameterized reduction from k-Independent Set. The fixed parameter complexity of Weighted Max Leaf strongly depends on the range S of weights that are allowed. Throughout this work we distinguish two versions: Weighted Max Leaf$_0$ where $S := \{0\} \cup \mathbb{N}$, and Weighted Max Leaf$_+$ where $S := \mathbb{N}$. The Weighted Max Leaf$_0$ problem generalizes Bipartite Max Leaf by giving white vertices a weight of 0 and black vertices a weight of 1; hence on general graphs it is hard for $W[1]$. We will show that the Weighted Max Leaf$_+$ problem on general graphs is in FPT, by presenting a problem kernel with $7.5k$ vertices. Although this work mainly focuses on Weighted Max Leaf with integer weights, we will also indicate how the results can be extended to rational-valued weights. We do not concern ourselves with weights from the set $\mathbb{R}$ to ensure that we do not have to spend too much attention on the encoding of the vertex weights.

When restricting $W[1]$-hard graph problems to planar graphs, they often become tractable. We will see that this is indeed the case for Weighted Max Leaf$_0$ by giving a linear problem kernel on planar graphs. We extend this kernel to graphs of *bounded genus*, which is basically the class of graphs that can be drawn without crossing edges on a surface with a bounded number of handles. Finally we also show that the kernelization algorithm works on graphs where the degree of the positive-weight vertices is bounded by some fixed δ. Table 2 gives an overview of the Fixed Parameter complexity of Weighted Max Leaf.

Since many graph problems appear to have polynomial problem kernels when restricted to graph classes that can be drawn on simple surfaces, efforts have recently been undertaken to create frameworks to derive small problem kernels for such parameterized problems. The first framework by Guo and Niedermeier [22] aids the search for reduction rules and correctness proofs in a systematic way to obtain linear problem kernels on planar graphs that satisfy two *distance properties*. The framework has been substantially extended by

Bodlaender et al. [23], who show how to find problem kernels of polynomial size for graph problems on planar graphs and graphs of bounded genus. The latter framework does not require problem-specific reduction rules, but uses bounded-treewidth decomposition techniques in combination with the tractability of deciding problems expressible in Counting Monadic Second Order Logic on graphs of bounded treewidth to give general-purpose reduction rules. This framework yields linear kernels for problems that have *Finite Integer Index* and that satisfy a *quasi-compactness* property. Refer to the paper for more details. It should be noted that the problem kernel for WEIGHTED MAX LEAF$_0$ obtained in this work could not have been derived using either framework. The technique of Guo and Niedermeier does not apply since it does not yield kernels for problems on graphs of bounded genus. Since the WEIGHTED MAX LEAF problem does not have the *Finite Integer Index* property, the framework by Bodlaender et al. could not be used to obtain this result either.

This work is structured as follows. In Section 2 we give some general graph-theoretic preliminaries. In Section 3 we prove that BIPARTITE MAX LEAF is hard for $W[1]$. We consecutively develop the structure theory that is required for the correctness proofs of the kernelization algorithms. Section 4 develops the theory that supports the kernel for WEIGHTED MAX LEAF$_+$, whereas the more technical Section 5 contains the structure theory for the WEIGHTED MAX LEAF$_0$ kernel.

We show that WEIGHTED MAX LEAF$_+$ is Fixed Parameter Tractable on general graphs by giving a $7.5k$ problem kernel in Section 7. In Section 8 we present a linear kernel for WEIGHTED MAX LEAF$_0$ on planar graphs, graphs of bounded genus and graphs satisfying the stated degree bound, using the structure theory from the earlier sections.

2 Preliminaries

Refer to standard textbooks on algorithms such as [24] for the basic terminology on graphs and (spanning) trees. All graphs in this work are undirected, connected and simple unless explicitly stated otherwise. 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 $V(G)$ to denote the vertex set of G when this is convenient. For $v \in V$ we denote the *open* neighborhood of v by $N_G(v) := \{u \in V \mid uv \in E\}$. 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 $\deg_G(v) := |N_G(v)|$. The *size* of G is $|G| := |V|$. A graph $G' = (V', E')$ is a *subgraph* of G iff. $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$. We denote this by $G' \subseteq G$. For $X \subseteq V$ we denote by $G[X]$ the subgraph of G that is induced by the vertices in X, i.e. $G[X] := (X, (X \times X) \cap E)$. 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 $N(x) := N(u) \cup N(v) \setminus \{u, v\}$.

A tree is a connected, acyclic graph. If $T \subseteq G$ is a tree with $V(T) = V(G)$ then T is a *spanning tree* for G. If T is a tree subgraph such that $T = (V', E') \subseteq G = (V, E)$ and e is an edge with $e \in E$ and $e \notin E'$, then we say that T *avoids* edge e. If $e \in E$ and $e \in E'$ then tree T *uses* edge e. A vertex of degree 1 is called a *leaf*, or an external node. 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 $\text{LEAVES}(G) := \{v \in V \mid \deg_G(v) = 1\}$. If we have a weight function $w : V \mapsto S$ for graph G then we can define its *leaf weight* as $\text{LW}_w(G) := \sum_{v \in \text{LEAVES}(G)} w(v)$. We define the *weight* of an edge uv as the sum of the weights of its endpoints. For a graph G that is weighted with function w we denote the maximum degree of a positive-weight vertex by $\text{POSDEG}_w(G) := \max\{\deg_G(v) \mid v \in V[G] \wedge w(v) > 0\}$.

A *path component* in a graph G is a simple path $P = \langle u, s_1, s_2, \ldots, s_q, v\rangle$ where $\deg(s_i) = 2$ for $1 \leq i \leq q$ and $\deg(u), \deg(v) \neq 2$. The vertices u, v are called the *endpoints* of the path component. The vertices s_i are the *inner vertices* of the component. We define the *size of a path component* to be equal to the number q of inner vertices.

We will often consider graphs in which each vertex is colored either black or white. Such a black/white colored graph is a tuple $\langle G = (V, E), c\rangle$ where $c : V \mapsto \{\text{BLACK}, \text{WHITE}\}$ is a coloring function. We denote the set of black vertices as $\mathcal{B}_c(G) := \{v \in V \mid c(v) = \text{BLACK}\}$ and similarly the set of white vertices as $\mathcal{W}_c(G) := \{v \in V \mid c(v) = \text{WHITE}\}$. If $\forall uv \in E : c(u) \neq c(v)$ then we say that G is bipartite according to the coloring c. If v is a vertex in G with $c(v) = \text{BLACK}$ (resp. WHITE), then we say that v is c-black (resp. c-white). We denote the maximum degree of a c-black vertex in $\langle G, c\rangle$ by $\text{BLACKDEG}_c(G) := \max\{\deg_G(v) \mid v \in \mathcal{B}_c(G)\}$.

The oriented genus of a graph G is denoted by $\text{GENUS}(G)$, and is informally defined as the minimum number of handles that an orientable surface S must have such that G can be drawn on S without crossing edges; this implies that planar graphs have a genus of 0. We will not consider the non-orientable graph genus in this work. Refer to [25] for more information on this topic.

2.1 Edge Counts and Low Degree Vertices

In this section we state a few known results about graphs of bounded genus and outerplanar graphs. These properties will be used in the structural analysis of spanning trees with many leaves.

Definition 2. *Let the* Heawood number $H(\gamma)$ *of* $\gamma \geq 0$ *be defined as follows:*

$$H(\gamma) := \left\lfloor \frac{1}{2}\left(7 + \sqrt{1+48\gamma}\right) \right\rfloor.$$

Theorem 1 (Graph Properties).

(i) For a simple graph $G = (V, E)$ *with* $\textsc{Genus}(G) \leq \gamma$ *it must hold [26, Theorem 6.10] that:*

$$|E| \leq 2|V| + 4\gamma \qquad \text{If } G \text{ is bipartite.}$$
$$|E| \leq 3|V| + 6\gamma \qquad \text{Otherwise.}$$

It is also known [26, Lemma 6.12] that if $\textsc{Genus}(G) \leq \gamma$ *and* $\gamma \geq 1$ *then graph* G *must have a vertex of degree at most:*

$$\left\lfloor \frac{1}{2}\left(5 + \sqrt{1+48\gamma}\right) \right\rfloor = H(\gamma) - 1.$$

(ii) Every simple outerplanar graph has a vertex of degree at most 2. [27, Corollary 11.9(a)]

2.2 Structural Properties of Spanning Trees

We present a simple 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 for* 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 for G in which all vertices of S are leaves.

Since S is a cutset, there are two vertices $x, y \notin S$ 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 Bipartite Max Leaf Is $W[1]$-Hard

We will show that Bipartite Max Leaf is hard for $W[1]$ by giving a parameterized reduction from k-Independent Set, which was shown to be complete for $W[1]$ by Downey and Fellows [18]. For completeness we give the formal definition of standard fixed-parameter reducibility, as used in [28].

Definition 3. *Let* $L, L' \subseteq \Sigma^* \times \mathbb{N}$ *be two parameterized problems. We say that* L reduces to L' by a standard parameterized (many-one-)reduction *if there are functions* $k \mapsto k'$ *and* $k \mapsto k''$ *from* $\mathbb{N}$ *to* $\mathbb{N}$ *and a function* $\langle x, k\rangle \mapsto x'$ *from* $\Sigma^* \times \mathbb{N}$ *to* Σ^* *such that*

1. $\langle x, k\rangle \mapsto x'$ *is computable in* $k'' \cdot |\langle x, k\rangle|^c$ *time for some constant* c *and*
2. $\langle x, k\rangle \in L \Leftrightarrow \langle x', k'\rangle \in L'$.

Since the reduction is fairly straight-forward, we shall be slightly less formal when presenting the reduction. The source problem of our reduction is defined as follows:

k-Independent Set
Instance: An undirected graph $G = (V, E)$; a positive integer k.
Question: Is there a set $V' \subseteq V$ of cardinality k such that $\forall u, v \in V' : uv \notin E$?
Parameter: The value k.

Using these definitions we can state the main result of this section.

Theorem 2. Bipartite Max Leaf *is hard for* $W[1]$.

Proof. Given an instance $\langle G, k\rangle$ of k-Independent Set we reduce to an instance $\langle G', k', c'\rangle$ of Bipartite Max Leaf by taking $k' = k$ and building G' and its coloring function $c' : V' \mapsto \{\text{black}, \text{white}\}$ by a series of operations on G:

1. Initialize G' as a copy of G, coloring all vertices black.
2. Replace every edge uv in G' by a vertex w with $c'(w) :=$ white and $N_{G'}(w) := \{u, v\}$.
3. Add a root vertex r with $c'(r) :=$ white to G' and add edges between r and every black vertex.

It is easy to verify that this construction satisfies the time and parameter bounds of a parameterized reduction. It only remains to prove that the transformation does not change the answer to the decision problem. This is established in Lemma 2. □

Lemma 2. *Graph $G = (V, E)$ has an independent set of size at least $k \Leftrightarrow$ graph $G' = (V', E')$ that is bipartite according to coloring function c' has a spanning tree with at least $k' = k$ c-black leaves.*

Proof. We shall use the fact that $\mathcal{B}_c(G') = V$, and hence that every vertex in G is a c-black vertex 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 $|S|$ c-black leaves. We initialize a tree $T \subseteq G'$ by taking the added white vertex r as the root, and adding edges to every black vertex in G'. Every c-black leaf in T now corresponds to a vertex of G. The only vertices that this tree does not reach are the c-white vertices that were added when subdividing edges of G. Since S is an independent set, we know $V \setminus S$ forms a vertex cover for G. By the equivalence between the white subdivider vertices in G' and edges in G, we can augment tree T to a spanning tree by connecting to the white subdivider vertices from the vertices in $V \setminus S$. If we augment T to a spanning tree in this way, every c-black vertex in G' that is a member the independent set S in G remains a leaf. So the augmented T is a spanning tree for G' with at least $|S| \geq k$ c-black leaves.

($\Leftarrow$) Suppose $T \subseteq G'$ is a spanning tree with $|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| \geq k$. Since $\mathcal{B}_c(T) = \mathcal{B}_c(G') = V$, the vertices $\textsc{Leaves}(T) \cap \mathcal{B}_c(T)$ also exist in graph G; we will prove that their form an independent set in that graph. This will be proven by showing that if there is an edge uv in G, then u and v are not both 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 c-white vertex w when forming G'. Since $N_{G'}(w) = \{u, v\}$ it follows that $\{u, v\}$ is a cutset for G' since it separates w from the c-white 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 vertex set $\textsc{Leaves}(T) \cap \mathcal{B}_c(T)$ is an independent set in G. This establishes that G has an independent set of size $|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| \geq k$. □

Corollary 1. Bipartite Max Leaf *does not have a constant-factor approximation algorithm unless $P = NP$.*

Proof. This follows from Theorem 2 since a constant-factor approximation algorithm for Bipartite Max Leaf would imply a constant-factor approximation algorithm for Independent Set by the correspondence shown in Lemma 2. According to [12] this is not possible unless P = NP. □

Corollary 2. Weighted Max Leaf *with weight range S is hard for $W[1]$ if $\{0, 1\} \subseteq S$.*

Proof. This is a direct consequence of Theorem 2 since we can reduce Bipartite Max Leaf to Weighted Max Leaf with $\{0, 1\} \subseteq S$ by giving black vertices a weight of 1 and white vertices a weight of 0.

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

In this section we present a piece of extremal graph theory that will be used for the correctness claim of the kernel for Weighted Max Leaf$_+$.

Definition 4. *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 p-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.

Using this definition we can formulate our claim regarding spanning trees in graphs without long paths.

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

Proof. Consider $G = (V, E) \in \mathcal{C}^{\mathcal{P}}$. If $G = K_1$ or $G = K_2$ then any spanning tree for G has $|G|$ leaves and so the lemma is trivially true. Therefore we assume that $|G| \geq 3$ in the remainder of the proof.

Our proof uses the method of amortized analysis by keeping track of dead leaves, as used by Griggs et. al [29] to lower bound the fraction of leaves in optimal spanning trees for cubic graphs. The proof is constructive, and consists of a series of operations that can be used to initialize a tree $T \subseteq G$, and to expand T if it does not yet span G. We will prove that T has sufficient leaves by showing for every augmentation step of T that the increase in the total size of the tree is balanced against the increase in the number of leaves of the tree. To analyse 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, 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 in $u \in N_G(v)$ 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 all satisfy the following *incremental inequality*:

$$(\mathcal{P} + 3)\Delta L + \left(\frac{\mathcal{P}}{2}\right)\Delta D \geq \Delta N. \tag{2}$$

Assuming for the moment that a set of such augmentation operations exists, and that those operations can always be used to grow T into a spanning tree, we can finish the proof as follows. Assume $T \subseteq G$ is a spanning grown according to the mentioned set of augmentation operations.

$$(\mathcal{P} + 3)L + \left(\frac{\mathcal{P}}{2}\right) D \geq N \qquad \text{Since every step satisfies Equation (2).}$$

$$(\mathcal{P} + 3)|\textsc{Leaves}(T)| + \left(\frac{\mathcal{P}}{2}\right)|\textsc{Leaves}(T)| \geq N \qquad \text{If } T \text{ spans } G \text{ then every leaf is dead.}$$

$$(\mathcal{P} + 3)|\textsc{Leaves}(T)| + \left(\frac{\mathcal{P}}{2}\right)|\textsc{Leaves}(T)| \geq |G| \qquad \text{If } T \text{ spans } G \text{ then } N = |T| = |G|.$$

$$|\textsc{Leaves}(T)| \geq \frac{|G|}{1.5\mathcal{P} + 3} \qquad \text{Rewriting.}$$

This proves the claim under the stated assumptions. So now all that remains to be done is to present the operations that construct the tree, prove that they satisfy the incremental inequality and that they are always sufficient to grow a spanning tree.

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 $\textsc{Leaves}(T) = N_G(v)$. 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 can not be negative because there were no dead leaves before the tree was initialized, so none can be lost. With this information it can easily be seen that the initialization of the tree satisfies the incremental inequality.

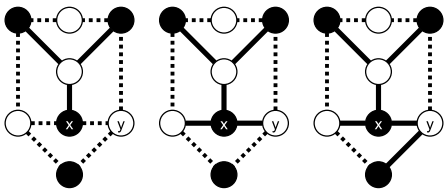


Fig. 1. This sequence shows the effect of successively expanding vertices x and y. Dotted lines indicate edges in the graph, and solid lines represent edges in T.

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

Definition 5 (Vertex Expansion). *To expand a vertex v in a tree $T \subseteq G$ means to add to T all the edges vu such that $u \in N_G(v) \setminus V(T)$.*

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 so there is never a net decrease in the number of leaves. If v has no neighbors in G that are 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 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.

Let us now consider the operations that augment a 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.

1. If there is a live leaf $v \in V(T)$ with $a \geq 2$ neighbors outside T, then expand v. Since $\Delta L = a - 1$, $\Delta D \geq 0$ and $\Delta N = a$ this satisfies the incremental inequality.
   If this rule can not be applied, then all live leaves of T have exactly one neighbor outside T.
2. If there is a vertex $v \notin V(T)$ with at least 2 neighbors x, y inside T, then expand the neighbor x. Since x is a live leaf it must have exactly one neighbor outside T by our earlier observation. So $\Delta L = 0$ and $\Delta N = 1$. Now note that vertex y must have had only one neighbor outside T, because it is a live leaf and the first operation is not applicable. So y only has the vertex v as a neighbor that is not in T, which means that all its neighbors will be in T after the expansion of x. Since the expansion of x does not change the fact that y is a leaf we find that y must have become a dead leaf by the expansion of x and therefore $\Delta D \geq 1$, which implies that this operation satisfies the incremental inequality.
   If this rule can not be applied, then all vertices outside T have at most one neighbor in T.
3. Combining the observations made so far, we find that if the first two rules can not be applied then any $v \in N_G(V(T))$ cannot have a degree of 3 or higher (since such a vertex of degree 3 either has two neighbors in T, or two neighbors outside T). If there is a $v \in N_G(V(T))$ with $\deg_G(v) = 1$, then expand its neighbor in T. For this operation we then find that $\Delta L = 0$, $\Delta D = 1$ and $\Delta N = 1$ which satisfies the incremental inequality since $\mathcal{P} \geq 2$.
   Incorporating this operation in our analysis, we find that if none of the operations so far are applicable then every $v \in N_G(V(T))$ has $\deg_G(v) = 2$.
4. Consider a vertex $v \in N_G(V(T))$. It must have a degree of 2 in G, with one neighbor in T and the other neighbor outside T. Let v' be the neighbor of v that is in T. We now look at the maximum path $P = \langle v = v_1, v_2, \ldots, v_q \rangle$ where $\deg_G(v_i) = 2$ and $v_i \notin V(T)$ for all $1 \leq i \leq q$. Since the size of a path component in G is bounded by $\mathcal{P}$ we know that $q \leq \mathcal{P}$. Let $\{u\} = N(v_q) \setminus (\{v'\} \cup \{v_1, \ldots, v_{q-1}\})$. We now distinguish based on the situation of u.
   (a) If $u \in V(T)$ then based on the above observations we know that u has exactly one neighbor not in T. We now expand the vertices $v_1, v_2, \ldots, v_q$. This expansion will kill vertex u since it adds v_q to T, and since v_q has degree 2 it will also be a dead leaf after the expansions. So we find that $\Delta L = 0$, $\Delta D = 2$ and $\Delta N \leq \mathcal{P}$, which satisfies the incremental inequality.

(b) If $u \notin V(T)$ then from the definition of u it follows that $\deg_G(u) \neq 2$. Since no degree-1 vertex is adjacent in G to a degree-2 vertex by Property (iv) of Definition 4, we find that $\deg_G(u) \geq 3$. By the earlier observations this implies that u is not adjacent to T. To augment the tree in this case, we expand $v_1, v_2, \ldots, v_q, u$. All neighbors of u except v_q become leaves after these expansions, and v_1 becomes an internal node, which means $\Delta L = \deg_G(u) - 2$ and $\Delta N = q + \deg_G(u)$. As always we have $\Delta D \geq 0$, which implies that this operation also satisfies the incremental inequality.

Since this exhausts all possibilities for the status of u we have established that whenever augmentation steps 1, 2 and 3 are not applicable, then one of the subcases of augmentation step 4 must be applicable. This proves that as long as T does not span G, there is an operation that is applicable to augment T, which shows that the given operations are always sufficient to grow T to a spanning tree.

Since all operations satisfy the incremental inequality, this concludes the proof of Theorem 3. □

Theorem 4. *Let $\mathcal{P} \geq 2$ and let graph $G = (V, E) \in \mathcal{C}^{\mathcal{P}}$. We can find a spanning tree $T \subseteq G$ with at least $|G|/(1.5\mathcal{P} + 3)$ leaves in $O(|V| + |E|)$ time.*

Proof. To prove this claim we show how the constructive proof of Theorem 3 can be turned into a linear time algorithm. If $|G| \leq 2$ then the graph itself is a tree that satisfies the requirements, so in the remainder we assume that $|G| \geq 3$ which implies by Definition 4 that G must have a vertex of degree at least 3. We find such a vertex v with $\deg_G(v) \geq 3$ and use it to initialize the tree T as the star rooted at v with edges to all its neighbors in G. The algorithm maintains the following sets as doubly-linked lists:

- the set L consisting of live leaves of the tree,
- the set L^* consisting of live leaves of the tree that have at least 2 neighbors outside T,
- the set N of vertices in $N(V(T))$ that have at least 2 neighbors in T.

We store for each $v \in V$ the value $|N_G(v) \cap V(T)|$, which is updated at the same time that the three lists are updated. When a vertex is in a list, we store a reference to its node in the list with the vertex so that it may be removed from the list in constant time. This allows us to update these three lists and the values $|N_G(v) \cap V(T)|$ in $O(\deg_G(v))$ time when adding a vertex v to T.

Using these lists, the algorithm proceeds as follows. As long as L^* is not empty it removes a vertex v from L^* and expands it, adding all vertices in $N_G(v) \setminus V(T)$ to the tree. If L^* is empty but N is not, then the algorithm takes a $n \in N$ and expands a vertex in $N_G(n) \cap V(T)$. If L^* and N are both empty then all vertices in $N_G(V(T))$ must have degree 2 in G. The algorithm then selects a live leaf vertex in L, finds its single degree-2 neighbor u outside T and expands all the vertices on the path outside T that starts in u, which can be done by simply traversing the path.

We spend $O(\deg_G(v))$ time on a vertex v when it is added to the tree since we need to update its membership in the lists, and for every neighbor u we have to decrease the stored value $|N_G(u) \cap N_T(w)|$. We also spend $O(\deg_G(v))$ time on a vertex v when expanding it, since we need to inspect all its neighbors. Since every vertex is added to T exactly once, and every vertex is expanded at most once, the total running time is $\sum_{v \in V} O(\deg_G(v)) = O(|V| + |E|)$. □

5 Spanning Trees with Many Leaves in Bipartite Graphs

In this section we give a result in the style of extremal graph theory that proves that for graphs that are bipartite according to a black/white coloring that satisfies a few simple properties, there is always a spanning tree in which the number of black leaves is lower bounded by a constant fraction of the total number of vertices. The proof of this result consists of two stages. First we show in the proof of Theorem 5 that for the given class of graphs there is always a spanning tree with a constant fraction of the *black* vertices as leaves. We then show in Lemma 7 that the number of white vertices can be bounded by the number of black vertices, which implies that the number of black vertices is at least some constant fraction of the total graph size.

5.1 Definitions and Proof Setup

In this section we define the class of graphs to which our results apply, and set up the basic ingredients for the proof by amortized analysis.

Definition 6. *Let $\mathcal{C}^b$ be the class of black/white colored graphs $\langle G = (V, E), c\rangle$ that satisfy the following properties:*

(i) Graph G is simple and connected.
(ii) G is not isomorphic to a simple cycle.
(iii) G is bipartite according to the coloring function c.
(iv) The minimum degree of G is at least 2.
(v) If u, v are two distinct white degree-2 vertices in G, then $N_G(u) \neq N_G(v)$.
(vi) The maximum size of a p-component in G is at most 4.

Using this definition we can state the main structural result on spanning trees in bipartite graphs.

Theorem 5. *Let $\langle G = (V, E), c\rangle \in \mathcal{C}^b$ such that $\textsc{blackdeg}_c(G) \geq 3$. There is a spanning tree for G with at least $|\mathcal{B}_c(G)|/(5 + 4\beta)$ black leaves, with $\beta := \min(H(\textsc{Genus}(G)) - 2, \textsc{blackdeg}_c(G) - 1)$.*

Proof. The main structure of the proof is similar to the proof of Theorem 3. Once again we give a method that can initialize and augment a spanning tree, and analyze the number of leaves in this tree by showing that every augmentation step satisfies an incremental inequality that balances the increase in the size of the tree against the increase in the number of live and dead leaves. Whereas Theorem 3 makes a claim about the total number of leaves (regardless of their structure), in this theorem we are only interested in finding leaves that are colored black by the coloring function c. To simplify the notation of vertex colors we adopt the following convention. Throughout this proof, any mention of a vertex color that does not specify a coloring function, refers to the coloring function c of the graph G for which we are constructing a spanning tree.

The focus on black leaves requires us to keep track of different properties of the tree $T \subseteq G$ that we are constructing. We use the following properties of T in our analysis:

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

Observe how the analysis is tailored towards proving the existence of a spanning tree with many black leaves, by measuring only the black vertices. In the proof of Theorem 3 we could use the fact that the expansion of a vertex cannot cause the number of leaves in the tree to decrease, which implied $\Delta L \geq 0$. In this proof the inequality $\Delta L \geq 0$ does no longer hold for all expansion steps: although the *total* number of leaves cannot decrease, the number of *black* leaves *can* decrease by the expansion of a vertex! Therefore we will often perform a number of successive expansions in a single augmentation operation, to ensure that every augmentation yields a sufficient increase in the number black leaves. Similar to the earlier proof we have $\Delta D \geq 0$ because once a black leaf is dead, it can no longer become an internal node by some vertex expansion; it must remain dead.

We will use the following *incremental inequality* for this proof:

$$4\Delta L + (4\beta + 1)\Delta D \geq \Delta B \tag{3}$$

We will show that there is a series of augmentation operations that all satisfy this incremental inequality, and that are sufficient to grow any properly initialized tree $T \subseteq G$ into a spanning tree for G. Assuming for the moment that we can indeed grow T to a spanning tree while satisfying the incremental inequality at each step, we can finish the proof as follows. Assume T is a spanning tree grown according to the mentioned set of operations.

$$\begin{aligned}
4L + (4\beta + 1)D &\geq B && \text{Since every step satisfies Equation (3).}\\
4|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| + (4\beta + 1)|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| &\geq B && \text{If } T \text{ spans } G \text{ then every leaf is dead.}\\
(5 + 4\beta)|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| &\geq B && \text{Simple arithmetic.}\\
(5 + 4\beta)|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| &\geq |\mathcal{B}_c(T)| && \text{If } T \text{ spans } G \text{ then } B = |\mathcal{B}_c(T)|.\\
|\textsc{Leaves}(T) \cap \mathcal{B}_c(T)| &\geq \frac{|\mathcal{B}_c(T)|}{5 + 4\beta} && \text{Rewriting.}
\end{aligned}$$

This will prove the theorem. So now all that remains to be done is to give the expansion operations, prove that they satisfy the incremental inequality and that they are sufficient to obtain a spanning tree.

5.2 Tree Initialization

We initialize T by taking a white vertex w as the root and adding edges to all its neighbors, making them leaves.

Lemma 3. *Every tree initialization satisfies the incremental inequality.*

Proof.

$$\begin{aligned} 4\Delta L + (4\beta+1)\Delta D &\geq 4\Delta L && \text{Since } \Delta D \geq 0. \\ &\geq 4|N_G(w)| && \text{All vertices in } N_G(w) \text{ become black leaves.} \\ &\geq |N_G(w)| \\ &= \Delta B && N_G(w) \text{ are the only black vertices in } T. \end{aligned}$$

This concludes the proof. □

5.3 Extending the Tree

We introduce some terminology to describe the operations that extend the initial tree to a spanning tree. We will often often expand a sequence of vertices that are on a simple path. A *grow path* P is a simple path in G that starts in vertex $v \in \textsc{LiveLeaves}_G(T)$ and does not visit any other vertices of T. We say that the *size* $s(P)$ of the grow path is the number of black vertices that it visits *outside* T, so $s(P) := |(V(P) \setminus V(T)) \cap \mathcal{B}_c(G)|$.

Definition 7 (Path Expansion). *To expand a grow path P means to successively expand all vertices on P, starting with the vertex on P that is a live leaf of the tree T. If these expansions add white vertices to T that are not on P, then expand these vertices in arbitrary order.*

The expansion of a grow path P will cause new vertices to be added to the tree. We define the *yield* $y(P)$ of a grow path P as the set of black vertices that are leaves of T after expansion of P but were outside T before the expansion. If $v \in y(P)$ then we say that expansion of P yields v. Using this terminology we present the expansion operations that grow the tree. We will ensure that all live leaves of the tree are black, by expanding every white vertex in the same operation that adds it to the tree. More formally, this means that the following statement is an invariant of the construction procedure: $\textsc{LiveLeaves}_G(T) \subseteq \mathcal{B}_c(G)$. This invariant will be used later, in the proof that the given list of operations is always sufficient to grow a spanning tree.

Expansion Operation 1 (Scout Operation) If there is a grow path P of size at most 2 such that $|y(P)| \geq 2$ then expand P.

Lemma 4. *Every Scout operation satisfies the incremental inequality.*

Proof. Since we lose at most one existing black leaf (the live leaf where P starts), and we gain all vertices in $y(P)$ as black leaves by definition, we have $\Delta L \geq |y(P)| - 1$. The only black vertices that are added to T and might not become leaves are those on P, so there are at most $s(P) \leq 2$ of them which implies $\Delta B \leq y(P) + s(P)$. Now observe:

$$\begin{aligned} 4\Delta L + (4\beta+1)\Delta D &\geq 4\Delta L && \text{Since } \Delta D \geq 0. \\ &\geq 4(|y(P)| - 1) && \Delta L \geq |y(P)| - 1 \text{ as argued.} \\ &\geq |y(P)| + 3|y(P)| - 4 && \text{Expanding.} \\ &\geq |y(P)| + 6 - 4 && \text{Since } |y(P)| \geq 2. \\ &\geq |y(P)| + s(P) && \text{Since } s(P) \leq 2. \\ &\geq \Delta B && \Delta B \leq y(P) + s(P) \text{ as argued.} \end{aligned}$$

This concludes the proof. □

Expansion Operation 2 (Sandwich operation) If there are vertex-disjoint grow paths P_1, P_2 of size at most 1 and a black vertex $b \notin V(P_1) \cup V(P_2) \cup V(T)$ such that all vertices in $N_G(b)$ would be added to T after expansion of P_1 and P_2, then expand P_1 and P_2 in arbitrary order.

Lemma 5. *Every Sandwich operation satisfies the incremental inequality.*

Proof. Let N be the set of black leaves that are added to T by this operation. We expand two grow-paths and thus lose at most two existing black leaves: $\Delta L \geq |N| - 2$. The grow paths we expand each have size at most 1, so at most 2 black vertices not in N are added to T by the operation: $\Delta B \leq |N| + 2$. Vertex b is not in T before the operation. Since it is black, all its neighbors are white. All such neighbors are added to T by the operation, which implies by the definition of the expansion of a grow path that they will also be expanded. Therefore vertex b is added to T by this operation, and since its neighbors are also added to T it must be dead after the operation. This implies that $|N|, |D| \geq 1$. Now observe:

$$\begin{aligned}
4\Delta L + (4\beta+1)\Delta D &\geq 4\Delta L + (4\beta+1) && \text{Since } \Delta D \geq 1.\\
&\geq 4\Delta L + 9 && \beta \geq 2 \text{ by definition.}\\
&\geq 4(|N| - 2) + 9 && \text{Since } \Delta L \geq |N| - 2.\\
&\geq 4|N| + 1 && \text{Simplifying.}\\
&\geq |N| + 3|N| + 1 && \text{Splitting.}\\
&\geq |N| + 3 + 1 && \text{Since } |N| \geq 1.\\
&\geq \Delta B && \text{Since } \Delta B \leq |N| + 2.
\end{aligned}$$

This concludes the proof. □

Expansion Operation 3 (Surround Operation) If there is a vertex $b \in \textsc{LiveLeaves}_G(T)$, a set $S \subseteq \textsc{LiveLeaves}_G(T)\backslash\{b\}$ with $|S| \leq \beta$ and a set $S' \subseteq N_G(N_G(S))\backslash V(T)$ with $|S'| \leq 1$ such that $N_G(b)\backslash V(T) \subseteq N_G(S \cup S')$ then expand the vertices in S, in S' and in $N_G(S \cup S')$.

Lemma 6. *Every Surround operation satisfies the incremental inequality.*

Proof. We will first give some intuition for this operation. The idea is that we select suitable vertices in S and S', so that all the neighbors of b that are not in T can be reached from a vertex in $S \cup S'$. This will ensure that all the neighbors of b that are not yet in T will be added to T after the mentioned expansions. Afterwards all the neighbors of b are in T and so the leaf b has been killed. Let us now analyze this operation.

Let N be the set of black leaves that are added to T by this operation. Since the set S' contains the only black vertices not yet in T that are possibly expanded by the operation and $|S'| \leq 1$, there is at most 1 black vertex not in N that is added to T by the operation (the other black vertices added to T must end up in N). This implies that $\Delta B \leq |N| + 1$. Since we expand at most β grow paths, we lose at most β existing live leaves and therefore $\Delta L \geq |N| - \beta$. Since the leaf vertex b is killed by this operation we have $\Delta D \geq 1$. Now observe:

$$\begin{aligned}
4\Delta L + (4\beta+1)\Delta D &\geq 4\Delta L + (4\beta+1) && \text{Since } \Delta D \geq 1.\\
&\geq 4(|N| - \beta) + (4\beta+1) && \text{Since } \Delta L \geq |N| - \beta.\\
&\geq 4|N| + 1 && \text{Canceling } 4\beta.\\
&\geq \Delta B && \text{Since } \Delta B \leq |N| + 1.
\end{aligned}$$

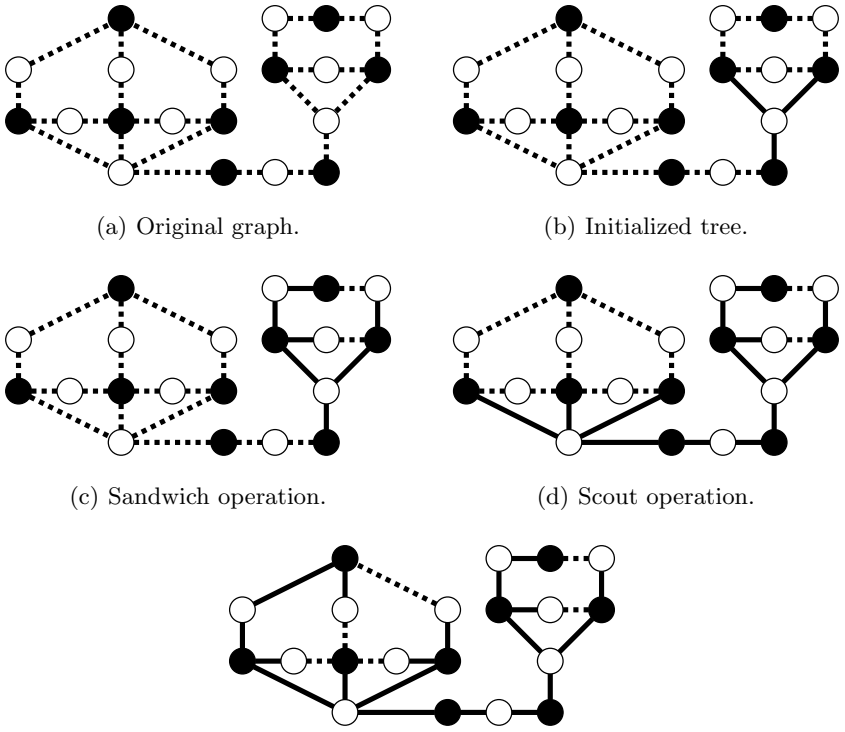
This concludes the proof. □

Figure 2 gives an example of the application of each of the expansion operations.

5.4 Applicability of the Expansion Rules

We will show that the given set of expansion rules is sufficient to grow any tree into a spanning tree, by proving that a Surround operation must be applicable whenever the Scout and Sandwich operations are not. The proof of this claim is fairly involved. Therefore we start by deriving some properties of the structure of the graph when the Scout and Sandwich operations are not applicable.

First observe that every white vertex in T is expanded by the same operation that adds it to the tree, which implies that every live leaf of the tree must be black. We introduce some new concepts to make it easier to speak of the structure near the boundary of the tree.



(a) Original graph. (b) Initialized tree.

(c) Sandwich operation. (d) Scout operation.

(e) Surround operation.

Fig. 2. Example of the steps that grow a spanning tree. Solid lines represent edges in the tree and dotted lines represent edges in the graph.

Definition 8. *For a black vertex $b \notin V(T)$ we define the tree-neighborhood of b as $N^{\mathcal{T}}_{G,T}(b) := N_G(N_G(b)) \cap V(T)$. We define the border-neighborhood of b as $N^{\mathcal{B}}_{G,T}(b) := N_G(b) \cap N_G(V(T))$, which can also be interpreted as the neighbors of b that are adjacent to T. We define the set of* connector vertices *as the vertices with a border-neighborhood consisting of at least 2 vertices, i.e. the set $\{b \in V \mid b \notin V(T) \wedge |N^{\mathcal{B}}_{G,T}(b)| \geq 2\}$.*

Lemma 7. *If no Scout or Sandwich operation is applicable, then the following must hold:*

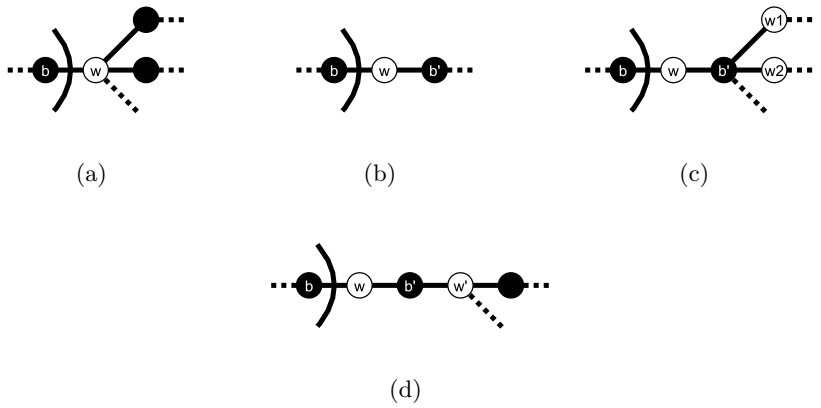
1. *Every black vertex in T is in the tree-neighborhood of at most one connector vertex.*
2. *Every white vertex in $N_G(V(T))$ has at most one black neighbor outside T.*

Proof. (1) Proof by contraposition. If $b^* \in V(T)$ is in at least two different tree-neighborhoods for black connector vertices $b, b' \notin V(T)$ then expansion of the grow path $P = \langle b^* \rangle$ must yield b and b' as new black leaves which implies that a Scout operation is applicable over grow path P.

(2) Proof by contraposition. Assume $w \in N_G(V(T))$ is adjacent to $b', b'' \notin V(T)$ with $b' \neq b''$. Since w is adjacent to T, there is some $b \in V(T)$ with $w \in N_G(b)$. Consider the grow path $P = \langle b, w \rangle$. Expansion of P must yield b', b'' as new black leaves, and since P has a size of 0 a Scout operation is applicable over grow path P. □

Lemma 8. *If w, w' are two distinct white vertices in $\langle G, c \rangle \in \mathcal{C}^b$, then $|N_G(w) \cup N_G(w')| \geq 3$.*

Proof. Assume there are $w, w' \in \mathcal{W}_c(G)$, $w \neq w'$ with $\langle G, c \rangle \in \mathcal{C}^b$ such that $|N_G(w) \cup N_G(w')| < 3$. We will derive a contradiction. Since w, w' have degree at least 2 by Property (iv) of Definition 6, this situation can only arise if $N_G(w) = N_G(w')$ and $|N_G(w)| = |N_G(w')| = 2$. But then w and w' are white vertices of degree 2 with the same neighborhood, which violates Property (v). This implies that $\langle G, c \rangle \notin \mathcal{C}^b$, which is a contradiction. □



(a) (b) (c)

(d)

Fig. 3. Situations as encountered in the proof of Theorem 5. The tree T is schematically represented by the arc; all vertices to the left of the arc are in T, and all vertices to the right of the arc are outside T. Dotted lines represent possible edges in the graph, and solid lines represent edges that must be in G.

Lemma 9. *If the Scout and Sandwich operations are not applicable, then every vertex $w \in N_G(V(T))$ that does not have two neighbors in T is adjacent to a connector vertex.*

Proof. Suppose there is a vertex $w \in N_G(V(T))$ that does not have two neighbors in T. We will prove that either a Scout or Sandwich operation is applicable, or w is adjacent to a black vertex $b \notin V(T)$ with at least 2 neighbors adjacent to T (a connector vertex). Since $w \in N_G(V(T))$ we know w must have at least one neighbor in T. Vertex w does not have two neighbors in T (by assumption), so we can conclude that w has exactly one neighbor in T; let this be $b \in V(T)$. If $\deg_G(w) \geq 3$ then w must have at least two black neighbors outside T (as depicted in Figure 3(a)) which means a Scout operation is applicable over the grow path $P = \langle b, w\rangle$. So in the remainder of the proof we can assume that the degree of w is at most 2. Since every graph in $\mathcal{C}^b$ has minimum degree at least 2, we can conclude that w must have degree exactly 2. Let $\{b'\} = N_G(w) \setminus \{b\}$. Then the situation must be as depicted in Figure 3(b).

If b' has at least two neighbors adjacent to T then b' is a connector vertex by definition. Since $b' \in N_G(w)$ the claim then follows. So in the remainder we may assume that b' has exactly one neighbor adjacent to T, which is w. If b' has degree at least 3 as shown in Figure 3(c) then b' has at least two neighbors w_1, w_2 that are not adjacent to T, so by Lemma 8 it must hold that $|(N_G(w_1) \cup N_G(w_2)) \setminus \{b'\}| \geq 2$ and so a Scout operation is applicable over grow path $P = \langle b, w, b'\rangle$ since $(N_G(w_1) \cup N_G(w_2)) \setminus \{b'\} \subseteq y(P)$.

We can conclude that the degree of b' must be 2 in the remaining cases. Let $\{w'\} = N_G(b') \setminus \{b'\}$. By our earlier argument, vertex w' is not adjacent to T and the situation must be as depicted in Figure 3(d).

Consider the vertices on the simple path that starts with w, b', continues over all degree 2 vertices and ends with the first encountered vertex that is in T, or has a degree of at least 3. Number the vertices on this path as $v_0 = w, v_1 = b', \ldots, v_q$. Since any path component in $G \in \mathcal{C}^b$ has size at most 4 it follows that $q \leq 4$.

- If the path ended because it encountered a vertex in the tree (so $v_q \in V(T)$) then we can do a Sandwich operation over the grow paths $P_1 = \langle b, w = v_0\rangle$ and $P_2 = \langle v_q, v_{q-1}, \ldots, v_2\rangle$ that will kill vertex $v_1 = b'$. The grow path P_1 obviously has a size of 0 according to the definition. Note that the size of P_2 is at most one, since $q \leq 4$ and only odd-numbered vertices are black. So vertex v_3 is the only possible black vertex on P_2.
- If the path did not end because it encountered a vertex in T, then the last vertex v_q has a degree of at least 3 and is outside of T. We make a distinction based on the value of q.
  - If q is even then v_q must be a white vertex since the colors on the path alternate and $w = v_0$ is white.
    - If v_q is adjacent to $x \in V(T)$, then we can do a Sandwich operation over the grow paths $P_1 = \langle b, w = v_0\rangle$ and $P_2 = \langle x, v_q, v_{q-1}, \ldots, v_2\rangle$ that will kill vertex $v_1 = b'$. As before we find $s(P_1) = 0$ and $s(P_2) \leq 1$ since the only black vertex on P_2 outside T is v_3; this implies that the given grow paths indeed satisfy the requirements of a Scout operation.
    - If v_q is not adjacent to T, then we can do a Scout operation over the grow path $P = \langle b, v_0, v_1, \ldots, v_q\rangle$. Since $N_G(v_q) \setminus \{v_{q-1}\} \subseteq y(P)$ and the degree of v_q is at least 3, this grow path yields at least 2 new black leaves. Since $q \leq 4$ there are at most 2 black vertices v_1, v_3 outside T on this grow path which shows that $s(P) \leq 2$.

- If q is odd then $q \leq 3$ and v_q is black. Since all leaves of T are black the vertex v_q is not directly adjacent to T.
  - If v_q has a neighbor w^* that is adjacent to some vertex $x \in V(T)$, then we can do a Sandwich operation over the grow paths $P_1 = \langle b, w = v_0 \rangle$ and $P_2 = \langle x, w^*, v_q, v_{q-1}, \ldots, v_2 \rangle$. Since $q \leq 3$ in this case, the grow path P_2 has a size of 1 and the only black vertex that is outside of T on the grow path is the vertex v_q.
  - If none of the neighbors of v_q are adjacent to T, then we can do a Scout operation over the grow path $P = \langle b, v_0, v_1, \ldots, v_q \rangle$. Since v_q has degree at least 3, it must have at least two neighbors $w_1, w_2 \neq v_{q-1}$. Since w_1, w_2 are not adjacent to T, all their neighbors except v_q will become leaves of T after expansion of P. By Lemma 8 this implies that expansion of P yields at least two new black leaves. Since $q \leq 3$ in this case, the size of grow path P is at most 2 since the only black vertices outside T it can contain are v_1 and v_3.

Since we have exhausted all possibilities, this concludes the proof. □

Lemma 10. *If the Scout and Sandwich operations are not applicable and tree T does not span G then for every live leaf $b \in \text{LIVELEAVES}_G(T)$ there is a set S and a set S' such that:*

- $S \subseteq \text{LIVELEAVES}_G(T) \setminus \{b\}$,
- $|S| \leq \text{BLACKDEG}_c(G) - 1$,
- $S' \subseteq N_G(N_G(S)) \setminus V(T)$,
- $|S'| \leq 1$,
- $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$.

Proof. Assume that the Scout and Sandwich operations are not applicable, and T does not span G. Consider a live leaf $b \in \text{LIVELEAVES}_G(T)$. We will construct the sets S and S' and show that they satisfy $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$.

Since any tree is initialized with at least 3 vertices, vertex b has at least one neighbor in T. Since its degree is bounded by $\text{BLACKDEG}_c(G)$ it can have at most $\text{BLACKDEG}_c(G) - 1$ neighbors outside T.

We initialize S and S' as empty sets. We fill them by inspecting each $w \in N_G(b) \setminus V(T)$. To analyze how w should be treated we use the set $X = (N_G(w) \cap V(T)) \setminus \{b\}$.

- If $X \neq \emptyset$, then add an arbitrary vertex $x \in X$ to B. This ensures that the neighbor $w \in N_G(b)$ can be reached from $x \in S$.
- If $X = \emptyset$, then w has only one neighbor in T which implies by Lemma 9 that w is adjacent to some connector vertex $b^* \notin V(T)$. By Lemma 7 there is only one connector vertex b^* such that b is in its tree-neighborhood, which means that all the vertices w that we consider in this step must be adjacent to the same connector vertex b^*. We now assign $S' := \{b^*\}$. By the definition of a connector vertex there is some $b' \neq b$ such that $w \in N_G(b') \cap N_G(b^*)$. We add such a b' to the set S, which ensures that $S' \subseteq N_G(N_G(S)) \setminus V(T)$. The inclusion of b^* in the set S' means that we can reach the neighbor w from the vertex b^*.

For each w we have argued that one of its neighbors is added to S or S', which shows that $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$ holds. Since b has at most $\text{BLACKDEG}_c(G) - 1$ neighbors outside T, we consider at most $\text{BLACKDEG}_c(G) - 1$ different vertices as w in the distinction above. Since each step of the distinction adds one vertex to S, the size of S is bounded as required. The set S' is either empty or a singleton set, which shows that all requirements are satisfied. □

Definition 9. *We define the concept of a* covering graph *for the situation that no Scout or Sandwich operations are applicable. A graph G_C is a covering graph of G with respect to tree T if the following conditions hold:*

1. $V(G_C) = \text{LIVELEAVES}_G(T) \cup \{r\}$ *for some* root *vertex r,*
2. $N_{G_C}(r) = \text{LIVELEAVES}_G(T)$,
3. *for all $b \in \text{LIVELEAVES}_G(T)$ it holds for all $w \in N_G(b) \setminus V(T)$ that:*
   (a) *if $|N_G(w) \cap V(T)| \geq 2$, then in graph G_C the vertex b has a neighbor $b' \in \text{LIVELEAVES}_G(T) \setminus \{b\}$ such that $b' \in N_G(w) \cap V(T)$,*
   (b) *if $|N_G(w) \cap V(T)| = 1$ then by Lemma 9 vertex w is adjacent in G to a connector vertex b^*, which implies that b is in the tree-neighborhood of b^*; we demand that vertex b has an edge in G_C to a vertex $b' \neq b$ that is also in the tree-neighborhood of b^*.*

In the construction of the covering graph we need the concept of collapsing vertices.

Definition 10. *Let $G = (V, E)$ and assume we have a minimum-genus embedding of G. To* collapse the vertex $x \in V$ *means to take the following steps:*

a. For every $u, v \in N_G(x)$ with $u \neq v$ such that the edges xu and xv are adjacent in the cyclic permutation of edges around x (as defined by the embedding), add an edge uv to G if this edge does not exist already.
b. Remove vertex x from G.

Lemma 11. *If G is a simple, connected graph and G' is the result of collapsing some vertex $x \in V(G)$ then* $\textsc{Genus}(G') \leq \textsc{Genus}(G)$ *and G' is also simple and connected.*

Proof. (G' is simple) Since the collapse step only adds edges that do not already exist, such a step does not introduce parallel edges and hence the graph remains simple.

(G' is connected) Graph G is connected. We will show that if there is a path P between two vertices $a, b \neq x$ in G, then there is also such a path P' in G'. If P does not use vertex x then this is obvious since the collapse operation does not delete any vertices except x, and does not delete any edges not incident on x. If P does use vertex x then we can always make an alternative path P' in G' because G' contains a cycle that visits all vertices in $N_G(x)$: such a cycle is created by the first step of the collapse operation. So if the sequence $\langle u, x, v\rangle$ is on P, we can replace the occurence of x by the traversal of a suitable part of the cycle through $N_G(x)$.

($\textsc{Genus}(G') \leq \textsc{Genus}(G)$) If the collapse operation adds an edge uv to G, then in the used embedding of G the edges xu and xv are adjacent in the cyclic permutation of edges around x. This implies that there is face such that u and v are both on its boundary; hence the edge uv can be drawn in that face without causing any edges to cross. Since G' has an embedding on the same surface as on which G has a minimum-genus embedding, it follows that $\textsc{Genus}(G') \leq \textsc{Genus}(G)$. □

Lemma 12 (Covering Graph Construction). *If no Scout or Sandwich operation is applicable then there is a covering graph G_C with* $\textsc{Genus}(G_C) \leq \textsc{Genus}(G)$.

Proof. For the construction of the covering graph G_C we assume we have a minimum-genus embedding of G. Refer to Appendix A for an illustration of the following steps.

1. Initialize the graph $G_C \subseteq G$ as the subgraph of G that is induced by the vertices in T, the vertices in $N_G(V(T))$ and by the connector vertices of G. More formally:

$$G_C := G[V(T) \cup N_G(V(T)) \cup \{v \in V \setminus V(T) \mid |N^{\mathcal{B}}_{G,T}(b)| \geq 2\}]. \tag{4}$$

   Let $T_C \subseteq G_C$ denote the tree in G_C that corresponds to T in G. We adopt the convention that if we modify T_C, for example by contracting an edge, then we perform this same operation in G_C and vice versa. This ensures that T_C will stay a tree-subgraph of G_C.
2. Contract all edges between vertices in T_C that are not in the set $\textsc{LiveLeaves}_G(T)$. Afterwards there is a single vertex in T_C that is not in the set $\textsc{LiveLeaves}_G(T)$: we use this vertex as r as mentioned in Property (1) of Definition 9.
3. Collapse every vertex in the set $X := N_{G_C}(V(T_C))$. Since all live leaves of the tree are black, the vertices in X are adjacent to live leaves, and graph G is bipartite according to the coloring function we know that the vertices in the set X are white vertices in graph G. This implies that there are no edges between vertices in X, and therefore the order in which we collapse them does not matter.
   After these collapse operations the only vertices remaining in G_C are the root vertex r, the vertices $\textsc{LiveLeaves}_G(T)$ and the vertices that are connector vertices in G. It can be verified that Property (3a) of Definition 9 holds after this step.
4. Collapse every vertex in the set $Y := N_{G_C}(V(T_C))$. Note that $X \neq Y$ since the graph was modified by the previous step; every vertex in Y is a connector vertex in G. Since all connector vertices of G are black, there were initially no edges between the vertices in Y. No such edge was added by the collapse of vertices in X since every vertex $x \in X$ is adjacent in G (and hence in G_C at the beginning of step 3) to at most one black vertex outside T by Lemma 7.
   After these collapse operations Property (3b) holds for G_C, since every vertex in Y must have at least two neighbors of the right form in G_C at the beginning of this step. Collapsing the vertices in Y caused all vertices in Y to be removed from G_C. This ensures that the only vertices remaining in G_C after this step are the vertices $\textsc{LiveLeaves}_G(T)$ and the root vertex r. This implies that the graph G_C now satisfies Property (1) and Property (2).

The resulting graph G_C satisfies all the conditions of a covering graph. It was built by first taking an induced subgraph of G_C, then taking a minor of it and finally performing a series of collapse operations. Since none of these operations increase the genus we find that $\textsc{Genus}(G_C) \leq \textsc{Genus}(G)$. □

Lemma 13. *If no Scout or Sandwich operation is applicable and T does not yet span G, then in a covering graph G_C with $\textsc{Genus}(G_C) \leq \textsc{Genus}(G)$ there is a vertex $b \in V(G_C) \cap \textsc{LiveLeaves}_G(T)$ such that b has at most $H(\textsc{Genus}(G)) - 2$ neighbors in the graph $G_C - \{r\}$.*

Proof. Assume that the conditions in the lemma hold, and let G_C be a suitable covering graph. We make a case distinction on the genus of G, which is needed because Theorem 1 does not apply for planar graphs of genus 0.

- If G is planar then $\textsc{Genus}(G) = 0$ and $H(0) - 2 = 2$. By Definition 9 the covering graph G_C has a vertex r such that all other vertices of G_C are adjacent to r. Since G is planar and $\textsc{Genus}(G_C) \leq \textsc{Genus}(G)$ the graph G_C must be planar as well. This implies that $G_C - \{r\}$ is outerplanar. To see this, consider a plane embedding of G_C. Since every vertex is adjacent to r, every vertex must be on the boundary of the face that results from the removal of r. By the classical balloon-argument we know that for every planar embedding and face F in the embedding, there exists an embedding in which F is the outer face. Thus there is an embedding of $G_C - \{r\}$ such that all vertices are on the outer face, which proves that $G_C - \{r\}$ is outerplanar. With this knowledge we can apply Lemma 1 to find that $G_C - \{r\}$ has a vertex v of degree at most 2. So v has at most $2 = H(\textsc{Genus}(G)) - 2$ neighbors in $G_C - \{r\}$.
- In the remaining case we know that $\textsc{Genus}(G) \geq 1$ and $\textsc{Genus}(G_C) \leq \textsc{Genus}(G)$. By Theorem 1 we find that G_C has a vertex of degree at most $H(\textsc{Genus}(G)) - 1$. If the low degree vertex is r, then since all other vertices are adjacent to r we know that any of those other vertices has at most $H(\textsc{Genus}(G)) - 2$ neighbors in $G_C - \{r\}$, which proves the claim. So in the remainder assume that there is a vertex $v \neq r$ of degree at most $H(\textsc{Genus}(G)) - 1$ in G_C. Since v is adjacent to r this accounts for one of the edges incident on v, which shows that v has at most $H(\textsc{Genus}(G)) - 2$ neighbors in $G_C - \{r\}$.

Since we have exhausted all possibilities for the genus of G this concludes the proof. □

Theorem 6 (Covering Application). *If no Scout or Sandwich operation is applicable and G_C is a covering graph of G containing a vertex $b \in \textsc{LiveLeaves}_G(T)$ with at most $H(\textsc{Genus}(G)) - 2$ neighbors in $G_C - \{r\}$, then there are sets S and S' such that:*

- $S \subseteq \textsc{LiveLeaves}_G(T) \setminus \{b\}$,
- $|S| \leq H(\textsc{Genus}(G)) - 2$,
- $S' \subseteq N_G(N_G(S)) \setminus V(T)$,
- $|S'| \leq 1$,
- $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$.

Proof. Assume we have a vertex b as stated in the theorem. We show how to construct the sets S and S'. We choose $S := N_{G_C}(b) \setminus \{r\}$. Since b has at most $H(\textsc{Genus}(G)) - 2$ neighbors in $G_C - \{r\}$ we have $|S| \leq H(\textsc{Genus}(G)) - 2$ as required. Since all vertices in the covering graph except r are members of $\textsc{LiveLeaves}_G(T)$, by this definition we also satisfy $S \subseteq \textsc{LiveLeaves}_G(T) \setminus \{b\}$ since the covering graph is simple and hence b cannot have an edge to itself in G_C.

We define the set S' based on the situation of the neighbors of b.

- If there is a vertex $w \in N_G(b) \setminus V(T)$ such that $|N_G(w) \cap V(T)| = 1$, then by Lemma 9 the vertex w is adjacent to a connector vertex $b^* \notin V(T)$ which implies that b is in the tree-neighborhood of this connector vertex. By Lemma 7 there is at most one such connector vertex; we choose $S' := \{b^*\}$ which satisfies the size bound of S'. Let us verify that $S' \subseteq N_G(N_G(S)) \setminus V(T)$, which we will prove by showing that there is some $b' \in S$ such that $b^* \in N_G(N_G(b'))$. By the definition of S this is equivalent to showing that b has an edge in G_C to a vertex $b' \in \textsc{LiveLeaves}_G(T)$ with $b^* \in N_G(N_G(b'))$. By the definition of the tree-neighborhood, this reduces to showing that b has an edge in G_C to a vertex $b' \in \textsc{LiveLeaves}_G(T)$ such that b and b' are both in the tree-neighborhood of the connector vertex b^*. But since such an edge is required to exist in a covering graph by Property (3b) of Definition 9, this must be the case.
- If all $w \in N_G(b) \setminus V(T)$ satisfy $|N_G(w) \cap V(T)| \geq 2$ then we set $S' := \emptyset$. By this definition S' satisfies the stated size bound, and S' is trivially a subset of $N_G(N_G(S)) \setminus V(T)$.

It only remains to verify that $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$. Consider some $w \in N_G(b) \setminus V(T)$. We will argue that there must be some vertex $s \in S \cup S'$ that is adjacent to w.

– If $|N_G(w) \cap V(T)| = 1$ then w is adjacent to a connector vertex b^* by Lemma 9. By definition of S' we now have $S' = \{b^*\}$ and so the vertex $b^* \in S'$ is adjacent to w.
– If $|N_G(w) \cap V(T)| \geq 2$ then by Property (3a) of Definition 9 the vertex b has an edge in G_C to a vertex $b' \in \text{LIVELEAVES}_G(T)$ such that $b' \in N_G(w) \cap V(T)$. This vertex b' must be in S by definition, which means that the vertex $b' \in S$ is adjacent to w.

Since we have shown that for each $w \in N_G(b) \setminus V(T)$ there is a $s \in S \cup S'$ with $w \in N_G(s)$ we may conclude that $N_G(b) \setminus V(T) \subseteq N_G(S \cup S')$. Since we have shown that S and S' satisfy all requirements, the proof is complete. □

Corollary 3. *If no Scout or Sandwich operation is applicable and T does not yet span G, then a Surround operation is applicable.*

Proof. By definition $\beta = \min(H(\text{GENUS}(G)) - 2, \text{BLACKDEG}_c(G) - 1)$. If the term with $\text{BLACKDEG}_c(G)$ yields the minimum value for the determination of β, then a Surround operation is applicable by Lemma 10.

Otherwise $\beta = H(\text{GENUS}(G)) - 2$. By Lemma 12 there is a covering graph G_C for G with $\text{GENUS}(G_C) \leq \text{GENUS}(G)$. By Lemma 13 such a covering graph must contain a vertex $b \in \text{LIVELEAVES}_G(T)$ with at most $H(\text{GENUS}(G)) - 2$ neighbors in $G_C - \{r\}$. By Lemma 6 this implies that there are sets S and S' that satisfy the requirements of a Surround operation for vertex b, which implies that a Surround operation is applicable. □

Let us summarize our line of argument. We have seen that we can always initialize a tree appropriately. Corollary 3 shows that if the tree is not yet spanning then a Surround operation is always applicable if the Scout and Sandwich operations are not. This shows that we can always grow the tree into a spanning tree using these operations. Since every individual operation satisfies the Incremental Inequality (3), the proof of Theorem 5 is now complete. □

5.5 The Balance Between Black and White

Theorem 5 lower bounds the number of black leaves in the total number of black vertices. In order to prove a kernelization lemma we need to lower bound the number of black leaves in a graph in class $\mathcal{C}^b$ in the *total size* of the graph, also counting the white vertices. We will show how such an extension of Theorem 5 can be established by proving that the number of black vertices is at least a constant fraction of the total graph size.

Theorem 7. *Let $\langle G = (V, E), c\rangle \in \mathcal{C}^b$. It must hold that:*

1. $|\mathcal{B}(G)_c| \geq \frac{|G| - 10\,\text{GENUS}(G)}{6}$,
2. $|\mathcal{B}(G)_c| \geq \frac{|G|}{1 + \frac{1}{2}\,\text{BLACKDEG}_c(G)}$.

Proof. (Case 1) First we will bound the number of white vertices $|\mathcal{W}_c(G)|$ in the graph, by giving separate bounds for the set W_2 of white degree-2 vertices, and the set $W_{\geq 3}$ of white vertices with degree ≥ 3.

To bound the size of W_2 we consider the graph $G' = (V', E')$ that is obtained from G by deleting all white vertices of degree $\neq 2$, and by then replacing every white vertex by a direct edge between its two neighbors. Since G is bipartite every white vertex of degree 2 in G is represented by an edge in G'. By Property (v) of Definition 6 we know that graph G' is simple since no two degree-2 white vertices have the same neighborhood. Every vertex in the resulting graph G' is a c-black vertex in G. Since G' is a minor of G we know that $\text{GENUS}(G') \leq \text{GENUS}(G)$, which means we can apply Lemma 1 to derive that the number of edges $|E'|$ is at most $3|V'| + 6\,\text{GENUS}(G)$. Using the correspondences derived earlier we find that:

$$|W_2| = |E'| \leq 3|V'| + 6\,\text{GENUS}(G) = 3|\mathcal{B}(G)| + 6\,\text{GENUS}(G). \tag{5}$$

To bound the size of $W_{\geq 3}$ we consider the black/white colored graph $\langle G'', c''\rangle$ obtained from G by deleting all white vertices of degree 2, and letting c'' be the function c restricted to the vertices of G''. Since $G'' \subseteq G$ and the coloring c'' is a restriction of c we know that G'' is simple, and bipartite according to the coloring function c''. By the structure of the transformation we know that the set $\mathcal{W}_{c''}(G'')$ equals the set $W_{\geq 3}$. Since every c''-white vertex in G'' has degree at least 3, we find that:

$$|E''| \geq 3|\mathcal{W}_{c''}(G'')| = 3|W_{\geq 3}|. \tag{6}$$

Since G'' is bipartite with vertex sets $\mathcal{B}_{c''}(G'') = \mathcal{B}_c(G)$ and $\mathcal{W}_{c''}(G'') = W_{\geq 3}$ we find using Lemma 1 that:

$$|E''| \leq 2\left(|\mathcal{B}_{c''}(G'')| + |\mathcal{W}_{c''}(G'')|\right) + 4\,\textsc{Genus}(G) = 2\left(|\mathcal{B}(G)| + |W_{\geq 3}|\right) + 4\,\textsc{Genus}(G). \tag{7}$$

Combining Equation (6) with Equation (7) we now find that $3|W_{\geq 3}| \leq 2(|\mathcal{B}_c(G)| + |W_{\geq 3}|) + 4\,\textsc{Genus}(G)$, from which our final bound on $|W_{\geq 3}|$ follows as

$$|W_{\geq 3}| \leq 2|\mathcal{B}_c(G)| + 4\,\textsc{Genus}(G). \tag{8}$$

By adding Equation (5) to Equation (8) we find a general bound on $|\mathcal{W}_c(G)|$:

$$|\mathcal{W}_c(G)| = |W_2| + |W_{\geq 3}| \leq 3|\mathcal{B}_c(G)| + 6\,\textsc{Genus}(G) + 2|\mathcal{B}_c(G)| + 4\,\textsc{Genus}(G) = 5|\mathcal{B}_c(G)| + 10\,\textsc{Genus}(G). \tag{9}$$

We use this bound to obtain the final result.

$$\begin{aligned}
|G| &= |\mathcal{B}_c(G)| + |\mathcal{W}_c(G)| && G \text{ is bipartite.}\\
&\leq |\mathcal{B}_c(G)| + 5|\mathcal{B}_c(G)| + 10\,\textsc{Genus}(G) && \text{By Equation (9).}\\
&= 6|\mathcal{B}_c(G)| + 10\,\textsc{Genus}(G) && \text{Simplifying.}\\
6|\mathcal{B}_c(G)| &\geq |G| - 10\,\textsc{Genus}(G) && \text{Moving terms.}\\
|\mathcal{B}_c(G)| &\geq \frac{|G| - 10\,\textsc{Genus}(G)}{6} && \text{Dividing.}
\end{aligned} \tag{10}$$

(Case 2) The case utilizing the bound on the degree of positive-weight vertices is considerably simpler. Since G is bipartite according to coloring function c and every c-white vertex has degree at least 2, we find that $|E| \geq 2|\mathcal{W}_c(G)|$. Since every c-black vertex has degree at most $\textsc{blackdeg}_c(G)$ we find that $|E| \leq \textsc{blackdeg}_c(G)|\mathcal{B}_c(G)|$. By transitivity we find that $2|\mathcal{W}_c(G)| \leq \textsc{blackdeg}_c(G)|\mathcal{B}_c(G)|$. Noting that $|G| = |\mathcal{B}_c(G)| + |\mathcal{W}_c(G)|$ as for the previous case, the result follows easily since $|G| = |\mathcal{B}_c(G)| + |\mathcal{W}_c(G)| \leq |\mathcal{B}_c(G)| + \frac{1}{2}\,\textsc{blackdeg}_c(G)|\mathcal{B}_c(G)|$. □

5.6 Constructing a Spanning Tree

Theorem 8. *Let $\langle G = (V, E), c\rangle \in \mathcal{C}^b$ such that $\textsc{blackdeg}_c(G) \geq 3$. Assuming that a minimum-genus embedding of G is supplied to the algorithm and that a collapse operation on a vertex x in an embedded graph can be executed in $O(\deg(x))$ time, we can find a spanning tree for G with at least $|\mathcal{B}(G)|/(5 + 4\beta)$ black leaves in $O(|V| + |E|)$ time with $\beta := \min(H(\textsc{Genus}(G)) - 2, \textsc{blackdeg}_c(G) - 1)$.*

Proof. We give a high-level description of an algorithm that constructs a spanning tree that satisfies the stated requirements.

The algorithm starts by selecting a white vertex w and initializing T as the star rooted at w. During its operation, the algorithm maintains the following sets as doubly-linked lists:

- the set L consisting of live leaves of the tree - by the invariant of the construction process the vertices in L are all black,
- the set W consisting of white degree-2 vertices contained in $N_G(V(T))$.

For each vertex $v \in V$ we store a boolean $\textsc{inTree}$ whether it is in T, and a boolean $\textsc{neighborOfTree}$ whether it is in $N_G(V(T))$. If a vertex is a member of the list L or W, we store a pointer to its entry in the list with the vertex so it may be removed in constant time. In vertex v we store its regular adjacency list containing the vertices $N_G(v)$, but we also store a separate tree-adjacency list with pointers to the vertices in $N_G(v) \cap (V(G) \setminus V(T))$. If the neighbor u is stored in both lists, then we keep cross-pointers between the nodes representing u in the lists so that one entry may be removed in constant time when we are accessing the other entry.

Without going into too much detail, we claim that using this representation a series of vertex expansions take $O(\deg_G(v))$ per vertex v that is expanded or added to T. Since every vertex is added to T only once and a vertex is expanded at most once, this will then yield the stated running time bound if we can show that the remaining computations (i.e. determining which vertices to expand) can be done in $O(\deg_G(v))$ time per vertex.

For notational convenience we define the potential $p(v)$ of a vertex $v \in \textsc{LiveLeaves}_G(T)$ as $p(v) := N_G(N_G(v)) \setminus V(T)$. The status information described above allows us to determine in $O(\deg_G(v))$ time whether $p(v) \geq 2$, by taking the following steps: for each $u \in N_G(v)$ we test if $u \notin V(T)$, and if this is the case we determine if the size of the tree-adjacency list (containing the pointers to $N_G(u) \cap (V(G) \setminus V(T))$) is at least 2. If this is the case then the potential of v is at least 2. If the size of the tree-adjacency list is 0 then u does not contribute to the potential of v. If the size is 1, then we remember the single entry in the tree-adjacency list of u if we had not remembered such an entry before. If we have remembered an entry we test whether it is different from the current entry; if the new entry differs from the remembered entry then we know the potential of v is at least 2. If we have tested all $u \in N_G(v)$ without concluding that the potential of v is at least 2, then the potential must be less than 2.

The importance of the potential test is that if $v \in \textsc{LiveLeaves}_G(T)$ and $p(v) \geq 2$, then it is a valid Scout operation to expand v (which must be black since it is a live leaf) and all the vertices in $N_G(v) \setminus V(T)$, since that corresponds to the expansion of the grow-path $P = \langle v \rangle$ with a yield of at least 2.

We only have to test the potential of a vertex v once, since the potential of v can only decrease when other vertices are expanded. If $p(v) \geq 2$ then we expand v (and hence never have to reconsider it since it is already in the tree), and if $p(v) < 2$ then that will always remain the case so we do not have to test again.

As long as there are leaf vertices with a potential of at least 2, we expand these vertices and their white neighbors in Scout operations. When this is no longer possible, we inspect the list W. If there is a white vertex w of degree 2 in the list W, such that w is adjacent to a black vertex $b \notin V(T)$ that is not a connector vertex, then by Lemma 9 we know that we can do a valid Scout or Sandwich operation that includes w and b. So we need to be able to test if a $w \in W$ is adjacent to a black non-connector vertex outside T. Since w has degree 2 we can determine in constant time if it has a black neighbor b outside T, by using the described representation. We can then test in $O(\deg_G(b))$ time whether b is a connector vertex, by determining how many of its neighbors are adjacent to T. If we find that b is not a connector vertex then we expand w and b through a suitable operation, as described in the case analysis of Lemma 9. If b is a connector vertex, then we know that all its neighbors in T must remain in T after future vertex expansions, so b will remain a connector vertex for as long as it is outside T. So for each vertex we only have to test once whether it is a non-connector vertex; we can store the outcome of this test to ensure we only do this once.

When there are no vertices in W that are adjacent to a black non-connector vertex outside T, then we know from Corollary 3 that we can do a Surround operation. To execute the Surround operation we make a distinction based on whether the genus-bound or the degree-bound yields the lowest term in the definition of β.

- If β was determined by the genus-bound, then we use the minimum-genus embedding of G to construct a covering graph as described in Lemma 12. We assume that we can extract the embedding of a subgraph $G' \subseteq G$ in $O(\sum_{v \in V(G')} \deg_G(v))$ time from the embedding of G. We only extract the embedding of the live-leaf vertices and the connector vertices of G, and do no explicitly copy and contract all internal vertices of the tree as described in Lemma 12. We collapse the connector vertices in the embedding, which takes $O(\deg_G(b))$ time per connector vertex b by assumption. In this way we obtain a valid covering graph G_C for G. Rather than using this covering graph for a single Surround operation, we utilize the information from the covering graph more efficiently by extracting the information for a series of Surround operations from it. We scan through the occurrences of vertices in L in the covering graph, and for each vertex v check if its degree in the covering graph is sufficiently low (as described in Lemma 13). If this is the case, then we mark v to be killed, and all its neighbors in the covering graph to be expanded. We then continue until all vertices in L are either marked to be killed, or marked to be expanded. When we have marked a vertex to be expanded in an earlier decision, then we no longer count it when finding low-degree vertices to kill. It can be derived from the theory about the applicability of Surround operations that all the Surround operations resulting from this scan through the covering graph are valid Surround operations. Since we expand or kill all live-leaf vertices occurring in a covering graph, every vertex of G occurs in at most one covering graph. This means that we can distribute the cost of the covering graph construction over the vertices that are involved, and find a cost of $O(\deg_G(v))$ per vertex v of graph G.

– If β was determined by the degree-bound then we can simply pick a vertex in L and kill it by expanding other live leaves, as described by Lemma 10.

We argued earlier that the status information of each vertex can be updated efficiently. Since we have now also shown that we can determine which vertices to expand by spending $O(\deg_G(v))$ per vertex $v \in V$, this concludes the proof. □

6 Kernelization for Weighted Max Leaf

In this section we introduce some concepts regarding kernelization that are common to the kernel for Weighted Max Leaf$_0$ and the kernel for Weighted Max Leaf$_+$.

6.1 Kernelization Strategy

Although the details of the kernelization for Weighted Max Leaf$_0$ and Weighted Max Leaf$_+$ differ considerably, the overall strategy is similar. Each kernel utilizes a set of reduction rules that reduce the instance to a smaller, but equivalent instance. Some instances are so easy that the reduction rules will reduce them to graphs consisting of no more than 2 vertices. An instance of Weighted Max Leaf is called *trivial* if it has at most 2 vertices, or if the reduction rules reduce it to an instance with at most 2 vertices. The kernelization algorithms can deal with trivial instances by determining the answer to the problem in constant time, and outputting a 1-vertex graph with the same answer: if the answer is YES then the algorithm outputs an instance consisting of a graph with a single vertex of weight 1 and $k := 1$; if the answer is NO then we set the target weight to $k := 2$. Such a 1-vertex instance trivially satisfies any reasonable upper bounds on the number of vertices in the kernel.

A kernelization algorithm proceeds as follows. Given an instance $\langle G, w, k\rangle$ it repeatedly applies reduction rules to the instance. When no reduction rules can be applied anymore, the resulting instance $\langle G', w', k'\rangle$ is called a *reduced instance*. If the reduced instance is not trivial, we will use the structure theory that was developed in Sections 4 and 5 to show the answer to the decision problem must be YES for large reduced instances. Therefore we derive a suitable linear bound on the number of vertices in a graph, such that any reduced instance whose size exceeds the bounds must be a YES instance. We reduce to a a trivial YES-instance for such large reduced instances. If the graph size does not exceed the given bound, then we know that the instance is small and the algorithm outputs the reduced instance $\langle G', w', k'\rangle$.

6.2 Reduction Rules

Throughout this work we present reduction rules in a structured, 3-stage format. We first describe the structure of the graph that can be reduced. This is followed by an explanation of the operations that have to be performed to achieve the reduction. The last stage of the description of a reduction rule consists of a justification for its correctness. 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*. We will use two methods to prove that the reduction rules for this kernel are safe. For most rules we will directly prove the equivalence between the two instances. In the remaining cases the reduction consists of the removal of an edge e, and we will then prove that any spanning tree using that edge can be transformed into a spanning tree of at least as much weight that avoids edge e. The equivalence between the original and the reduced instance then follows from this proof.

The following two reduction rules will be used by both kernels. They are illustrated by examples in Figure 4.

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.
Operation: Define $m := \max(w(s_1), w(s_q))$. 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 $w'(s') := \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}) - m)$.
Justification: If a spanning tree avoids an edge with both endpoints inside the p-component, it is always optimal to avoid an edge of maximum weight. The reduced graph offers the same connection and weighting possibilities as the original.

Lemma 14. *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 $\text{LW}(T) \geq k$. 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.

If T uses all edges on P, then add all edges on P' to T'. In this case $\text{LEAVES}(T) = \text{LEAVES}(T')$ so $\text{LW}(T) = \text{LW}(T')$.

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 p-component, then add all edges on P' to T' except the edge that is equivalent to e in G'. In this case the leaf weight achieved by T in the component P is equal to the leaf weight achieved by T' in component P', which shows that $\text{LW}(T') \geq \text{LW}(T) = k = k'$.

If the edge that T avoids on P is not incident to an endpoint of the p-component, then add all edges on P' to T' except for the edge between s' and the vertex of s_1, s_q that has maximum weight. 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 $w'(s') + m = \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}))$, from which we can conclude $\text{LW}(T') \geq \text{LW}(T)$.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\text{LW}(T') \geq k'$. 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 p-component, then add all edges on P to T. The tree T then spans G and $\text{LEAVES}(T) = \text{LEAVES}(T')$.

If T' avoids edge us_1 or $s_k v$, then add to T all edges on P except the one that is avoided by T'. After this operation it once again holds that T is a spanning tree with $\text{LEAVES}(T) = \text{LEAVES}(T')$.

So in the only remaining case tree T' avoids exactly one edge in the reduced p-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 $w'(s') + \max(w(s_1), w(s_q)) = w'(s') + m$ to $\text{LW}(T')$. To augment T to a spanning tree we need an edge e^* on P of maximal weight, which must be $w'(s') + m$ by definition. 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 that 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 p-component under consideration, so $\text{LW}(T) \geq \text{LW}(T')$ which proves that G has a spanning tree with leaf weight at least $k' = k$. □

Reduction Rule 2 Cut a simple cycle

Structure: The graph G is a simple cycle.
Operation: Let uv be an edge that maximizes the weight $w(u) + w(v)$. Remove edge uv from G.
Justification: A spanning tree for a cycle avoids exactly one edge, and it is optimal to avoid an edge of maximum weight.

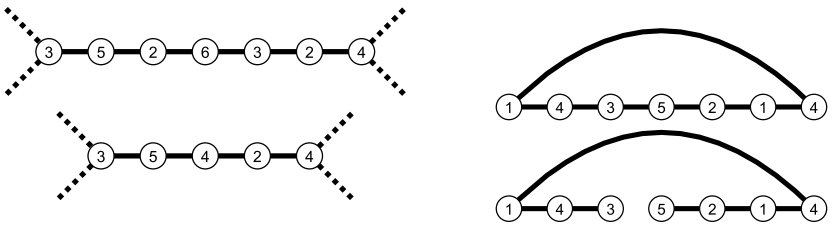
Lemma 15. *Rule [2] is safe: there is an optimal spanning tree that avoids* uv*.*

Proof. Suppose T is a spanning tree for G that uses the edge uv. We will prove that there is a spanning tree T' for G that avoids edge uv and that has at least as much leaf weight as T. Since G is a cycle, any spanning tree for G avoids exactly one edge. Let edge xy be the edge avoided by T. The weight of this spanning tree is $w(x) + w(y)$. Now consider the spanning tree T' that is obtained by taking G and removing edge uv. The leaf weight of T' is $w(u) + w(v)$. Since edge uv is an edge of maximal weight, the leaf weight of T' must be at least as large as the leaf weight of T which proves the claim. □

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.

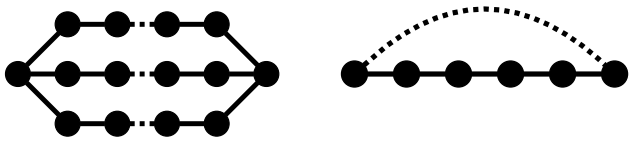
Lemma 16. *Let* $\langle G, w, k\rangle$ *be an instance of* WEIGHTED MAX LEAF *with* $G = (V, E)$ *such that Rule [1] can be applied, and let* $\langle G', w', k'\rangle$ *with* $G' = (V', E')$ *be an instance after application of Rule [1]. Then it holds that* $\min\{w'(v) \mid v \in V'\} \geq \min\{w(v) \mid v \in V\}$*.*

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 V under w. The weight of s' is defined as $w'(s') := \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1}) - \max(w(s_1), w(s_q)))$. Since the edges $s_1 s_2$ and $s_{q-1} s_q$ are also considered in the maximum, it follows that $w'(s') \geq \max\{w(s_1) + w(s_2), w(s_{q-1}) + w(s_q)\} - \max\{w(s_1), w(s_q)\} \geq \min\{w(s_2), w(s_{q-1})\} \geq \min\{w(v) \mid v \in V\}$. □



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

Fig. 4. Examples of the reduction rules common to both kernels. The image at the top shows the original structure, whereas the image at the bottom shows how it is reduced.



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

Fig. 5. Constructions to show the necessity of some reduction rules. To show necessity of Rule [R] we give a graph G that is not reducible without using [R], such that G can be made arbitrarily large without increasing the leaf weight that can be gained in a spanning tree $T \subseteq G$. The weight of each drawn vertex is 1.

6.3 Necessity of the Reduction Rules

When presenting reduction rules for a kernelization algorithm it is always interesting to see whether all reduction rules are indeed needed to be able to obtain a kernelization result, or if there are some redundant reduction rules. For both kernels we shall show that all reduction rules are indeed necessary.

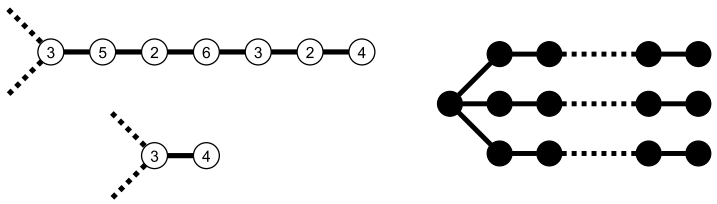
Our kernelization algorithms rely on a statement of the form: if the reduced graph is large with respect to k, then the graph must contain a spanning tree with leaf weight at least k. To be able to prove such a statement we require that larger a reduced instances contain spanning trees with higher leaf weight. If there is a graph that can grow arbitrarily large without increasing the leaf weight that can be obtained in a spanning tree, then we cannot obtain the desired kernelization lemma and hence we cannot derive a kernel using our strategy. So to show that a reduction rule [R] is necessary we will present a graph G that is not reducible without using rule [R], such that G can be made arbitrarily large without increasing the leaf weight that can be gained in a spanning tree $T \subseteq G$. Figure 5 shows such constructions for Rule [1] and Rule [2], which implies that these rules are necessary for the Weighted Max Leaf$_+$ kernel and the Weighted Max Leaf$_0$ kernel. Of course the problem-specific reduction rules given for these kernels in Sections 7.1 and 8.1 should be taken in mind when verifying that the given graphs are not reducible without the use of the rule [R] that is shown to be necessary.

7 Linear Kernel for Weighted Max Leaf$_+$ on General Graphs

In this section we prove the existence of a problem kernel with $7.5k$ vertices for the Weighted Max Leaf$_+$ problem on general graphs. The correctness proof for the kernelization algorithm is based on Theorem 3.

7.1 Reduction Rules

This kernel uses Rule [1] and Rule [2] as described in Section 6.2. Additionally, we use the following rule that is specific to the case of strictly positive weight values. Figure 6(a) gives an example of the execution of this rule.



(a) (b)

Fig. 6. 6(a): Example of the application of Rule [3]. The original structure is shown on the top, whereas the figure at the bottom shows how it is reduced. 6(b): Construction to show the necessity of Rule [3]. To show necessity of the rule we give a graph G that is not reducible without using it, such that G can be made arbitrarily large without increasing the leaf weight that can be gained in a spanning tree $T \subseteq G$, as described in Section 6.3. The weight of each drawn vertex is 1.

Reduction Rule 3 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 17. *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 $\text{LW}(T) \geq k$. 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 $k = k'$. □

Figure 6(b) shows that this reduction rule is necessary.

7.2 Structure of a Reduced Instance

The structure of reduced instances is captured by the class $\mathcal{C}^3$ (see Definition 4), which is proven in the following theorem.

Theorem 9. *If* $\langle G', w', k'\rangle$ *is a non-trivial reduced instance of* Weighted Max Leaf$_+$*, then* $G' \in \mathcal{C}^3$.

Proof. Suppose $\langle G', w', k'\rangle$ is a non-trivial reduced instance. We will prove that G' satisfies all properties of Definition 4 with $\mathcal{P} = 3$.

(i) The input graph is simple and connected by definition. It is easy to verify that the reduction rules preserve these properties.
(ii) By Rule [2] 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 [3] the reduced graph G' does not have degree-1 vertices that are adjacent to degree-2 vertices.

Since G' satisfies all the required properties we may conclude that $G' \in \mathcal{C}^3$. □

7.3 Kernelization Algorithm

Theorem 10. *The* Weighted Max Leaf$_+$ *problem has a kernel with at most* $7.5k$ *vertices.*

Proof. When supplied with an input $\langle G, w, k\rangle$ the kernelization algorithm first exhaustively applies the reduction rules to obtain a reduced instance $\langle G', w', k'\rangle$. 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 $|G'| \geq 7.5k'$ or $k' \leq 0$ then the algorithm builds a trivial yes-instance consisting of a single vertex and outputs it. Otherwise, the algorithm outputs $\langle G', w', k'\rangle$.

Since each reduction rule can be executed in polynomial time (see Section 7.4 for more details) and each reduction rule decreases the number of vertices or edges (or both) in the graph, it follows that we can obtain the reduced instance in polynomial time.

Let us verify that the algorithm is correct when it outputs a trivial yes-instance. Suppose $|G'| \geq 7.5k'$. Since $G' \in \mathcal{C}^3$, we know by Theorem 3 that G' has a spanning tree with at least $|G'|/(1.5{\cdot}3{+}3) = |G'|/7.5 \geq k'$ leaves. Since every vertex in G' has a weight of at least 1 by definition, we find that G' has a spanning tree with leaf weight at least k'. By the correspondence between the instance $\langle G, w, k\rangle$ and $\langle G', w', k'\rangle$ this implies that $\langle G, w, k\rangle$ is indeed a yes-instance, which proves the correctness of the algorithm in this case. The safety of the reduction rules shows the correctness of the algorithm when it outputs the reduced instance.

As the last part of the proof we argue that the output of the algorithm is a valid instance of Weighted Max Leaf$_+$, which is needed to show that the kernelization is indeed a self-reduction. Since the reduction rules preserve the fact that the graph is simple and connected, 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 16 proves that the minimum vertex weight is not decreased by application of Rule [1], which proves that the kernelization is indeed a self-reduction. This concludes the proof of the correctness of the kernel. □

7.4 Implementation of the Kernelization Algorithm

Theorem 11. *Let* $\langle G, w, k\rangle$ *be an instance of* Weighted Max Leaf$_+$ *with* $G = (V, E)$*. Assuming that basic arithmetic on the vertex weights can be done in constant time, we can obtain the reduced instance* $\langle G', w', k'\rangle$ *in* $O(|V| + |E|)$ *time.*

Proof. Without going into too much detail, we give a high-level description of an implementation that achieves this running time.

If G does not have vertices of degree 3 or more then we can easily find the maximum leaf weight of a spanning tree in linear time: either G is a path P_n and the maximum leaf weight is the sum of the weights of the path endpoints, or G is a cycle C_n and the maximum leaf weight is the maximum combined weight of the endpoints of an edge.

So it remains to show how to reduce an instance in linear time when G has a vertex of degree at least 3. If there is a vertex $v \in V$ with $\deg_G(v) \geq 3$, then no application of a reduction rule will decrease the degree of v. Therefore Rule [2] will never be applicable in this case. We can do a single Depth-First Search scan through the graph that starts at v and immediately applies Rule [1] and Rule [3] where they are relevant. By keeping appropriate cross-references we can ensure that whenever we do a DFS-visit on a vertex v, we have constant-time access to the first parent of v in the DFS-tree that has a degree of at least 3. Using such references and by keeping track of the largest weight of edge endpoints that are encountered when traversing a path component, we can execute Rule [1] and Rule [3] in constant time once the DFS has visited both p-component endpoints. This yields a total running time of $O(|V| + |E|)$ for computing the reduced instance. □

7.5 Extension to Rational Vertex Weights

This kernel can also be used for the Weighted Max Leaf problem with a weight function that gives each vertex a rational-valued weight of at least 1; no changes are required. The kernelization argument only relies on the fact that each weight is at least 1, but not on the fact that weights are integers. Since a kernel should perform a self-reduction, we need to verify that the reduction rules will not assign weights below 1: since Lemma 16 does not use the fact that the weights are integers, the lemma still holds for the rational-weight case and the kernelization is indeed a self-reduction.

7.6 Discussion

We have presented a simple problem kernel for Weighted Max Leaf$_+$ with $7.5k$ vertices, and argued that the given kernelization also works when the vertex weights are elements of $\mathbb{Q}_{\geq 1}$. The kernel uses three reduction rules that can be implemented efficiently.

When the algorithm outputs a trivial yes instance - because it knows a spanning tree with sufficient leaf weight must exist - then such a spanning tree can also be found in $O(|V|+|E|)$ time by applying Theorem 4. The safety proofs of the reduction rules show that the rules are reversible, in the sense that a spanning tree for a reduced instance can be lifted back to a spanning tree for the original instance. This implies that the given kernel can also solve the construction variant of Weighted Max Leaf$_+$, where we are not only looking for a yes or no answer but also want to build a spanning tree with the specified leaf weight.

The linear time algorithm that constructs a leafy spanning tree can in practice also help to obtain smaller reduced instances. In the given form, the kernelization algorithm will not change the size of an instance with $7.4k$ vertices - this is not needed, because the size of the instance is small enough. But when we run the algorithm of Theorem 4 on the reduced instance, it might find a spanning tree with leaf weight at least k so that the kernelization can directly solve the question. This suggests that we implement the kernel by searching for a spanning tree in the reduced instance using the mentioned algorithm, and comparing its leaf weight to k. If the reduced instance has at least $7.5k$ vertices then we are guaranteed to find a spanning tree with leaf weight at least k, but we might also find such high-weight spanning trees on smaller instances which means that such instances do not have to be solved by an expensive exponential-time algorithm.

Finally let us discuss the size of the kernel. The size of the kernel is directly related to the bound of Theorem 3. The factor 7.5 in this theorem is best-possible as shown in Appendix B, which implies that the analysis of the kernel size is tight; using the current set of reduction rules it is not possible to claim for any $c < 7.5$ that a graph with ck vertices always has a spanning tree with leaf weight at least k. The use of new reduction rules (such as contracting bridges whose endpoints have degree at least 2, adapting the KW-dissolver rule from [15] to a weighted setting, or reducing triangles in the graph if each vertex in the triangle is adjacent to a different degree-2 vertex) might allow the bound of Theorem 3 to be improved, thus resulting in a smaller kernel.

8 Linear Kernel for Weighted Max Leaf$_0$ on Graphs of Bounded Genus or Degree

In this section we present a linear kernel Weighted Max Leaf$_0$ restricted to graphs of bounded genus and graphs in which the degree of positive-weight vertices is bounded. We start the presentation of the kernelization algorithm in Section 8.1 by giving 4 additional problem-specific reduction rules. We analyze the reduced instances in Section 8.2. Using these ingredients we present the kernelization algorithm in Section 8.3. In Section 8.4 we briefly consider the running time of an implementation of the reduction rules.

8.1 Reduction Rules

The kernel uses Rule [1] and Rule [2] as described in Section 6.2. Additionally, we use the following 4 rules that are specific to the case of non-negative weight values. Figure 7 shows examples of the application of these rules.

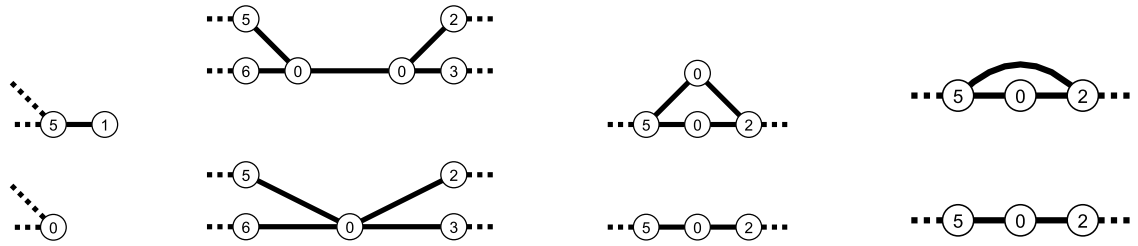
Reduction Rule 4 Remove vertices of degree 1

Structure: Vertex v of degree 1, adjacent to u with $\deg(u) > 1$.
Operation: Remove v, set $k' := k - w(v)$ and $w'(u) := 0$.
Justification: In G vertex v will be leaf in any spanning tree since it has degree 1, whereas vertex u will never be a leaf of positive weight in any spanning tree since it is a cut vertex.

Lemma 18. *Rule [4] 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 $\text{lw}(T) \geq k$. Then $T' := T - \{v\}$ is a spanning tree for G' with $\text{lw}(T') = \text{lw}(T) - w(v)$. So G' has a spanning tree of weight at least $k - w(v) = k'$.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\text{lw}(T') \geq k'$. We obtain spanning tree T for G by adding vertex v and the edge uv to T'. This addition can cause u to become internal in tree T, but since $w(u) = 0$ in G' this does not decrease the leaf weight. So every positive-weight leaf in T' must also be a leaf in T. In addition, tree T also has v as a leaf. So $\text{lw}(T) \geq k' + w(v) = k$. □



(a) Rule [4]. (b) Rule [5]. (c) Rule [6]. (d) Rule [7].

Fig. 7. Examples of the reduction rules that are only for Weighted Max Leaf$_0$. The image at the top shows the original structure, whereas the image at the bottom shows how it is reduced.

Reduction Rule 5 Contract edges between weight-0 vertices

Structure: Two distinct vertices u, v with $w(u) = w(v) = 0$ and $uv \in E$.
Operation: Contract the edge uv into a new vertex x with $w'(x) := 0$.
Justification: There is always an optimal spanning tree that uses edge uv, and the endpoints will never be weight-contributing leaves of any spanning tree.

Lemma 19. *Rule [5] 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 will first show that there is a spanning tree T^* for G that uses edge uv with $\text{LW}(T^*) \geq \text{LW}(T)$.

If T uses edge uv we are done, so assume T avoids the edge. By adding uv to T and removing a different edge from the resulting cycle, we obtain a spanning tree T^*. Vertices $\{u, v\}$ are the only vertices whose degree in T^* is higher than their degree in T. Since those vertices have weight 0 we can conclude that $\text{LW}(T^*) \geq \text{LW}(T)$.

So we may assume we have a spanning tree T^* with $\text{LW}(T^*) \geq k$ that uses edge uv. If we contract edge uv in T^* we obtain a spanning tree T' for G'. Since the endpoints of the contracted edge have weight 0 this operation does not decrease the leaf weight, and we find that $\text{LW}(T') \geq k = k'$.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\text{LW}(T') \geq k'$. We initialize T as $T - \{x\}$. Augment T to a spanning tree for G by the following steps. First add isolated vertices $\{u, v\}$ and the edge uv to T. For every vertex $y \in N_{T'}(x)$ add an edge uy to T if $y \in N_G(u)$ and vy otherwise. By the structure of the reduction T must be a spanning tree for G. For every vertex except x the degree in T is the same as the degree in T'. Since x has weight 0 we must have $\text{LW}(T) = \text{LW}(T')$, which proves that G has a spanning tree with leaf weight at least $k' = k$. □

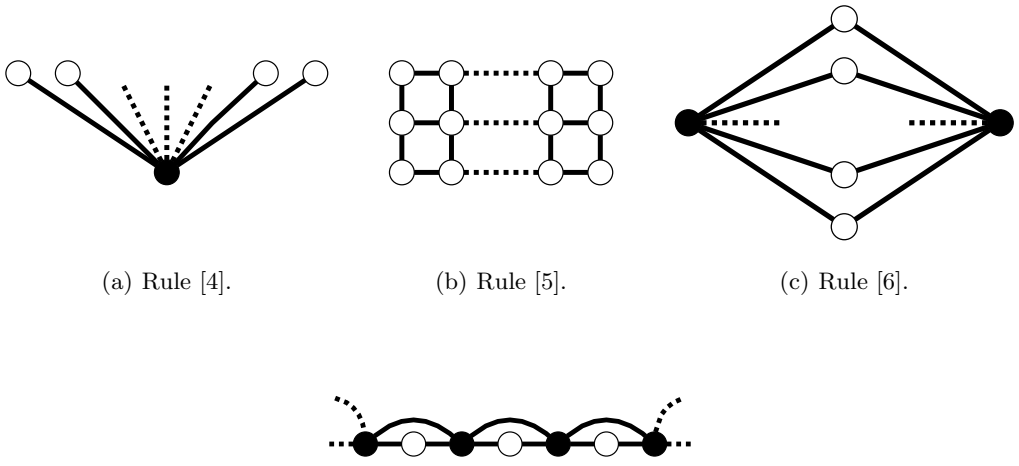
Reduction Rule 6 Remove duplicate degree-2 weight-0 vertices

Structure: Two distinct vertices u, v of degree 2 with $w(u) = w(v) = 0$ such that $N(u) = N(v)$ and $(V \setminus \{u, v\}) \setminus N(u) \neq \emptyset$.
Operation: Remove vertex v and its incident edges.
Justification: The neighborhood $N_{G'}(u)$ forms a cutset in G', so we can turn a spanning tree for G' into a spanning tree for G by attaching the deleted vertex v to the internal neighbor of u in the spanning tree.

Lemma 20. *Rule [6] 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 $\text{LW}(T) \geq k$. At least one of the vertices $\{u, v\}$ must have degree 1 in T, otherwise the tree contains a cycle. Let x be a vertex in $\{u, v\}$ of degree 1 in T. Remove x and its incident edge from T and re-label the remaining vertex in $\{u, v\} \setminus \{x\}$ to u to obtain a spanning tree T' for G'. Since we deleted a leaf of weight 0 and did not increase the degree of any vertex in the spanning tree we have $\text{LW}(T') \geq \text{LW}(T)$.

($\Leftarrow$) Suppose G' has a spanning tree T' with $\text{LW}(T') \geq k'$. Since $(V \setminus \{u, v\}) \setminus N(u) \neq \emptyset$, the vertices $N_{G'}(u)$ form a cutset for G' since they separate u from the remainder of the graph. By Lemma 1 at least one vertex $x \in N_{G'}(u)$ is internal in T'. We create a spanning tree T for G by adding vertex v and the edge xv to T'. Since $N_G(u) = N_G(v)$, we obtain a valid spanning tree for G. By construction we have $\text{Leaves}(T) \supseteq \text{Leaves}(T')$ and so $\text{LW}(T) \geq \text{LW}(T') = k' = k$. □



(a) Rule [4]. (b) Rule [5]. (c) Rule [6].

(d) Rule [7].

Fig. 8. Constructions to show the necessity of some reduction rules. To show necessity of Rule [R] we give a graph G that is not reducible without using [R], such that G can be made arbitrarily large without increasing the leaf weight that can be gained in a spanning tree $T \subseteq G$, as described in Section 6.3. The weight of black vertices is 1, whereas white vertices have weight 0.

Reduction Rule 7 Remove edges that bypass a degree-2 weight-0 vertex

Structure: A vertex w of weight 0 such that $N(w) = \{u, v\}$ and $uv \in E$.
Operation: Remove edge uv from G.
Justification: There is always an optimal spanning tree that avoids edge uv.

Lemma 21. *Rule [7] is safe: there is an optimal spanning tree that avoids uv.*

Proof. Suppose T is a spanning tree for G that uses the edge uv. We will prove that there is a spanning tree T' for G that avoids edge uv with $\text{LW}(T') \geq \text{LW}(T)$. Since T is a spanning tree it is acyclic and it must avoid either edge uw or vw. Let e be the avoided edge. Tree T must use the other edge to be spanning. Consider the spanning tree T' that we obtain from T by removing uv and adding edge e. The only vertex whose degree in T' is higher than in T is w, and it has weight 0. So all vertices with positive weight that were leaves in T are also leaves in T. This implies that $\text{LW}(T') \geq \text{LW}(T)$. □

Figure 8 shows that these reduction rules are necessary.

8.2 Structure of a Reduced Instance

We shall see that the exact weights of the vertices are not relevant to the analysis; we only need to distinguish positive-weight vertices from zero-weight vertices.

Definition 11. *If w is a weight function for graph G, then we define the associated black/white coloring* BW_w *as*

$$\underset{w}{\text{BW}}(v) := \begin{cases} \text{BLACK} & \text{if } w(v) \geq 1. \\ \text{WHITE} & \text{if } w(v) = 0. \end{cases}$$

Theorem 12. *Let $\langle G', w', k' \rangle$ be a non-trivial reduced instance of* Weighted Max Leaf$_0$ *with $G' = (V', E')$. There is a black/white colored graph $\langle G^* = (V^*, E^*), c^* \rangle$ such that:*

1. $\langle G^*, c^* \rangle \in \mathcal{C}^b$,
2. $|G^*| \geq |G'|$,
3. $\text{Genus}(G^*) \leq \text{Genus}(G')$,
4. $\text{blackdeg}_{c^*}(G^*) \leq \text{posdeg}_{w'}(G')$,
5. *if there is a spanning tree $T^* \subseteq G^*$ with k leaves that are c^*-black, then there is a spanning tree $T' \subseteq G'$ with $\text{LW}_{w'}(T') \geq k$.*

Proof. The proof consists of several steps. We first show how to construct a black/white colored graph $\langle G^*, c^* \rangle$ that satisfies Conditions (2), (3) and (4). This construction is based on the black/white coloring $\text{BW}_{w'}$ of the reduced graph. We will then show that a spanning tree $T^* \subseteq G^*$ with k leaves that are c^*-black can be converted to a spanning tree $T' \subseteq G'$ with at least k leaves that are $\text{BW}_{w'}$-black. Since every $\text{BW}_{w'}$-black leaf is a vertex of weight value at least 1 by w', this will prove that Condition (5) holds. Finally we will analyze the structure of the black/white colored graph $\langle G^*, c^* \rangle$ and show that it is a member of the class $\mathcal{C}^b$, which will prove that Condition (1) also holds.

Constructing $\langle G^*, c^* \rangle$ We show how to obtain $\langle G^*, c^* \rangle$ by a transformation from G' and the coloring $\text{BW}_{w'}$. The transformation rules are aimed at eliminating the edges between $\text{BW}_{w'}$-black vertices: if $\langle G', \text{BW}_{w'} \rangle$ does not have edges between black vertices, then $\langle G', \text{BW}_{w'} \rangle \in \mathcal{C}^b$.

We treat the construction of $\langle G^*, c^* \rangle$ as an iterative process that starts with graph $\langle G_0, c_0 \rangle := \langle G', \text{BW}_{w'} \rangle$ and step by step transforms it into $\langle G^*, c^* \rangle$. If G_i contains an edge between c_i-black vertices, then we select one such edge and apply a transformation as described below to obtain $\langle G_{i+1}, c_{i+1} \rangle$. Since the transformation operations will not introduce new black-black edges, this process is finite and stops when there are no black-black edges in $\langle G_i, c_i \rangle$, which is then output as $\langle G^*, c^* \rangle$.

So assume that $\langle G_i, c_i \rangle$ has an edge uv with $c_i(u) = c_i(v) = \text{BLACK}$; we show how this black-black edge is eliminated.

- If $\deg_{G_i}(u) = 2$ or $\deg_{G_i}(v) = 2$ then edge uv lies on path component $P = \langle x, s_1, \ldots, s_q, y \rangle$. We remove vertices $s_1, \ldots, s_q$ and add vertices $s'_1, \ldots, s'_{q'}$ with the edges xs'_1, $s'_{q'}y$ and all $s'_j s'_{j+1}$ for $1 \leq j < q$. The new number of inner vertices q' is chosen based on the colors of the endpoints of P: if $c_i(x) = c_i(y)$ then $q' := 3$, and $q' := 4$ otherwise. We set $c_{i+1}(x) := c_i(x)$, $c_{i+1}(y) := c_i(y)$ and choose the colors of the vertices s'_j such that their color differs from their two neighbors, which is always possible. Since no path component in the reduced graph has more than 3 inner vertices by Property (vi) of Definition 6, we always replace a path component by a component that is at least as large: this implies that these types of transformations do not decrease the size of the graph.
- If $\deg_{G_i}(u), \deg_{G_i}(v) \geq 3$ then we remove edge uv and add a new vertex w with $c_{i+1}(w) := \text{WHITE}$ and $N_{G_{i+1}}(w) := \{u, v\}$. Since we insert a vertex in this transformation, this does not decrease the size of the graph.

Since Rule [4] removes vertices of degree-1 we know that $\deg(v) \geq 2$ for all $v \in V'$, which shows that the given list of possibilities is exhaustive. Figure 9 illustrates some transformations.

We will now show how these operations can be reversed on a spanning tree without decreasing the number of black leaves, to establish Condition (5). Assuming we have a spanning tree $T_{i+1} \subseteq G_{i+1}$ with k leaves that are c_{i+1}-black, we describe how to transform it to a spanning tree $T_i \subseteq G_i$ with at least k leaves that are c_i-black.

- If $\langle G_{i+1}, c_{i+1} \rangle$ was obtained by replacing the inner vertices of a path component $P = \langle x, s_1, \ldots, s_q, y \rangle$ to form a new path component $P' = \langle x, s'_1, \ldots, s'_{q'}, y \rangle$:
  - If T_{i+1} connects x and y over the inner vertices of P', then we let T_i connect x and y over the inner vertices of P. Afterwards it holds that $\textsc{Leaves}(T_i) = \textsc{Leaves}(T_{i+1})$.
  - If T_{i+1} does not connect x and y over P' then it avoids exactly one edge e on P'. Since the colors on P' alternate the edge e has one black endpoint, and T_{i+1} gains at most one black leaf inside P'. This transformation was triggered by a black-black edge with a degree-2 endpoint on P; we let T_i use all edges on P, except for one edge incident on the degree-2 black vertex. Then T_i gains at least one black leaf inside P, which implies the number of c_i-black leaves in T_i is at least the number of c_{i+1}-black leaves in T_{i+1}.
- If $\langle G_{i+1}, c_{i+1} \rangle$ eliminated an edge uv with endpoints of degree at least 3 by splitting it with a vertex w:
  - If $\deg_{T_{i+1}}(w) = 2$ then we use the direct edge uv in T_i.
  - If $\deg_{T_{i+1}}(w) = 1$ then we do not use edge uv in T_i.

  Afterwards it holds that $\textsc{Leaves}(T_i) \cap \mathcal{B}(T_i) = \textsc{Leaves}(T_{i+1}) \cap \mathcal{B}(T_{i+1})$.

Structure After Transformations Every transformation of $\langle G_i, c_i \rangle$ to $\langle G_{i+1}, c_{i+1} \rangle$ eliminates one edge between black vertices without introducing new edges between black vertices, and therefore after a finite number of steps we obtain a graph without edges between black vertices; we use this graph as $\langle G^*, c^* \rangle$. Since none of the transformation steps decrease the size of the graph we have $|G_{i+1}| \geq |G_i|$ and therefore by induction we have $|G^*| \geq |G_0| = |G'|$ to satisfy Condition (2). Since the transformation steps locally replace path

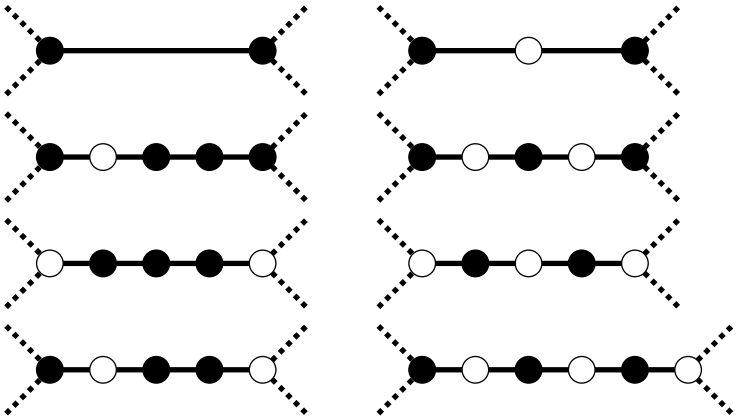


Fig. 9. Examples of the transformations that are applied to $\langle G', \text{BW}_{w'} \rangle$ to construct $\langle G^*, c^* \rangle \in \mathcal{C}^b$. The structures on the left are black-black edges before the transformation; the structures on the right show how they are transformed.

components (planar structures) by other path components (also planar structures), they do not increase the genus of the graph. It is not hard to verify that the transformation steps do not increase the degree of black vertices either. Since $\text{BLACKDEG}_{\text{BW}_{w'}}(G') = \text{POSDEG}_{w'}(G')$ this proves that Condition (4) also holds.

It remains to prove that $\langle G^*, c^* \rangle \in \mathcal{C}^b$. We will prove this by showing that every property of Definition 6 holds for $\langle G^*, c^* \rangle$.

(i) The reduced graph G' must be simple and connected, because any input graph is required to be so and it is trivial to verify that the reduction rules do not change this. Since the transformation to G^* corresponds to subdividing edges, the graph G^* must be simple and connected as well.

(ii) By Rule [2] the graph G' is not isomorphic to a simple cycle. Since the transformation from G' to G^* corresponds to subdividing edges by degree-2 vertices, the graph G^* is not isomorphic to a simple cycle either.

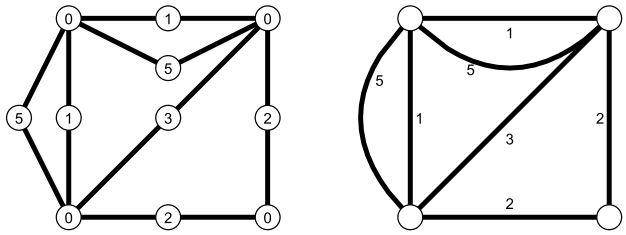
(iii) By Rule [5] the graph $\langle G', w' \rangle$ has no edges between vertices of weight 0, which implies that $\langle G', \text{BW}_{w'} \rangle$ does not have edges between white vertices. The transformation to $\langle G^*, c^* \rangle$ did not change the color of any vertices of degree 3 or more. Since the transformation eliminated all edges between black vertices by inserting path components with a suitable alternating black/white coloring, this transformation did not introduce any edges between white vertices. This implies that $\langle G^*, c^* \rangle$ must be bipartite according to c^*.

(iv) By Rule [4] the graph G' has no vertices of degree 1, hence G' has minimum degree 2 or more. Since the transformation to G^* only inserted vertices with degree 2 or higher and did not decrease the degree of any existing vertices in the graph we know that the minimum degree of G^* is at least 2.

(v) By Rule [6] the graph G' does not have 2 distinct degree-2 vertices u, v of weight 0 such that $N_{G'}(u) = N_{G'}(v)$. Therefore $\langle G', \text{BW}_{w'} \rangle$ does not have 2 distinct $\text{BW}_{w'}$-white degree-2 vertices with the same neighborhood. Whenever a black-black edge with a degree-2 endpoint is eliminated in the transformation to $\langle G^*, c^* \rangle$, the path component is replaced by a path component of length at least 3 and therefore this does not introduce any bad situations. Whenever a black-black edge is eliminated with endpoints x, y that both have degree at least 3, then this edge is replaced by a degree-2 white vertex w with $N(w) := \{x, y\}$. But this transformation does not violate Property (v) either, because whenever there is such a black-black edge in $\langle G', \text{BW}_{w'} \rangle$ with degree ≥ 3 endpoints there is no white degree-2 vertex with those endpoints as neighbors by Rule [7].

(vi) By Rule [1] the size of path components in G' is bounded by 3. The transformation to G^* does not change the degrees of p-component endpoints, but only replaces the inner vertices of path components to a maximum length of 4. Therefore the size of p-components in G^* is bounded by 4.

This proves that $\langle G^*, c^* \rangle \in \mathcal{C}^b$, which concludes the proof of Theorem 12. □



(a) Reduced $\langle G', w', k' \rangle$. (b) Multigraph G''.

Fig. 10. Example of the transformation from a reduced instance $\langle G', w', k' \rangle$ with $\text{POSDEG}_{w'}(G') \leq 2$ to a multigraph G'' with edge-weight function f; there is a spanning tree $T' \subseteq G'$ with $\text{LW}(T') = k$ if and only if there is a spanning tree $T'' \subseteq G''$ such that the combined weight of the edges not used by T'' is k.

8.3 Kernelization Algorithm

There is a special case of Weighted Max Leaf$_0$ that can be solved in polynomial time, which is shown by the following lemma.

Lemma 22. *Every instance $\langle G, w, k \rangle$ of* Weighted Max Leaf$_0$ *with* $\text{POSDEG}_w(G) \leq 2$ *can be solved in polynomial time.*

Proof. Consider an instance $\langle G, w, k \rangle$ that satisfies the stated requirement. We apply the reduction rules to obtain a reduced instance $\langle G', w', k' \rangle$ in polynomial time. Since the reduction rules do not increase the degrees of positive-weight vertices, and since the minimum degree of G' is at least 2 by Rule [4] we can conclude that every positive-weight vertex in G' has degree 2.

Consider the multigraph $G'' = (V'', E'')$ with edge weight function $f : E'' \mapsto \{0\} \cup \mathbb{N}$ that is defined as follows. We take $V'' := \{v \in V' \mid w'(v) = 0\}$. Every $v \in V'$ with $w'(v) > 0$ must have $\deg_{G'}(v) = 2$; let x, y be its neighbors. For every such v we add an edge xy to E'' and set $f(xy) := w'(v)$. Figure 10 shows an example of this transformation.

There is a 1-to-1 correspondence between leaf-weighted spanning trees for G' and edge-weighted spanning trees for G'': there is a spanning tree $T'' \subseteq G''$ such that the total weight of the edges *not* used by T'' is k, if and only if there is a spanning tree $T' \subseteq G'$ with leaf weight k. This shows that the Weighted Max Leaf$_0$ instance can be solved by solving an instance of Minimum Edge-Weight Spanning Tree, which can be done in polynomial time [24]. □

Theorem 13. *The* Weighted Max Leaf$_0$ *problem has a kernel with at most* $(24H(\gamma)-18)k+10\gamma$ *vertices when restricted to graphs of oriented genus at most γ, and a kernel with at most* $(1+4\delta)(1+\frac{\delta}{2})k$ *vertices on graphs where the degree of positive-weight vertices is bounded by δ.*

Proof. We prove this theorem by presenting a kernelization algorithm that achieves the desired reduction. We assume that the algorithm receives an instance $\langle G, w, k \rangle$ of Weighted Max Leaf$_0$, along with a bound γ on the genus of the graph, or a bound δ on the degree of the positive-weight vertices. The algorithm exhaustively applies the reduction rules to obtain a reduced instance $\langle G', w', k' \rangle$. We then test for some simple cases. If the reduced instance is trivial ($|G'| \leq 2$ or $k' \leq 0$) then we solve the problem in polynomial time and output a 1-vertex instance with the same answer as described in Section 6.1. If the algorithm was supplied with a bound $\delta \leq 2$ then we can also solve the problem in polynomial time using Lemma 22. For the remaining cases, we use the structure theory about leafy spanning trees in bipartite graphs to determine that large reduced instances must be yes instances. If $|G'| \geq (24H(\gamma) - 18)k' + 10\gamma$ (in the bounded genus case) or $|G'| \geq (1 + 4\delta)(1 + \frac{\delta}{2})k'$ (in the bounded degree case) then the algorithm outputs a trivial yes instance consisting of 1 vertex because the answer must be yes. If the size of G' does not exceed this bounds, then the reduced instance is sufficiently small and the algorithm outputs $\langle G', w', k' \rangle$.

To show that the algorithm is correct when the size of G' is large, we prove that in such cases G has a spanning tree with leaf weight at least k. By the safety of the reduction rules this corresponds to proving that G' has a spanning tree with leaf weight at least k'.

Let $\langle G^*, c^* \rangle \in \mathcal{C}^b$ be a black/white colored graph derived from the instance $\langle G', w', k' \rangle$ as described by Theorem 12. When the algorithm is executed, it is either supplied a bound γ on the genus of the input graph, or a bound δ on the degree of positive-weight vertices in the input. We prove these two cases separately.

The Algorithm Receives a Genus Bound γ We first establish an upper bound on the genus of G^*.

$$\begin{aligned}
\textsc{Genus}(G^*) &\leq \textsc{Genus}(G') && \text{By Property (3) of Theorem 12.} \\
&\leq \textsc{Genus}(G) && \text{Reduction rules do not increase genus.} \\
&\leq \gamma && \text{Since } \textsc{Genus}(G) \leq \gamma \text{ is given.}
\end{aligned} \tag{11}$$

Using this bound on the genus, we can lower bound the number of vertices in G^* that are colored black by c^*.

$$\begin{aligned}
\mathcal{B}_{c^*}(G^*) &\geq \frac{|G^*| - 10\,\textsc{Genus}(G^*)}{6} && \text{By Theorem 7.} \\
&\geq \frac{|G'| - 10\,\textsc{Genus}(G^*)}{6} && \text{By Property (2) of Theorem 12.} \\
&\geq \frac{|G'| - 10\gamma}{6} && \text{By Equation (11).} \\
&\geq \frac{(24H(\gamma) - 18)k' + 10\gamma - 10\gamma}{6} && \text{Since } |G'| \geq (24H(\gamma) - 18)k' + 10\gamma \text{ by assumption.} \\
&= (4H(\gamma) - 3)k' && \text{Simplifying.}
\end{aligned} \tag{12}$$

We will use this lower bound on the number of black vertices in combination with Theorem 5. To apply Theorem 5 we need the following step to bound the value β that is defined in that theorem. When we apply the statement of the theorem to the graph $\langle G^*, c^* \rangle$, we can derive the following about the value of β:

$$\begin{aligned}
\beta &= \min(H(\textsc{Genus}(G^*)) - 2, \underset{c^*}{\textsc{blackdeg}}(G^*) - 1) && \text{By the definition in Theorem 5.} \\
&\leq H(\textsc{Genus}(G^*)) - 2 && \text{By definition of min.} \\
&\leq H(\gamma) - 2 && \text{By Equation (11) since } \forall x \geq 0 : H(x+1) \geq H(x).
\end{aligned} \tag{13}$$

With this information we can apply Theorem 5. Consider a spanning tree $T^* \subseteq G^*$ that maximizes the number of c^*-black leaves of T^*.

$$\begin{aligned}
|\,\textsc{Leaves}(T^*) \cap \mathcal{B}_{c^*}(T)| &\geq \mathcal{B}_{c^*}(G^*)/(5 + 4\beta) && \text{By Theorem 5.} \\
&\geq (4H(\gamma) - 3)k'/(5 + 4\beta) && \text{By Equation (12).} \\
&\geq (4H(\gamma) - 3)k'/(5 + 4(H(\gamma) - 2)) && \text{By Equation (13).} \\
&\geq (4H(\gamma) - 3)k'/(4H(\gamma) - 3) && \text{Simple arithmetic.} \\
&\geq k' && \text{Dividing.}
\end{aligned}$$

We have shown that G^* has a spanning tree with at least k' leaves that are c^*-black, which proves by Property (5) of Theorem 12 that there is a spanning tree $T' \subseteq G'$ with $\textsc{lw}(G') \geq k'$. By the arguments given earlier this proves that $\langle G, w, k \rangle$ is a YES-instance.

The Algorithm Receives a Degree Bound δ In this case we start by deriving an upper bound on the degree of a black vertex in $\langle G^*, c^* \rangle$.

$$\begin{aligned}
\underset{c^*}{\textsc{blackdeg}}(G^*) &\leq \underset{w'}{\textsc{posdeg}}(G') && \text{By Property (4) of Theorem 12.} \\
&\leq \underset{w}{\textsc{posdeg}}(G) && \text{Reduction does not increase degrees of positive-weight vertices.} \\
&\leq \delta && \text{Since } \textsc{posdeg}_w(G) \leq \delta \text{ is given.}
\end{aligned} \tag{14}$$

As before, we use the derived information to lower bound the number of black vertices.

$$\begin{aligned}
\mathcal{B}_{c^*}(G^*) &\geq \frac{|G^*|}{1+\frac{1}{2}\,\textsc{blackdeg}_{c^*}(G^*)} && \text{By Theorem 7.}\\
&\geq \frac{|G'|}{1+\frac{1}{2}\,\textsc{blackdeg}_{c^*}(G^*)} && \text{By Property (2) of Theorem 12.}\\
&\geq \frac{|G'|}{1+\frac{\delta}{2}} && \text{By Equation (14).}\\
&\geq \frac{(1+4\delta)(1+\frac{\delta}{2})k'}{1+\frac{\delta}{2}} && \text{Since } |G'| \geq (1+4\delta)(1+\tfrac{\delta}{2})k' \text{ by assumption.}\\
&= (1+4\delta)k' && \text{Simplifying.} \qquad (15)
\end{aligned}$$

We once again upper bound the β value for the graph $\langle G^*, c^*\rangle$.

$$\begin{aligned}
\beta &= \min(H(\textsc{Genus}(G^*)) - 2, \underset{c^*}{\textsc{blackdeg}}(G^*) - 1) && \text{By the definition in Theorem 5.}\\
&\leq \underset{c^*}{\textsc{blackdeg}}(G^*) - 1 && \text{By definition of min.}\\
&\leq \delta - 1 && \text{By Equation (14).} \qquad (16)
\end{aligned}$$

And similarly to the previous case, we now apply Theorem 5 using the statements derived so far. Consider a spanning tree $T^* \subseteq G^*$ that maximizes the number of c^*-black leaves of T^*.

$$\begin{aligned}
|\,\textsc{Leaves}(T^*) \cap \mathcal{B}_{c^*}(T)| &\geq \mathcal{B}_{c^*}(G^*)/(5+4\beta) && \text{By Theorem 5.}\\
&\geq (1+4\delta)k'/(5+4\beta) && \text{By Equation (15).}\\
&\geq (1+4\delta)k'/(5+4(\delta-1)) && \text{By Equation (16).}\\
&\geq (1+4\delta)k'/(1+4\delta) && \text{Simple arithmetic.}\\
&\geq k' && \text{Dividing.}
\end{aligned}$$

As earlier this establishes that $\langle G, w, k\rangle$ is a YES-instance.

Conclusion of the Proof We have just argued that the algorithm is correct when it outputs a trivial YES instance based on the size of the reduced graph. Since all reduction rules are safe, the algorithm is also correct when it outputs the reduced instance $\langle G', w', k'\rangle$ of bounded size. The reduced instance must satisfy the same bounds on the genus and degree of positive-weight vertices as the input graph, since the reduction rules do not increase these values.

We can obtain a reduced instance in polynomial time, as is shown in Section 8.4. The reduction rules do not increase k (so $k' \leq k$), which proves that the algorithm satisfies all requirements of a kernel as stated in Definition 1. □

Corollary 4. *The* Weighted Max Leaf$_0$ *problem has a kernel with at most* $78k$ *vertices when restricted to planar graphs.*

Proof. This is a direct result of Theorem 13 since the genus of a planar graph is 0. □

8.4 Implementation of the Kernelization Algorithm

Before describing the implementation of the kernel, we give the following lemma that simplifies the implementation.

Lemma 23. *Suppose* $\langle G, w, k\rangle$ *with* $G = (V, E)$ *is an instance of* Weighted Max Leaf$_0$ *such that there are no edges between weight-0 vertices, no vertices of degree* 1 *and such that all cut vertices have a weight of* 0*. After application of Rule [6] or Rule [7] to obtain* $\langle G', w', k'\rangle$ *with* $G' = (V', E')$ *the following statements hold:*

(i) $\langle G', w', k'\rangle$ *does not have edges between weight-0 vertices,*
(ii) either G' *has no vertices of degree* 1*, or* G' *has only two vertices of positive weight,*
(iii) every cut vertex in G' *has a weight of* 0 *under* w'*.*

Proof. It is easy to see that (i) must hold, since G does not have edges between weight-0 vertices and the mentioned rules do not add new edges or change any vertex weights.

Let us turn our attention to (ii). Assume that G' contains a vertex v of degree 1. Since G has minimum degree at least 2, the degree of v must have decreased by the applied reduction rule. Since one application of Rule [6] or Rule [7] only removes a single edge or a single degree-2 vertex, one application of such a rule can only decrease the degree of a vertex by one. So if $\deg_{G'}(v) = 1$ then it must hold that $\deg_G(v) = 2$. We distinguish based on the weights of the neighbors of v in G. Let $N_G(v) = \{x, y\}$.

- If $w(x), w(y) > 0$ then Rule [6] will not remove x or y, since that rule only removes vertices of weight-0. So the decrease of the degree of v must have been caused by the application of Rule [7]. But Rule [7] only removes edges between positive-weight vertices $\{a, b\}$ if there is a degree-2 weight-0 vertex with $\{a, b\}$ as its neighborhood. Since the only vertices adjacent to v have positive-weight, this rule cannot have been triggered. This implies that the resulting situation cannot have been a result of an application of Rule [6] or Rule [7], which contradicts the assumption on $\langle G', w', k' \rangle$ at the start of the proof.
- Suppose $w(x) = w(y) = 0$. Since Rule [7] only removes edges between positive-weight vertices, application of that rule cannot cause v to lose x or y as a neighbor in G'. So if $\deg_{G'}(v) = 1$ then this must have been caused by Rule [6].
  If $\deg_G(x) > 2$ or $\deg_G(y) > 2$ then Rule [6] can not have deleted x or y. Since the degree of v has decreased from 2 to 1, it must hold that $\deg_G(x) = \deg_G(y) = 2$. If $N_G(x) \neq N_G(y)$ then Rule [6] does not apply so x and y must still exist in G'. This implies that if $\deg_{G'}(v) = 1$ then $N_G(x) = N_G(y) = \{v, z\}$ for some vertex z. Since z is adjacent to x and y that both have weight 0, we find that $w(z) > 0$ since G does not have edges between weight-0 vertices by assumption. If $V' \setminus \{v, x, y, z\} = \emptyset$ then v and z are the only positive-weight vertices in G', so the claim is true. Otherwise the vertex z must be a cut vertex since its removal disconnects the vertices v, x, y from the remainder of the graph; but since $w(z) > 0$ and z is a cut vertex, this contradicts the assumption that all cut vertices have a weight of 0.
- In the remaining case we may assume without loss of generality that $w(x) > 0$ and $w(y) = 0$. Rule [6] cannot cause y to be deleted in this case, so if the degree of v decreases then this must be because the edge vx is removed by Rule [7]. Now observe that if $V' \setminus \{v, x, y\} = \emptyset$ then v and x are the the only positive-weight vertices in G' and the claim is true; otherwise we find that x is a cut vertex of positive weight, which is a contradiction.

Since this case analysis is exhaustive we have established that (ii) holds. To show the validity of (iii) we make a distinction on the reduction rule that was applied.

- If Rule [7] was applied, then this rule deleted an edge $uv \in E$ with $w(u), w(v) > 0$ because there was a vertex r with $w(r) = 0$ and $N_G(r) = \{u, v\}$. Now assume there is a cut vertex $z \in V'$ of positive weight. Then there are two distinct vertices $a, b \in V'$ such that there is no path between a and b in $G' - \{z\}$. Since all cut vertices in $\langle G, w, k \rangle$ have a weight of 0 by assumption, z was not a cut vertex in V' and there is a path P in $G - \{z\}$ that connects a and b. If this path does not use edge uv then it is also a path that connects a and b in $G' - \{z\}$, contradicting the assumption that z is a cut vertex. So P must use edge uv. But now consider the path P' that is obtained from P by replacing all occurrences of edge uv by the successive edges ur and rv. Note that $r \neq z$ since $w(r) = 0$ and $w(z) > 0$. So we find that P' must be a path in $G' - \{z\}$ that connects a and b, which contradicts the assumption that z is a cut vertex; this proves the claim for this case.
- If Rule [6] was applied, then this rule deleted a vertex $v \in V$ with $\deg_G(v) = 2$ and $w(v) = 0$ because there was a vertex $u \neq v$ with $w(u) = 0$ and $N_G(u) = N_G(v)$. Let $N_G(v) = \{x, y\}$. Since $w(v) = 0$ and $\langle G, w, k \rangle$ has no edges between weight-0 vertices, we know that $w(x), w(y) > 0$. Suppose there is a cut vertex $z \in V'$ with $w'(z) > 0$. Then there are two distinct vertices $a, b \in V'$ such that there is no path between a and b in $G' - \{z\}$. Since no positive-weight vertex is a cut vertex in G by assumption, there is a path P in G that does not use z, and that connects a and b. If P does not use the deleted vertex v then P is also a path in G' that connects a and b and avoids z; since this contradicts our choice of a and b, the path P must use the deleted vertex v. If P uses the deleted vertex v, then we can replace the occurence of v in this path by the occurrence of u since $N_G(u) = N_G(v) = N_{G'}(u)$. Since $w(z) > 0$ and $w(u) = 0$ it follows that $z \neq u$. But this means that there is a path P' in $G' - \{z\}$ that connects a and b, which is a contradiction.

Since we have shown that none of the two reduction rules can introduce positive-weight cut vertices, this concludes the proof. □

Lemma 24. *Let $\langle G, w, k\rangle$ be an instance of* Weighted Max Leaf$_0$ *with $G = (V, E)$ that contains c positive-weight vertices. Assuming that basic arithmetic on the vertex weights can be done in constant time, the problem can be decided in $O(2^c(|V| + |E|))$ time.*

Proof. We use the fact that there is a spanning tree $T \subseteq G$ with $\textsc{Leaves}(T) = S$ if and only if $V \setminus S$ is a connected dominating set. For a given set S it is easy to test whether it forms a connected dominating set by performing a Breadth-First Search. Using this information we can decide the problem as follows.

Let P be the positive-weight vertices in the instance. We test all subsets $S \subseteq P$ and see whether $V \setminus S$ forms a connected dominating set in $O(|V| + |E|)$ time. If $V \setminus S$ is a CDS then we know that there is a spanning tree for G with leaf weight $\sum_{v \in S} w(v)$. We try all $2^{|P|} = 2^c$ possible subsets of P, compute the largest possible leaf weight and test whether this is at least k. Since each subset takes $O(|V| + |E|)$ time we can decide the problem in $O(2^c(|V| + |E|))$ time. □

Theorem 14. *Let $\langle G, w, k\rangle$ be an instance of* Weighted Max Leaf$_0$ *with $G = (V, E)$. Assuming that basic arithmetic on the vertex weights can be done in constant time, we can compute an instance $\langle G', w', k'\rangle$ with $G' = (V', E')$ in $O(|V| + |E|)$ time such that:*

- *no reduction rule is applicable to $\langle G', w', k'\rangle$,*
- *$|V'| \leq |V|$,*
- *$|E'| \leq |E|$,*
- *$k' \leq k$,*
- *the answer to decision problem $\langle G, w, k\rangle$ is the same as the answer to $\langle G', w', k'\rangle$.*

Proof. We present a high-level sketch of an implementation of the reduction rules. The kernelization proceeds in several phases. Each phase inspects and modifies the graph in $O(|V| + |E|)$ time. The correctness of this approach is based on Lemma 23 and Lemma 24, along with the following observations.

- If Rule [2] is applicable then we can determine the answer to the decision problem in $O(|V| + |E|)$ time, so we need no further kernelization.
- If the graph does not contain edges between vertices of weight 0, then an application of Rule [1] can only create a single edge between vertices of weight 0; this edge must lie on the reduced p-component and it is trivial to reduce it by Rule [5] in constant time.
- The weight that is assigned to cut vertices does not affect the outcome of the Weighted Max Leaf$_0$ decision problem. Cut vertices never contribute to the leaf weight of a spanning tree, since they are never leaves in a spanning tree by Lemma 1. This will be exploited in the reduction algorithm by giving every cut vertex a weight of 0, that possibly allows it to be reduced even further by contracting edges between neighboring vertices of weight 0. It should be noted that this re-weighting of cut vertices merely helps in obtaining a linear time reduction algorithm; it is *not* needed to establish the fact that all reduced instances have large weight spanning trees.

The implementation of these phases uses an adjacency-list representation of the graph G. We assume that in the adjacency list of u, we store with every entry v that represents one direction of the edge uv a pointer to the entry containing u in the adjacency list of v; this ensures that when we are iterating over the neighbors of u and we find v, we can remove the edge uv in constant time if needed. With this in mind we present the phases of the reduction algorithm.

1. Find all cut vertices in the graph using Depth-First Search, and change their weight to 0. As argued before, this does not change the answer to the decision problem. Refer to [24] for more information on how this can be done in $O(|V| + |E|)$ time.
2. Initialize a stack S that contains all vertices of weight 0. While S is not empty, pop a vertex s from S and take the following steps.
   (a) If s was already visited during this phase, do nothing.
   (b) Otherwise, make a new isolated vertex x and give it a weight of 0.
   (c) Consider the graph $G_0 := G[\{v \in V \mid w(v) = 0\}]$ induced by the weight-0 vertices. Vertex s is in a connected component of G_0. We use a Breadth-First Search to visit all weight-0 vertices in the same connected component of G_0 as s. For every vertex v that we visit during this BFS, we take the following steps:
      i. For all $u \in N_G(v)$ remove the edge uv from G and add the edge ux if it does not exist already. With every vertex we store a pointer to the last vertex that has become its neighbor, allowing us to test whether $ux \in E$ in constant time.

ii. If u has a degree of 1 (possibly after multiple weight-0 vertices have removed their edges to it), remove it from G, decrease k by $w(u)$ and also remove the edge ux from G.

After the execution of this phase the graph does not contain vertices of degree 1, all cut vertices have a weight of 0 and there are no edges between vertices of weight 0. Since we spend $O(\deg(v))$ time for every weight-0 vertex v that we visit and every vertex is visited at most once, this phase can be executed in $O(|V| + |E|)$ time.

3. The next phase consists of the execution of Rule [6] and Rule [7] where this is possible. For every $v \in V$ with $w(v) > 0$ we do the following:
   (a) Iterate over the neighbors of v. For every $u \in N_G(v)$ with $\deg_G(u) = 2$ such that $N_G(u) = \{v, w\}$ for some vertex w:
      – if $w.\text{LAST} \neq v$ then set $w.\text{LAST} := v$.
      – otherwise, if $w.\text{LAST} = v$ then remove u and its incident edges because u needs to be deleted by Rule [6], since there was some other neighbor $x \in N_G(v)$ of degree 2 with $N_G(u) = N_G(x) = \{v, w\}$ which has already set the LAST pointer.
   (b) Iterate again over the neighbors of v. Inspect the vertices $u \in N_G(v)$ and for every u such that $u.\text{LAST} = v$, remove the edge vu since it needs to be removed by Rule [7] because there was a neighbor $x \in N_G(v)$ with $N_G(x) = \{u, v\}$.

   During the execution of this phase we keep track of the number of positive-weight vertices in the graph. If this number drops to 2, then we solve the problem in $O(|V|+|E|)$ time using Lemma 24. If this number stays above 2 then we know by Lemma 23 that this phase has not introduced any degree-1 vertices or edges between weight-0 vertices. So after this phase, the graph cannot be reduced by Rule [4], Rule [5], Rule [6] or Rule [7].
4. This phase consists of applying Rule [1] where this is possible. The implementation can be done in $O(|V| + |E|)$ time by performing a Depth-First Search from a vertex of degree at least 3, as described in Section 7.4. The execution of Rule [1] may introduce edges between weight-0 vertices in the graph, but only if the new vertex that is created by the rule has a weight of 0. We can easily test in constant time whether this is the case, and contract the edge between the weight-0 vertices if needed.
   So after this phase is completed, the graph is not reducible by any rule except possibly Rule [2].
5. The final phase consists of checking whether G is a simple cycle. If this is the case we compute the answer to the decision problem in $O(|V| + |E|)$ time and output a trivial YES or NO instance that is consistent with the answer to $\langle G, w, k\rangle$.

After execution of all phases we have obtained an instance that is not reducible by any rule, that is not larger than the original instance, and that has preserved the answer to the original decision problem. Since each phase can be executed in $O(|V| + |E|)$ time and there are a constant number of phases, this completes the proof of Theorem 8.4. □

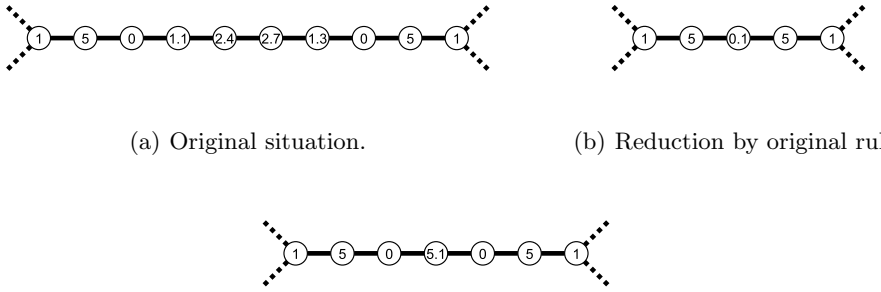
8.5 Extension to Rational Vertex Weights

The extension of this kernel to handle vertex weights that are rational numbers is not as straight-forward as for the WEIGHTED MAX LEAF$_+$ kernel.

The first problem is that the WEIGHTED MAX LEAF problem with weights that are allowed to be arbitrary positive rational numbers is not in FPT, since it is NP-complete for $k = 1$ by a reduction from regular MAX LEAF that gives all vertices a weight of $\frac{1}{|G|}$. Therefore we need to establish a lower bound on positive weights that may be assigned; without loss of generality we can fix this bound at 1.

So we now consider whether the given kernel can be adapted to work for vertex weights that are elements of $\{0\} \cup \mathbb{Q}_{\geq 1}$. This is indeed the case, but we need to modify Rule [1] to make this possible. The problem with the rule in its existing form is that it may assign a vertex a weight in the interval $(0, 1)$, which is not allowed. Image 11 shows an example of a long path-component that would cause a vertex s' with $w'(s') \in (0, 1)$ to be created according to the original reduction rule. Recall that $w'(s')$ is defined as $w'(s') := \max_{i=1}^{q-1}(w(s_i)+w(s_{i+1})-\max(w(s_1), w(s_q)))$, where the vertices $s_1, \ldots, s_q$ are the degree-2 vertices in the interior of the path component. It follows that if $\max(w(s_1), w(s_q)) = \epsilon + \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1})$ for some $0 < \epsilon < 1$ which can happen if $w'(s_2) = w'(s_{q-1}) = 0$, then the new vertex s' gets a weight of ϵ which is not allowed.

This issue can be solved in the following way. We reduce the graph by removing vertices $s_2, \ldots, s_{q-1}$ and their incident edges, and adding three vertices s'_2, s'_3, s'_4 of degree 2, with edges to make them a path component. We then set $w'(s'_2) := 0$, $w'(s'_4) := 0$ and $w'(s'_3) = \max_{i=1}^{q-1}(w(s_i) + w(s_{i+1})$. Since the weight



(a) Original situation.

(b) Reduction by original rule.

(c) Reduction by adapted rule.

Fig. 11. Example of a path component weighted with elements of $\{0\} \cup \mathbb{Q}_{\geq 1}$ that results in a vertex v with weight $0 < w'(v) < 1$ after reduction by the original Rule [1]. The last subfigure shows how this structure is reduced by the adapted rule, which results in longer path components.

of s'_3 is computed as a sum of existing weights, it must be either 0 or a rational number of at least 1 which shows that this alternative reduction rule ensures that the kernelization is a self-reduction.

The suggested change to the reduction rule causes the size of path-components in the reduced graph to be bounded by 5, rather than by 3 as before. The structure theory that forms the backbone of the kernelization lemma needs to be updated for this change. These changes are straight-forward, but they will cause the constant in the kernel size to increase slightly.

8.6 Discussion

We presented a linear problem kernel for Weighted Max Leaf$_0$ when restricted to graphs of bounded genus, and graphs in which the degree of positive-weight vertices is bounded. There are 6 reduction rules needed by this kernel, which can be implemented efficiently. All reduction rules are parameter-independent, and can also be applied in graphs for which the genus is not known, or the degree of positive-weight vertices is not bounded. Since the reduction rules have an efficient implementation, this suggests that they might be useful as a preprocessing step for any unrestricted instance of Weighted Max Leaf$_0$.

The smallest δ for which the bounded-degree Weighted Max Leaf$_0$ problem is non-trivial is 3. Our technique yields a kernel with $(1+4\delta)(1+\frac{\delta}{2})k = 13 \cdot 2.5k = 32.5k$ vertices for this case. The smallest possible genus is 0 which results in planar graphs, for which we obtained a kernel with $78k$ vertices. Compared to the $3.75k$ kernel for the regular Max Leaf problem, these kernels are still quite large. Let us therefore consider the potential for decreasing the kernel size.

Although the estimation of the kernel size for Weighted Max Leaf$_+$ is tight at $7.5k$ (as described in Section 7.6), this is not the case for Weighted Max Leaf$_0$. The main difficulty is that the proof of the kernel size uses several intermediate steps. Even if each step would use a tight bound, this might still yield to overestimations for the final combined bound. Let us give an example of this issue using the class of planar graphs. The proof that every bipartite planar graph $\langle G, c\rangle \in \mathcal{C}^b$ has a spanning tree with $78k$ black leaves was obtained by the following steps:

- there is a spanning tree $T \subseteq G$ with $|\mathcal{B}_c(G)|/13$ c-black leaves (Theorem 5), and
- the number of black vertices in G is lower bounded as $\mathcal{B}_c(G) \geq \frac{|G|}{6}$ (Theorem 7).

The final bound follows from multiplying the two denominators in these bounds. But it is very well possible that the worst cases of the two lemmas cannot occur on the same graph; for example, all graphs that minimize the possible number of black vertices might have spanning trees with relatively higher fractions of the black vertices as leaves.

With regard to the tightness of the individual bounds - again using planar graphs as an example - we can say the following. The bound of Theorem 5 that every planar $\langle G, c\rangle \in \mathcal{C}^b$ has a spanning tree with at least $|\mathcal{B}_c(G)|/13$ c-black leaves cannot be improved above $|\mathcal{B}_c(G)|/10$, as is illustrated in Appendix C. The bound of Theorem 7 is tight, which can be shown by taking an arbitrary maximally planar graph, coloring

all vertices black, subdividing every edge by a white vertex, and inserting a white vertex into every face of the triangulation and connecting that vertex to all three vertices on the face.

This implies that the technique of multiplying the two denominators of the individual bounds cannot show the existence of a spanning tree with more than $|G|/(6 \cdot 10)$ black leaves. To obtain a better bound it is needed to perform an analysis of spanning trees in black/white colored graphs that are not necessarily bipartite according to their coloring. But even the potential of such an approach is limited: the combined bound that every planar $\langle G, c\rangle \in \mathcal{C}^b$ has a spanning tree with at least $|G|/78$ black leaves cannot be improved above $|G|/31$. This shows that new reduction rules will be needed to obtain a kernel with less than $31k$ vertices.

It is not hard to come up with reduction rules that are able to reduce the graphs that form worst-case or bad-case examples for the current kernel. For example, the well-known reduction rule for Max Leaf that contracts bridges with endpoints of degree at least 2 is also valid in the context of weighted graphs. It is also possible to adapt the KW-dissolver rule that is used in the current-best kernelization for Max Leaf [15] to the weighted setting. One could also think of new rules to reduce the number of weight-0 vertices, such as the following.

Reduction Rule 8 Merge the neighborhoods of weight-0 vertices

> **Structure:** Three independent, distinct weight-0 vertices x, y, z such that $N(x) \subseteq N(y) \cap N(z)$.
> **Operation:** Remove y, z and their incident edges. Add a new vertex w with $N(w) := N(y) \cup N(z)$.
> **Justification:** The set $N(x)$ is a cutset in the reduced graph since it separates x from w. In any spanning tree for the reduced graph, at least one vertex $v \in N(x)$ is internal and we obtain a spanning tree for the original by connecting to y and z from v - which is possible since $v \in N(y) \cap N(z)$.

So the difficulty in improving this kernelization result does not consist of discovering reduction rules, but it consists of *proving* that such reduction rules actually decrease the size of the kernel.

As the last item of this discussion we consider the possibility of solving the construction version of Weighted Max Leaf$_0$ using the kernel. It is easy to see that every reduction rule can be reversed, in order to lift a solution to the reduced problem back to a solution to the original problem. The difficulty in solving the construction version of Weighted Max Leaf$_0$ lies in constructing a spanning tree of sufficient leaf weight when Theorem 5 shows that such a tree must exist. If the kernelization algorithm runs with a bound δ on the degree of positive-weight vertices then this is not hard, but when it runs with a bound on the genus then the algorithm needs to be supplied with an embedding of the graph in a surface of genus at most γ in order to find a good spanning tree. For planar graphs there are efficient algorithms to obtain planar embeddings, but for higher genus values this becomes a problem. Even though the graph genus problem is fixed parameter tractable with respect to the outcome value [30], the algorithms that determine the genus and construct an embedding are not practical at the current state of research [31].

9 Conclusion

In this work we have shown that the fixed parameter complexity of the Weighted Max Leaf problem strongly depends on the range of the weight function. Most of the work on this subject has gone into developing the structure theory about leafy spanning trees in bipartite graphs and the associated kernel. Since the kernels for Weighted Max Leaf$_+$ and Weighted Max Leaf$_0$ have already been discussed in Sections 7.6 and 8.6, we conclude here with some general remarks and pointers to future research.

It would be interesting to determine the exact fixed parameter complexity of Bipartite Max Leaf and Weighted Max Leaf$_0$. We have shown both problems to be hard for $W[1]$, but so far we have not been able to find a non-trivial membership proof for any parameterized hardness class. Another way to improve the classification of the fixed-parameter complexity of these problems is to find a hardness proof for $W[2]$ or higher. In this context it is also natural to ask whether Bipartite Max Leaf and Weighted Max Leaf$_0$ have the same fixed parameter complexity, or whether the latter problem is provably harder.

The structure theory developed in this work has strong connections to approximation theory. Corollary 1 is an example of this. Using the reduction rules in combination with the kernelization lemmas, one can easily prove the existence of linear-time constant factor approximation algorithms for Weighted Max Leaf with weight range $S = \{0, 1\}$ on graphs of bounded genus or graphs in which the degree of positive-weight vertices is bounded. It is an open problem to determine under which restrictions the Weighted Max Leaf problem has a PTAS; an obvious candidate would be Weighted Max Leaf on planar graphs, either using the Planar

Separator Theorem [32] or an algorithm in the style of Baker that relies on an outerplanar decomposition of the graph [33].

There is a long history of ever-improving FPT-algorithms for the regular Max Leaf problem. It might be possible to adapt these algorithms to work on Weighted Max Leaf. The reduction rules and problem structure that is presented in this work might simplify the adaptation.

As the final subject of this work on fixed parameter complexity we discuss the possibility of different parameterizations of the Weighted Max Leaf problem. When we parameterize by the treewidth of the graph it is highly likely that the problem is amenable to a complex dynamic programming procedure that is similar to the dynamic programming to solve Connected Dominating Set on graphs of bounded treewidth. Since the complexity of Weighted Max Leaf when parameterized by the target leaf weight depends on whether or not the weight value of 0 is allowed, this suggests that vertices of weight 0 are what makes Weighted Max Leaf$_0$ hard. Suppose $\langle G, w, k\rangle$ is an instance of Weighted Max Leaf$_0$ with exactly c vertices of weight 0. If we parameterize by the value c, then obviously the problem is still NP-complete for $c = 0$ because it results in the Weighted Max Leaf$_+$ problem. But the situation changes when we parameterize by $k + c$, where k is the target leaf weight. It can be shown that this parameterization leads to fixed parameter tractability since we can eliminate a vertex v of weight 0 by giving it a degree-1 neighbor of weight 1, setting the weight of v to 1 as well, and finally increasing the value of k by one. A further study into the effect of different parameterizations on the fixed-parameter complexity of Weighted Max Leaf seems interesting.

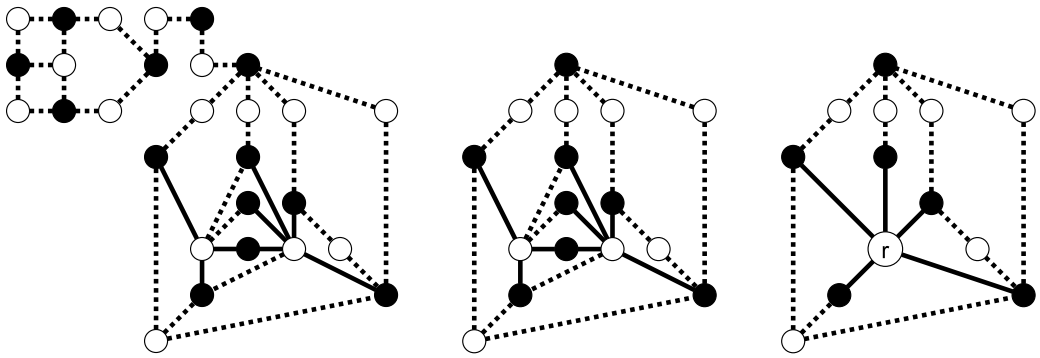
Acknowledgements I would like to thank Hans Bodlaender for his guidance and advice while writing my masters thesis, Peter Rossmanith for a brief but inspriring discussion about rational-valued vertex weights during TACO day '09 and finally Marinus Veldhorst for inspiring me to study spanning tree problems.

References

1. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20** (1998) 374–387
2. Storer, J.A.: Constructing full spanning trees for cubic graphs. Information Processing Letters **13** (1981) 8–11
3. Kranakis, E., Krizanc, D., Ruf, B., Urrutia, J., Woeginger, G.: The VC-dimension of set systems defined by graphs. Discrete Applied Mathematics **77** (1997) 237–257
4. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
5. Lemke, P.: The maximum leaf spanning tree problem for cubic graphs is NP-complete. IMA publication no. 428 (1988)
6. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Information Proccessing Letters (1994)
7. 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
8. Bonsma, P., Zickfeld, F.: A 3/2-approximation algorithm for finding spanning trees with many leaves in cubic graphs. Graph-Theoretic Concepts in Computer Science: 34th International Workshop, WG 2008, Durham, UK, June 30 — July 2, 2008. Revised Papers (2008) 66–77
9. Rahman, M.S., Kaykobad, M.: Complexities of some interesting problems on spanning trees. Information Proccessing Letters **94** (2005) 93 –97
10. Li, B.P., Toulouse, M.: Variations of the maximum leaf spanning tree problem for bipartite graphs. Information Processing Letters **97** (2006) 129–132
11. Fusco, E.G., Monti, A.: Spanning trees with many leaves in regular bipartite graphs. In: ISAAC. (2007) 904–914
12. Hastad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Annual IEEE Symposium on Foundations of Computer Science (1996) 627
13. Trevisan, L.: Inapproximability of combinatorial optimization problems. Electronic Colloquium on Computational Complexity (ECCC) (2004)
14. Fujie, T.: An exact algorithm for the maximum leaf spanning tree problem. Computers & Operations Research **30** (2003) 1931–1944
15. Estivill-Castro, V., Fellows, M., Langston, M., Rosamond, F.: FPT is P-time extremal structure I. In: Proc. 1st ACiD. (2005) 1–41
16. Kneis, J., Langer, A., Rossmanith, P.: A new algorithm for finding trees with many leaves. In: Proc. 19th ISAAC. (2008) 270–281
17. 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
18. Downey, R., Fellows, M.R.: Parameterized complexity. Monographs in computer science. Springer, New York (1999)

19. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels (extended abstract). In: Proc. ICALP. (2008) 563–574
20. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Parameterized algorithms for directed maximum leaf problems. In: ICALP. (2007) 352–362
21. Fernau, H., Fomin, F.V., Lokshtanov, D., Raible, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. In: Proc. 26th STACS, Dagstuhl, Germany (2009)
22. Guo, J., Niedermeier, R.: Linear problem kernels for NP-hard problems on planar graphs. In: Proc. 34th ICALP. Number 4596 in LNCS, Springer (2007) 375–386
23. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) Kernelization. arXiv (2009)
24. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. McGraw-Hill Science/Engineering/Math (2001)
25. Bollobas, B.: Extremal Graph Theory. Academic Press, London (1978)
26. Wilson, R.: Graphs, colourings and the four-colour theorem. Oxford Science Publications (2002)
27. Harary, F.: Graph Theory. Addison Wesley Publishing Company (1969)
28. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
29. Griggs, J.R., Kleitman, D., Shastri, A.: Spanning trees with many leaves in cubic graphs. J. Graph Theory **13** (1989) 669–695
30. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: Feasible Mathematics II, Birkhauser (1994) 219–244
31. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. SIAM Journal on Discrete Mathematics **12** (1999) 6–26
32. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. Symposium on Foundations of Computer Science (1977) 162–170
33. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM **41** (1994) 153–180

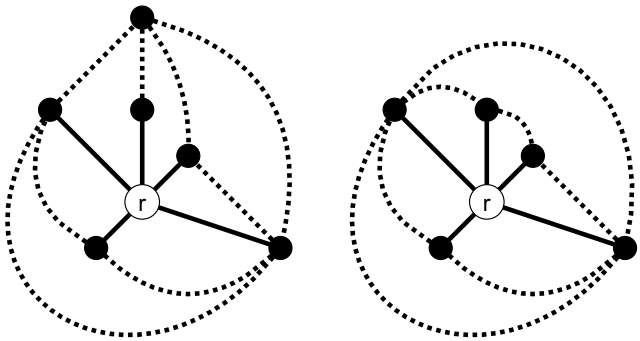
A Example of Covering Graph Construction



(a) Original graph and tree.

(b) Taking the induced subgraph.

(c) Contracting the interior of the tree.



(d) Collapsing X.

(e) Collapsing Y.

Fig. 12. Example of construction of a covering graph. No Scout or Sandwich operations are applicable. Solid lines represent edges in the tree and dotted lines represent edges in the graph.

B Example Graph for Weighted Max Leaf$_+$

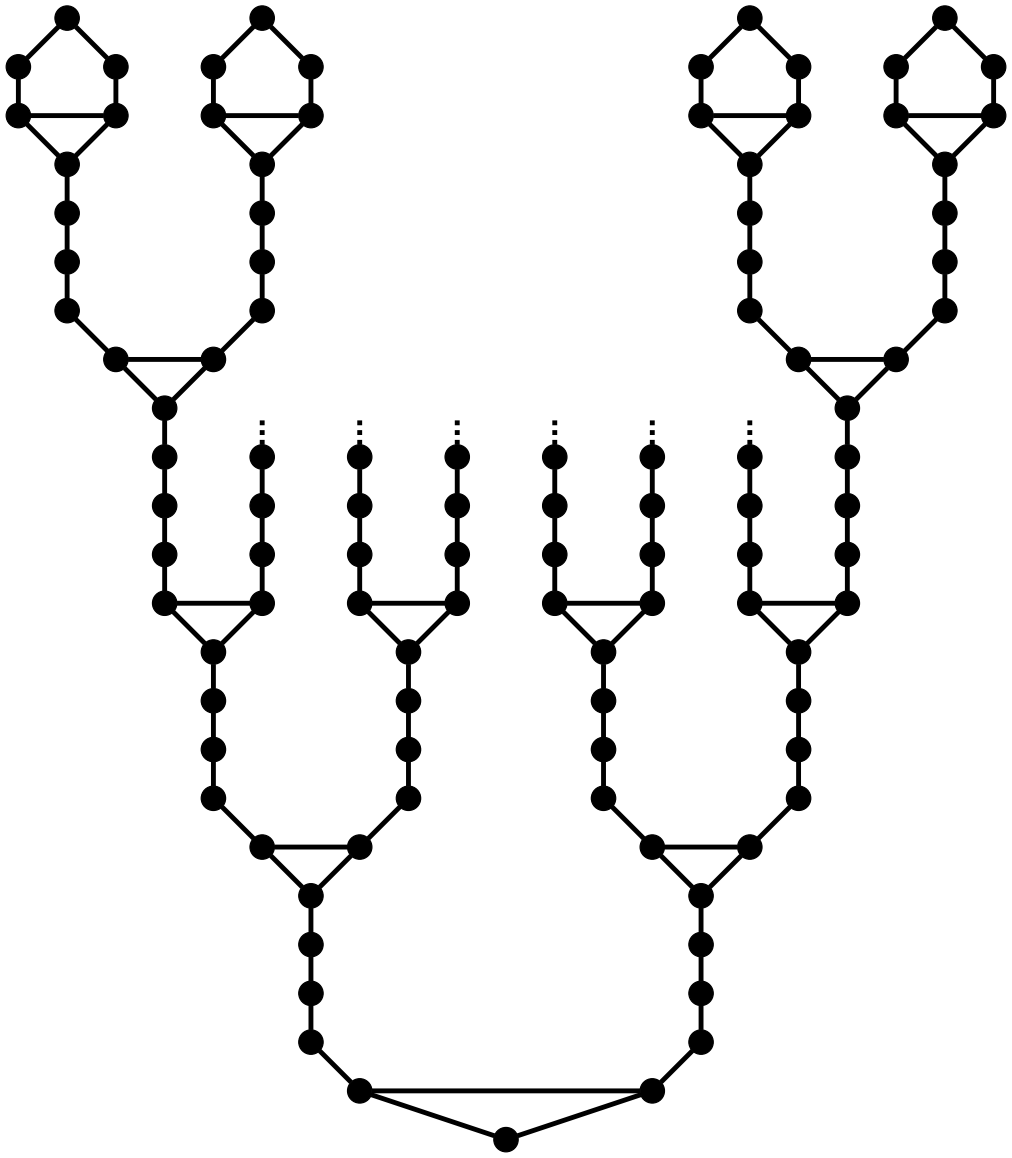


Fig. 13. Example of a graph in $\mathcal{C}^3$ with few leaves in any spanning tree. This graph can be considered to be a subdivision of a complete binary tree of $h = 5$, with a special structure S appended to every leaf of the binary tree. Not all such structures S are shown, for clarity. An optimal spanning tree gains one black leaf in every occurrence of S, and one in the root of the binary tree. In the limit of $h \to \infty$ this yields a graph with $|G|/7.5$ black leaves in an optimal spanning tree.

C Example Graph for Weighted Max Leaf$_0$

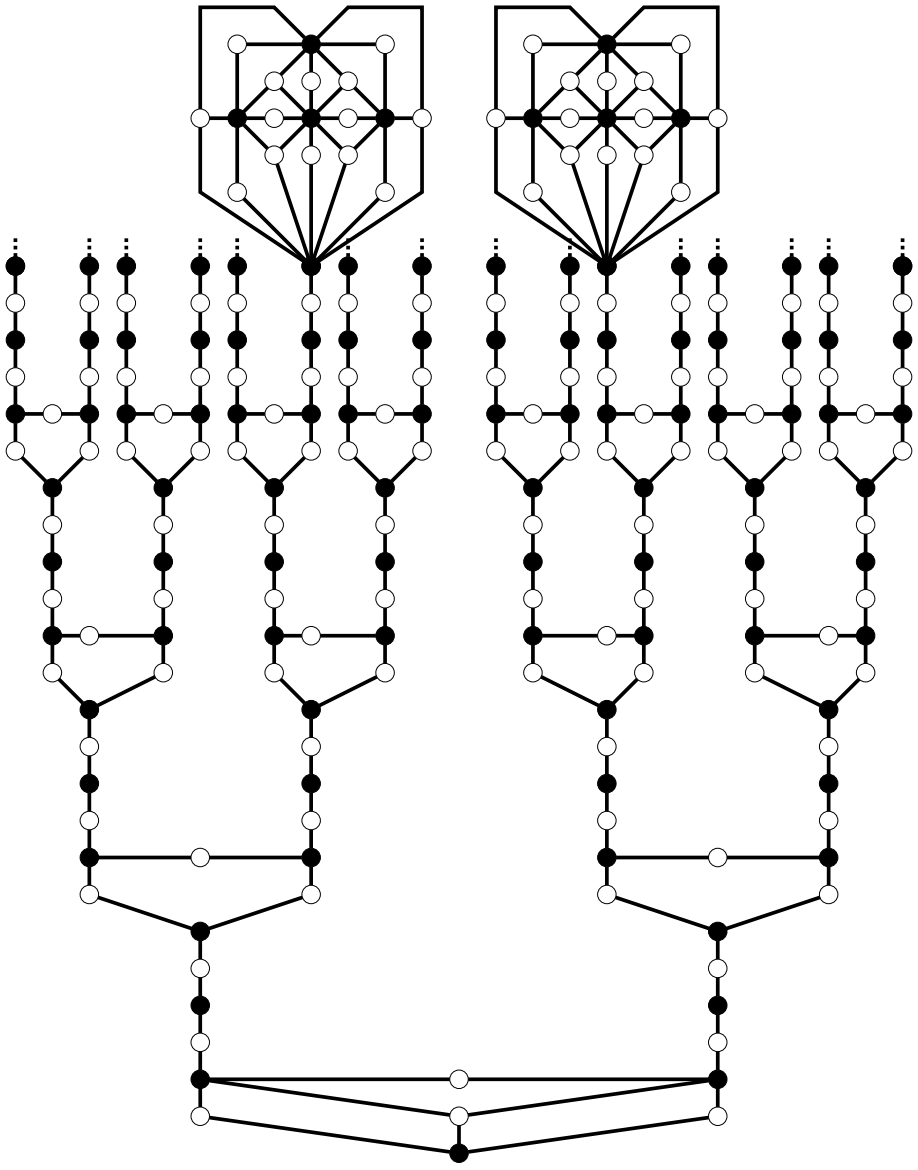


Fig. 14. Example of a graph in $\mathcal{C}^b$ with few black leaves in any spanning tree. This graph can be considered to be a subdivision of a complete binary tree of $h = 4$, with a special structure S appended to every leaf of the binary tree. Not all such structures S are shown, for clarity. An optimal spanning tree gains one black leaf in every occurrence of S, and one in the root of the binary tree. In the limit of $h \to \infty$ this yields a graph with $|\mathcal{B}(G)|/10$ black leaves in an optimal spanning tree, which is $|G|/31$.