A Parameterized Measure-and-Conquer Analysis for Finding a k-Leaf Spanning Tree in an Undirected Graph

Daniel Binkele-Raible, Henning Fernau

Discrete Mathematics and Theoretical Computer Science DMTCS vol. **16:1**, 2014, 179–200

A Parameterized Measure-and-Conquer Analysis for Finding a k-Leaf Spanning Tree in an Undirected Graph

Daniel Binkele-Raible* and Henning Fernau†

Universität Trier, FB 4 — Abteilung Informatikwissenschaften, Trier, Germany

received 8th May 2012, revised 16th Aug. 2013, 8th Jan. 2014, accepted 2nd Mar. 2014.

The problem of finding a spanning tree in an undirected graph with a maximum number of leaves is known to be $\mathcal{NP}$-hard. We present an algorithm which finds a spanning tree with at least k leaves in time $\mathcal{O}^*(3.4575^k)$ which improves the currently best algorithm. The estimation of the running time is done by using a non-standard measure. The present paper is one of the still few examples that employ the Measure & Conquer paradigm of algorithm analysis in the area of Parameterized Algorithmics.

Keywords: parameterized algorithms; measure-and-conquer analysis; maximum leaf spanning tree

1 Introduction

In this paper, we address the following combinatorial problem in graphs:

k-LEAF SPANNING TREE
Given: An undirected graph $G(V, E)$, and the parameter k.
We ask: Is there a spanning tree for G with at least k leaves?

This problem has found notable applications, for instance in the design of ad-hoc sensor networks [8, 34], in network design (see, e.g., [28]) and in computational biology (refer to [27]). According to [23], our problem is also known as the traveling tourist problem, as it models finding the shortest walk that leads to all attractions (modeled itself by a network) or allows at least to look at them (when only visiting the neighborhood of the attraction).

In a spanning tree with k leaves the non-leaf vertices form a connected dominating set with $n - k$ vertices in any graph on n vertices. The corresponding graph parameter, giving the smallest number of vertices forming a connected dominating set in a graph, was introduced in [30]. Hence, finding a spanning tree with a maximum number of leaves is equivalent to finding a minimum connected dominating set. In

*Email: raible@informatik.uni-trier.de
†Email: fernau@informatik.uni-trier.de

particular in application areas, the problem we deal with is hence also known as MINIMUM CONNECTED DOMINATING SET. The computational difficulty (i.e., NP-completeness) of our problem has long been established; we refer to [14] for a discussion of such types of results. However, it should be noticed that with respect to parameterized complexity theory (the focus of the present paper), the two problem variants (i.e., MINIMUM CONNECTED DOMINATING SET versus MAXIMUM LEAF SPANNING TREE) turn out to have a completely different flavor. The same comment applies to approximability (as discussed in the next paragraph).

Generally speaking, while MAXIMUM LEAF SPANNING TREE can be approximated up to a constant factor, this is not to be expected for MINIMUM CONNECTED DOMINATING SET. The MAXIMUM LEAF SPANNING TREE problem already has been widely studied with regard to its approximability. R. Solis-Oba [33] obtained a 2-approximation running in polynomial time. H.-I. Lu and R. Ravi [26] provided a 3-approximation that runs in almost linear time. P. S. Bonsma and F. Zickfeld [10] could show that the problem is $\frac{3}{2}$-approximable when the input is restricted to cubic graphs. Surprisingly, similar results were actually achieved also for the corresponding problem on directed graphs; we refer to N. Schwartges [31] and also to the conference paper [32] for a discussion of the related findings. Conversely, for MINIMUM CONNECTED DOMINATING SET, the best known approximation algorithms only reach an approximation factor of $H(\Delta(G))+2$, where $\Delta(G)$ is the maximum degree of graph G and H is the harmonic function; see [23]. That paper also proves that it is hard to improve the approximation guarantee of $H(\Delta)$ for any graph (reasoning about asymptotically large order and maximum degree), unless some inclusions between complexity classes hold that are considered to be unlikely.

running time	$\mathcal{O}^*(9.49^k)$	$\mathcal{O}^*(8.12^k)$	$\mathcal{O}^*(6.75^k)$	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(3.72^k)$	$\mathcal{O}^*(3.4575^k)$
klam value	20	22	24	33	35	37

Tab. 1: klam values for different record-claiming algorihms for producing leafy trees in undirected graphs.

Concerning parameterized algorithms for our problem, there is a sequence of papers culminating in the one of J. Kneis, A. Langer and P. Rossmanith [24]. This fairly simple branching algorithm achieves a running time of $\mathcal{O}^*(4^k)$. Prior to this, already several papers have been published on this parameterized problem, with running times of $\mathcal{O}^*(9.49^k)$ (by P. S. Bonsma *et al.* [9]), of $\mathcal{O}^*(8.12^k)$ (by V. Estivill-Castro *et al.* [16]) and of $\mathcal{O}^*(6.75^k)$ (by P. S. Bonsma and F. Zickfeld [11]). All these bounds have been obtained by using combinatorial arguments. The best kernelization result is due to [16], where the authors presented a kernel of (at most) $3.75k$ vertices. [(i)] Such seemingly minor improvements of the bases are of notable importance, as these will render inputs amenable to solutions that have been previously completely out of reach. For instance, M. Fellows *et al.* [18] raised the question if it is possible to develop an algorithm for our problem that runs in time $\mathcal{O}^*(f(k))$ for some function f such that $f(50) < 10^{20}$; the maximal value of k still allowing for $f(k) < 10^{20}$ is also known as the klam value of the according algorithm. It is instructive to compare the klam values of different algorithms for MAXIMUM LEAF SPANNING TREE, as compiled in Table 1. So, over the last decade, we are slowly approaching an affirmative answer of the question posed by M. Fellows *et al.* [18]. The problem to determine if there exists a connected dominating set in a given graph that is of size at most k', now k' being the parameter. is known to be W[2]-complete; see [15]. Hence, assuming that the parameterized complexity classes $\mathcal{FPT}$ and W[2] do not collapse

[(i)] The randomized algorithm by I. Koutis and R. Williams [25] claiming a running time of $\mathcal{O}^*(2^k)$ is believed to be flawed.

(which is considered unlikely, similar to the P versus NP question), there is no parameterized algorithm for this problem.

As the number of leaves of a maximum leaf spanning tree can be computed by a parameterized algorithm, the corresponding graph-theoretic number has been also employed in the development of parameterized algorithms for several other NP-hard optimization problems that may be solved in polynomial time for graphs of bounded maximum leaf number; see [17]. This area of research is called parameter ecology and also motivates the development of faster parameterized algorithms for our problem.

MAXIMUM LEAF SPANNING TREE is also a meaningful question for directed graphs, which can be also stated in the standard terminology of directed graphs [1]: find an out-branching with k leaves. Here an out-branching O in a directed graph G yields a spanning tree in the underlying undirected graph $UD(G)$ and O contains a unique vertex r (the root) that has in-degree zero (in O). Due to the uniqueness of r, the arcs are directed in O from the root to the leaves, which are the vertices of out-degree zero. The algorithm of J. Kneis, A. Langer and P. Rossmanith [24] solves also this problem in time $\mathcal{O}^*(4^k)$. Moreover, in J. Daligault *et al.* [13], an upper-bound of $\mathcal{O}^*(3.72^k)$ is shown. Hence, we are currently in the unusual situation that the running time for search tree algorithms for the directed case is no worse than the one obtained in the undirected case. By using rules that are specific for the undirected case, we are able to derive improved running times valid for the undirected case only.

Having shown a graph problem to be NP-complete, there is a third way of dealing with this problem, apart from developing (polynomial-time) approximation algorithms or parameterized algorithms (that are superpolynomial in the parameter). Namely, one could try to find algorithms that are moderately exponential in the number n of vertices of a graph (or even in the number of edges). This area of algorithmic research is also known as exact (moderately) exponential-time algorithms; see [22]. For vertex selection problems like MINIMUM CONNECTED DOMINATING SET, there is nearly always a rather trivial brute-force algorithm that tests all vertex subsets. Hence, the question is if it is possible to beat this trivial $\mathcal{O}^*(2^n)$ algorithm by replacing it with algorithms whose running time can be upper bounded by $\mathcal{O}^*(c^n)$ for some $c < 2$. For MINIMUM CONNECTED DOMINATING SET, the algorithm of F. V. Fomin, F. Grandoni and D. Kratsch [21] (yielding a running time of $\mathcal{O}^*(1.9407^n)$) was subsequently improved by a new one: H. Fernau *et al.* [19] gave an algorithm with running time $\mathcal{O}^*(1.8966^n)$ for undirected graphs. In passing, we will also improve on that result in this paper.

For a summary of the most recent results, we refer to Table 2.

1.1 Our Framework: Parameterized Complexity

A *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed alphabet and $\mathbb{N}$ is the set of all non-negative integers. Therefore, each instance of the parameterized problem P is a pair (I, k), where the second component k is called the *parameter*. The language $L(P)$ is the set of all YES-instances of P. We say that the parameterized problem P is *fixed-parameter tractable* [15] if there is an algorithm (also known as a *parameterized algorithm*) that decides whether an input (I, k) is a member of $L(P)$ in time $f(k)|I|^c$, where c is a fixed constant and $f(k)$ is a function independent of the overall input length $|I|$. We can also write $\mathcal{O}^*(f(k))$ for this running time bound. One can certify membership in the class $\mathcal{FPT}$ of fixed-parameter tractable problems as follows: present a polynomial-time transformation that, given an instance (I, k), produces another instance (I', k') of the same problem, where $|I'|$ and k' are bounded by some function $g(k)$; in this case, (I', k') is also called a *(problem) kernel*.

1.2 Our Contributions

We developed the simple and elegant algorithm of [24] further. This way, we can improve on the running time towards $\mathcal{O}^*(3.4575^k)$. This is due to two reasons: 1. We could improve the bottleneck case in the algorithm analysis of J. Daligault *et al.* [13] by new branching rules. 2. Due to using amortized analysis, we were able to prove a tighter upper-bound on the running time. For this analysis we use a non-standard measure, akin to the *Measure&Conquer*-approach in exact (non-parameterized) algorithmics; see F. V. Fomin and D. Kratsch [22]. Notice however that there are only few examples for using *Measure&Conquer* in parameterized algorithmics. In addition, we analyze our algorithm with respect to the number of vertices n and obtain also small improvements for the MINIMUM CONNECTED DOMINATING SET problem. This seems to be one the first published successful attempts to analyze *the same* algorithm both with respect to the standard parameter k and with respect to the number of vertices n, apart from [7]. We mention that the approaches of [13, 19] are going to some extent into the same direction. The basic scheme of the algorithms is similar. Nevertheless, our running time shows that our results are different. Moreover, the first paper [13] does not make use of *Measure&Conquer* techniques and the second one [19] follows a non-parameterized route. So, we conclude this paper by a moderately exponential-time re-analysis of our suggested algorithm along the lines of our parameterized analysis. This shows that there is an algorithm solving MAXIMUM LEAF SPANNING TREE in time $\mathcal{O}^*(3.4575^k)$ and in time $\mathcal{O}^*(1.89615^n)$.

MAXIMUM LEAF SPANNING TREE	parameterized algorithm	exact exponential-time
undirected graphs	$\mathcal{O}^*(3.4575^k)$ (*)	$\mathcal{O}^*(1.89615^n)$ (*)
directed graphs	$\mathcal{O}^*(3.72^k)$ [13]	$\mathcal{O}^*(1.9044^n)$ [6]

Tab. 2: Records for producing leafy trees

We summarize the various results known for maximum spanning tree problems in Table 2, including the given appropriate references; a star (*) indicates that the corresponding results are obtained in this paper.

1.3 Terminology

We are considering simple undirected graphs $G(V, E)$ with vertex set V and edge set $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We adhere to standard terminology, but we will explain most of it below for the sake of self-containment of this paper. Edges of simple undirected graphs can be viewed as two-element vertex sets, as "simple" prohibits loops and multi-edges. An edge $\{x, y\}$ might also be written as $x\,y$.

The *neighborhood* of a vertex $v \in V$ is $N_G(v) := \{u \mid \{u, v\} \in E\}$ and the *degree* of v is $d_G(v) = |N_G(v)|$. The *closed neighborhood* is $N_G[v] = N_G(v) \cup \{v\}$. For some $V' \subseteq V$, let $N(V') := (\bigcup_{v \in V'} N(v)) \setminus V'$, $N_{V'}(v) := \{u \in V' \mid \{u, v\} \in E\}$, $d_{V'}(v) = |N_{V'}(v)|$, and $E_{V'}(v) := \{\{u, v\} \in E \mid u \in V'\}$; when $V' = V$ or $E' = E$, we might suppress the subscript. $G[V']$ and $G[\tilde{E}]$ are the graphs induced by the vertex set V' and the edge set $\tilde{E}$, respectively. Hence, $G[V']$ is defined as $G(V', E')$ with $E' = \{xy \in E \mid x, y \in V'\}$, and $G[\tilde{E}]$ is equivalent to $G(\tilde{V}, \tilde{E})$ with $\tilde{V} = \{x \in V \mid \exists e \in \tilde{E} : x \in e\}$.

A sequence of vertices $p = v_1, \ldots, v_n$ is called a *path* from v_1 to v_n in $G(V, E)$ if, for all $1 \leq i < n$, $v_i v_{i+1} \in E$. The path p is a *cycle* if $v_1 = v_n$. The path p is *simple* if, for all $1 \leq i < j \leq n$, $v_i \neq v_j$. A graph $G(V, E)$ is *connected* if, for all $u, v \in V$, there is a path from u to v. An *edge cut-set* is a subset

$\hat{E} \subseteq E$ such that $G(V, E \setminus \hat{E})$ is not connected. A *tree* is a subset of edges $T \subseteq E$ such that $G[T]$ is connected and cycle-free. For simplicity, we occasionally identify $G[T]$ with T, so that we can write $d_T(v)$ to denote the degree of vertex v within the tree T. A *spanning tree* is a tree such that $\bigcup_{e \in T} e = V$. It is well-known that a graph is connected if and only if it admits a spanning tree. As connectedness can be decided in linear time, for the purpose of solving our problem, it is sufficient to consider graph instances that are connected.

We conclude with some terminology that is less standard, following the concepts introduced by Kneis *et al.* in [24]. A tree T' *extends* another tree T if $T \subseteq T'$. We will write $T' \succ T$ if T' extends T. An edge $e = xy \in E$ is called a *bridge* if there is no path from x to y in $G[E \setminus \{e\}]$. For any $E' \subseteq E$ let $leaves(E') := \{v \in V \mid d_{E'}(v) = 1\}$ and $internal(E') := \{v \in V \mid d_{E'}(v) \geq 2\}$. The following simple assertion is important for our strategy:

Lemma 1 *Let $G(V, E)$ be an undirected graph with at least three vertices and $k \geq 1$ be an integer. G admits a spanning tree with at least k leaves if and only if G is connected and there exists a spanning tree $T \subseteq E$ with at least k leaves and a vertex $r \in V$ with $d_T(r) \geq 2$.*

Proof: As G admits a spanning tree, G is connected. Let T be some spanning tree of G. As G contains at least three vertices, T must contain a non-leaf r. □

1.4 Overall Strategy

In the rest of the paper, we address the following annotated version of our problem:
ROOTED k-LEAF SPANNING TREE
Given: An undirected connected graph $G(V, E)$, a vertex $r \in V$ called *root*, and the parameter k.
We ask: Is there a spanning tree T for G with $|leaves(T)| \geq k$ with $d_T(r) \geq 2$?

According to Lemma 1, an algorithm solving this problem will also solve k-LEAF SPANNING TREE (with a polynomial overhead) by considering every $v \in V$ as the root, sorting out the trivial case when G has at most two vertices first.

All throughout the algorithm, we will maintain a tree $T \subseteq E$ whose vertices are $V_T := \bigcup_{e \in T} e$. Let $\overline{V}_T := V \setminus V_T$. T is always part of the solution. During the course of the algorithm, T will have two types of leaves: *leaf nodes (LN)* and *branching nodes (BN)*. The former ones will also appear as leaves in the solution. The latter ones can be leaves or internal vertices. Generally, we decide this by branching as far as reduction rules do not enforce exactly one possibility. *Internal nodes* (IN) are already determined to be non-leaves in T.

The algorithm will also produce a third kind of leaves: *floating leaves (FL)*. These are vertices from $\overline{V}_T$ which are already determined to be leaves, but are not yet attached to the tree T. If a vertex is neither a branching node, nor a leaf node, nor a floating leaf, nor an internal node, we call it *FREE*. We will refer to the different possible roles of a vertex by a labeling function $lab : V \to D$, with $D := \{\text{IN}, \text{FL}, \text{BN}, \text{LN}, \text{FREE}\}$. A given tree T' defines a labeling $V_{T'} \to D$ to which we refer by $lab_{T'}$. Let

- $\text{IN}_{T'} := \{v \in V_{T'} \mid d_{T'}(v) \geq 2\}$,
- $\text{LN}_{T'} := \{v \in V_{T'} \mid d_{\overline{V}_{T'}}(v) = 0, d(v) = 1\}$ and
- $\text{BN}_{T'} = V_{T'} \setminus (\text{IN}_{T'} \cup \text{LN}_{T'})$.

Then for any ID $\in D \setminus \{\text{FL}, \text{FREE}\}$, we have $\text{ID}_{T'} = lab_{T'}^{-1}(\text{ID})$. We always ensure that lab_T and lab are the same on V_T where T is the tree constructed by the algorithm. We suppress the subscript if we are refering to this tree. The subscript might be suppressed if $T' = T$. If $T' \succ T$, then we assume that $\text{IN}_T \subseteq \text{IN}_{T'}$ and $\text{LN}_T \subseteq \text{LN}_{T'}$. So, the labels IN and LN remain once they are fixed. For the other labels, we have the following possible label transitions in the course of the algorihm: FL $\rightarrow$ LN, BN $\rightarrow \{\text{LN}, \text{IN}\}$ and FREE $\rightarrow D \setminus \{\text{FREE}\}$.

Henceforth, we assume $|V| > 4$, as smaller instances can be easily resolved.

1.5 *Parameterized* Measure&Conquer

The number of leaves in the corresponding recursion tree provides a good estimate for the running time of branching algorithms that are designed to explore a large search space. This is based on the fact that there are at most twice as many nodes in a recursion tree of some computation as there are leaves in that tree, as nodes with exactly one child are merged with that child in this kind of reasoning. Assuming now a polynomial amount of work associated to each node of the recursion tree, the overall running time is dominated by the number of leaves of the recursion tree, in particular when using the $\mathcal{O}^*$-notation. Traditionally, to each node of the recursion tree, an (annotated) instance is associated, which would be in our case, a graph, a current parameter budget k, and vertex sets BN, LN, FL. Progress of the recursion would be measured by reducing the current parameter budget. To the root node of the recursion tree, the original instance would be associated; in particular, the parameter budget would be initialized with the parameter value associated to this instance. As can be seen by this type of explanations, the word "parameter" plays a sort of double role here. This is alleviated by a shift of perspective as provided by parameterized *Measure&Conquer*.

There, a so-called measure (also known as potential function) is instantiated with the parameter value associated to this instance. This measure will henceforth be used to keep track of the progress that the branching algorithm makes. The measure does not only capture the "current parameter", but also structural information obtained during the execution of the algorithm. In other words, this structural information does not necessarily refer to the case that some objects are fixed to be in the future solution. It can comprise much more, for instance, degree-one vertices, four-cycles, etc. Clearly, we have to show that the budget never increases on applying reduction rules and even decreases in case of recursive calls. But additionally, once our budget has been completely consumed, i.e., when the measure drops down to or below zero, we must be able to give an appropriate answer in polynomial time. As in general we accounted for more than only parts of a future solution, taking into account structural details of the graph, this might become a hard and tedious task. If we are able to fulfill all the mentioned conditions, we can prove a running time of the form $\mathcal{O}^*(c^k)$. Further examples of this approach are collected in the PhD thesis of one of the authors [2], and several publications using this approach have appeared [3, 4, 5, 20, 7].

Following usual conventions in this area, we will call the object named recursion tree in this subsection a *search tree* henceforth.

2 Reduction Rules & Observations

2.1 Reduction Rules

Most often, so-called reduction rules are the main building blocks of polynomial-time algorithms achieving a problem kernel. In our case, we are not so much interested in deriving a kernel, referring to [11]

for the currently best kernelization algorithm instead. Rather, our reduction rules are part of a search tree algorithm that we describe in details later. At this stage, it is important to keep in mind that the search tree will construct a (rooted) tree T within the graph to which we refer in the following rules. The leaf nodes of T that are not yet decided to become leaf nodes in the final spanning tree we are going to construct are collected into the set BN of branching nodes. Moreover, we have a possible empty set of floating leaves FL. Hence, when we speak about the correctness of the proposed reduction rules, we argue always in the sketched situation. So, more formally, as an instance can be now viewed as given by $I = (G(V, E), T, \text{BN}, \text{FL})$, a rule is sound if the following is true: for any such instance I, the instance $r(I) = (r(G(V, E)), r(T), r(\text{BN}), r(\text{FL}))$ produced by applying rule r to I (the effect of such an application is described below in each case) satisfies: I admits a spanning tree T' with $T' \succ T$, with at least k leaves if and only if $r(I)$ admits a spanning tree T'' with $T'' \succ r(T)$ with at least k leaves.

To see the overall correctness of the proposed algorithm and its running time analysis, it is also necessary that the measure never increases by any such operation. This will be shown in Lemma 10. One helpful observation for this (that can be seen already at this stage) is also the fact that neither our reduction rules nor our branching rules (stated below) will ever change the status of a vertex from an internal node to a leaf node (be it free or not) or vice versa.

We assume that reduction rule **(i)** is applied before **(i+1)**. The rules **(1)-(3)** also appeared in a previous paper coauthored by a superset of the present authors [19]. For the sake of completeness, we not only state but also prove the correctness of all reduction rules that we need in this paper.

(1) If there is an edge $e \in E \setminus T$ with $e \subseteq V_T$, then delete e.

(2) Every $u \in$ BN with $d(u) = 1$ becomes a leaf node and every $u \in$ FREE with $d(u) = 1$ becomes a floating leaf.

(3) If there is a vertex $u \in$ BN such that the removal of $E_{\overline{V}_T}(u)$ in $G[V \setminus \text{FL}]$ or $G[V]$ creates two components, then u becomes internal.

(4) If there are free vertices u, v such that there is a bridge $\{u, v\} \in E \setminus T$ in $G[V]$, where C_1, C_2 are the two components created by deleting $\{u, v\}$ and if $|V(C_1)| > 1$ and $|V(C_2)| > 1$, then contract $\{u, v\}$. The new vertex is also free.

(5) Delete $\{u, v\} \in E$ if u and v are floating leaves.

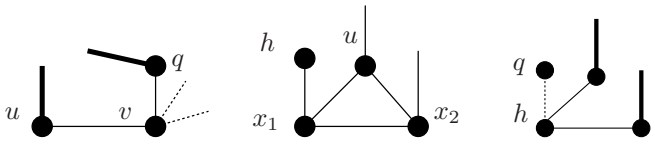
(6) Delete $\{u, v\} \in E \setminus T$ if $d_V(u) = 2$, $u \in$ BN and at least one of the following two cases apply: $a)$ $d_V(v) = 2$, or $b)$ $v \in$ FL.

(7) Delete $\{u, v\}$ if $u \in$ BN with $d_V(u) = 2$, $N_{\overline{V}_T}(u) = \{v\}$ and $d_{V_T}(v) \geq 2$; see Figure 1(a).

(8) If u, x_1, x_2 form a triangle, x_1 is free and $\{h\} = N_V(x_1) \setminus \{x_2, u\}$ such that $d_V(h) = 1$ (see Figure 1(b)) then x_1 becomes a floating leaf and h will be deleted.

(9) If $h \in \overline{V}_T$ is a free vertex such that $a)$ $N_{\overline{V}_T}(h) = \{q\}$ and $d(q) = 1$ or $b)$ $d_{\overline{V}_T}(h) = 0$ (see Figure 1(c)) then h becomes a floating leaf and q is deleted in case $a)$.

Lemma 2 *The reduction rules are sound.*



(a) RR**(7)** (b) RR**(8)** (c) RR**(9)**

Fig. 1: Bold edges are from T. Dotted edges may be present or not.

Proof: Let T' be a spanning tree with $T' \succ T$ such that $|leaves(T')| \geq k$.

(1) Any edge $e \in E \setminus T$ with $e \subseteq V_T$ added to T would introduce a cycle.

(2) In this case, u must be a leaf node in T' due to its degree constraint.

(3) If $E_{\overline{V}_T}(u)$ is an edge cut-set in $G[V]$, then u must be internal as we are looking for a spanning tree. Assume $E_{\overline{V}_T}(u)$ is an edge cut-set in $G[V \setminus \text{FL}]$ but not in $G[V]$. If $lab_{T'}(u) = \text{LN}$, i.e., u is a leaf node in T', then there must be a $z \in \text{FL}$ with $d_{T'}(z) \geq 2$. Thus, $z \in \text{IN}_{T'}$, a contradiction.

(4) Let G' be the graph which emerges by contracting $\{u, v\}$. We can assume that we have $d_{T'}(u) \geq 2$ and $d_{T'}(v) \geq 2$ due to connectivity. By contracting $\{u, v\}$ in T', we get a solution for G'. If G' has a spanning tree $T^* \succ T$ with k leaves, then the same is true for G; namely, the vertex obtained by identifying u and v cannot be a leaf node in T^*, as each of the components C_i contains at least two vertices and uv was a bridge in G; so, we can form an extension T'' from T^* with $|leaves(T'')| \geq k$ by making both u and v internal in T'' and keeping all other edges from T^*.

(5) If $\{u, v\}$ is part of a solution T' then $T' \cong K_2$. This contradicts $|V| > 2$.

(6.a) Consider $e := \{u, v\} \in E \setminus T$ with $u \in \text{BN}$ and $d_V(u) = d_V(v) = 2$. For the sake of contradiction, suppose that $e \in T'$ and that neither u nor v are leaves in T'. By removing e, u and v become leaves and T' is split into two components T'_1 and T'_2. As e is not a bridge and since T' is spanning, there is a $e' := \{a, b\} \in E \setminus (T \cup e)$ such that $T^* = T'_1 \cup T'_2 \cup \{e'\}$ is connected. Due to adjoining e', at most two leaves (in T') will become internal (in T^*). Hence, $|leaves(T^*)| \geq |leaves(T')|$. It is possible that $lab(a) = \text{FL}$, but if it was true that for every such edge e' one of its endpoints is a floating leaf, then we could have applied **(3)**, as $E_{\overline{V}_T}(v)$ would be an edge cut-set in $G[V \setminus \text{FL}]$. Since Rule **(3)** takes priority, $lab(a) = \text{FL}$ is excluded.

If $e := \{u, v\} \in T'$ and u or v is a leaf in T', then the proof of **(6.b)** applies, possibly exchanging the roles of u and of v.

(6.b) Suppose $e := \{u, v\} \in T'$. In this case, we know that $d_V(u) = 2$, $u \in \text{BN}$ and v is a leaf in T'. As e is not a bridge by **(3)** and **(4)**, there is an $e' = \{v, x\} \in E \setminus T'$ with $x \neq u$ and $x \notin \text{FL}$. Let $T^* := (T' \setminus \{e\}) \cup \{e'\}$. Note that $|leaves(T^*)| \geq |leaves(T')|$ as u is a leaf in T^* but not in T' and v is a leaf both in T' and in T^*.

(7) Let $q \in N_{V_T}(v) \setminus \{u\}$ and assume $e := \{u, v\} \in T'$. Then $e' := \{v, q\} \notin T'$ as $q \in V_T$.
Let $T^* = (T' \setminus \{e\}) \cup \{e'\}$. Then $|leaves(T^*)| \geq |leaves(T')|$ as $lab_{T^*}(u) =$ LN.

(8) Let G^* be the reduced graph. In T', x_1 must be internal. Observe that we can assume that $d_{T'}(x_1) = 2$ (❁). Otherwise, $\{u, x_1\}, \{x_1, x_2\}, \{x_1, h\} \in T'$, $\{u, x_2\} \notin T'$ and, w.l.o.g., u is internal as $|V| > 4$. Then simply delete $\{x_1, x_2\}$ from T' and adjoin $\{u, x_2\}$. This way we can ensure (❁). Due to (❁), we conclude that G^* has a spanning tree with k leaves iff G has one.

(9.a) Note that we have $d_{T'}(h) = 2$ since otherwise T' contains a cycle or is not connected. Let G^* be the reduced graph. Analogously as in **(8)**, we can show that G^* has a spanning tree with k leaves iff G has one.

(9.b) If $d_{T'}(h) > 1$, then T' would possess a cycle. □

Lemma 3 *Reduction rule* **(1)** *does not create any bridge in* $E \setminus T$.

Proof: Suppose an edge $e = \{u, v\} \notin T$ is deleted by **(1)** and a second edge $e' = \{x, y\} \in E \setminus T$ becomes a bridge in $G[E \setminus \{e\}]$. Then $G' := G[E \setminus \{e, e'\}]$ consists of two components G_1 and G_2 such that, w.l.o.g., $r, x \in V(G_1)$ and $y \in V(G_2)$. Thus, there is a simple path $P = rh_1 \dots h_\ell y$ in G such that $h_i \neq x, y$ $(1 \leq i \leq \ell)$ and there is a j, where $1 \leq j \leq \ell$, with, w.l.o.g., $h_j = u$ and $h_{j+1} = v$. As $u, v \in V_T$ there is a simple path P' in $G_1[T]$ from u to v such that $e, e' \notin E(P')$. Let $\hat{P} = rh_1 \dots h_{j-1} P' h_{j+2} \dots h_\ell y$. $\hat{P}$ is a path in G' which connects r and y avoiding e and e'. Thus, e' is not a bridge in $G[E \setminus \{e\}]$. □

From now on, we assume that G is reduced according to the given reduction rules in any further discussion of our branching algorithm.

2.2 Observations

If $N(internal(T)) \subseteq internal(T) \cup leaves(T)$, we call T an *inner-maximal* tree. This notion gave rise to the following crucial lemma in the paper of J. Kneis, A. Langer and P. Rossmanith [24]:

Lemma 4 ([24] Lemma 4.2) *If there is a tree T' with $leaves(T') \geq k$ such that $T' \succeq T$ and $x \in internal(T')$ then there is a tree T'' with $leaves(T'') \geq k$ such that $T'' \succeq T$, $x \in internal(T'')$ and $E_V(x) \subseteq T''$.*

Due to the previous lemma, we can restrict our attention to inner-maximal spanning trees. And in fact the forthcoming algorithm will only construct such trees. Then for a $v \in internal(T)$ we have that $E_V(v) \subseteq T$ as by Lemma 4 we can assume that T is inner-maximal. Thus, in the very beginning we have $T = E_V(r)$.

Lemma 5 *Let $v \in BN_T$ and $N_{\overline{V}_T}(v) = \{u\}$. If none of the reduction rules* **(1)-(9)** *applies, then u is free and $d_{\overline{V}_T}(u) \geq 2$.*

Proof: Note that $d_V(v) = 2$. Moreover, $u \notin$ IN $\cup$ BN $\cup$ FL due to T's inner-maximality, **(1)** and **(6.b)**. Thus, u is free. If $d_{\overline{V}_T}(u) = 0$ then **(9.b)** could be applied. If $d_{\overline{V}_T}(u) = 1$ then either we can apply **(3)** (if $\{u, v\}$ is a bridge) or **(6.a)** or **(7)** depending on whether $d_{V_T}(u) \geq 2$ or not. □

We are now going to define some function $co : \text{BN} \to V$. For $v \in \text{BN}$, let

$$co(v) = \begin{cases} v : d_{\overline{V}_T}(v) \geq 2, \\ u : N_{\overline{V}_T}(v) = \{u\}. \end{cases}$$

Note that co is well defined on a reduced instance as we have $d_{\overline{V}_T}(v) \geq 1$ (otherwise it becomes a leaf node **(2)**). Note that we have $d_{\overline{V}_T}(co(v)) \geq 2$ because either $d_{\overline{V}_T}(v) \geq 2$ or $d_{\overline{V}_T}(v) = 1$. In the latter case, $N_{\text{FREE}}(v) = N_{\overline{V}_T}(v) = \{u\}$, such that $d_{\overline{V}_T}(u) \geq 2$ by Lemma 5. This property will be used frequently.

The next lemma has been shown by [24] and is presented using the introduced definitions of this paper.

Lemma 6 ([24] Lemma 5) *Let $v \in BN_T$ such that $N_{\overline{V}_T}(v) = \{u\}$. If there is no spanning tree $T' \succ T$ with k leaves and $lab_{T'}(v) = LN$, then there is also no spanning tree $T'' \succ T$ with k leaves, $lab_{T''}(v) = IN$ and $lab_{T''}(u) = LN$.*

Observe that for a vertex v with $co(v) \neq v$ once we set $lab(v) = \text{IN}$ then it is also valid to set $lab(co(v)) = \text{IN}$. By Lemma 6 we must only ensure that we also consider the possibility $lab(v) = \text{LN}$.

A Further Reduction Rule With the assertion of Lemma 5 in mind, we state another reduction rule:

(10) If $w \in \text{BN}$ with $x_1 \in N_{\text{FREE}}(w)$ such that a free degree one vertex q is adjacent to x_1 and if further

$a)$ there exists $v \in \text{BN}$ with $N_{\overline{V}_T}(v) = \{x_1, x_2\}$ and $\{x_1, x_2\} \in E$, see Figure 2(b), or

$b)$ there exists $v \in \text{BN}$ with $co(v) \neq v$ and $N_{\overline{V}_T}(co(v)) = \{x_1, x_2\}$, see Figure 2(e),

then set $lab(v) = \text{LN}$.

Lemma 7 *Rule* **(10)** *is sound.*

Proof: We have to consider both cases of the rule separately.

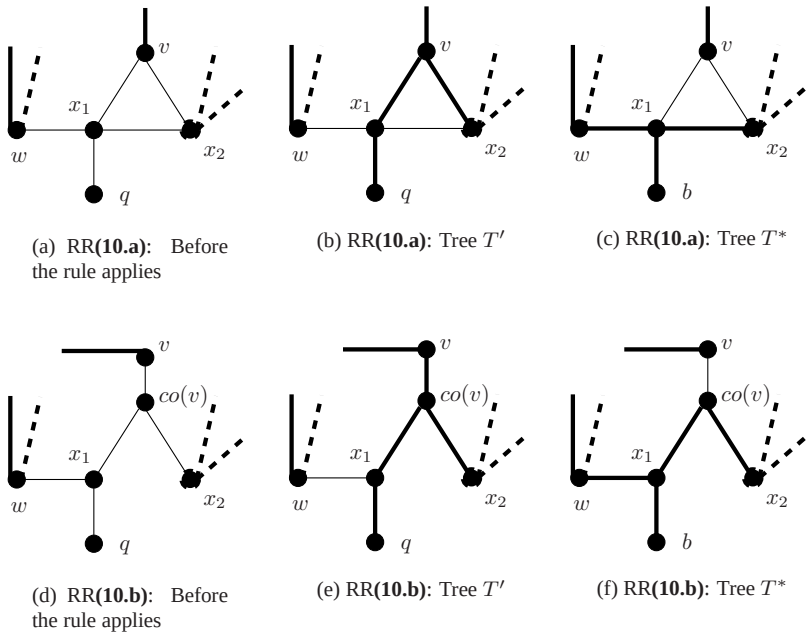
$a)$ Let $T' \succ T$ be a spanning tree with $lab_{T'}(v) = lab_{T'}(co(v)) = \text{IN}$. Consider $T^* := (T' \setminus \{x_1\, v, x_2\, v\}) \cup \{w\, x_1, x_1\, x_2\}$, see Figures 2(b) and 2(c). As $lab_{T'}(x_1) = lab_{T^*}(x_1) = \text{IN}$ and $lab_{T^*}(v) = \text{LN}$ we have $|leaves(T^*)| \geq |leaves(T')|$. Hence, we do not have to consider $lab(v) = lab(co(v)) = \text{IN}$.

$b)$ We can skip the possibility $lab(v) = lab(co(v)) = \text{IN}$. Assume the contrary. In that case consider $T^* := (T' \setminus \{v\, co(v)\}) \cup \{x_1\, w\}$, see Figures 2(e) and 2(f). Then we have $|leaves(T^*)| \geq |leaves(T')|$ as x_1 must be internal. Note that $lab_{T^*}(v) = \text{LN}$.

□

The next lemmas refer to the case where there is a $v \in \text{BN}_T$ with $d_{\overline{V}_T}(co(v)) = 2$. In the following, we use the abbreviation $\mathcal{N} := \{co(v), x_1, x_2\}$.

Lemma 8 *Let $T \subseteq E$ be a given tree such that $v \in BN_T$ and $N_{\overline{V}_T}(co(v)) = \{x_1, x_2\}$. Let T', T^* be optimal spanning trees under the restrictions $T' \succ T, T^* \succ T$, $lab_{T'}(v) = LN$, $lab_{T^*}(v) = lab_{T^*}(co(v)) = IN$ and $lab_{T^*}(x_1) = lab_{T^*}(x_2) = LN$.*



(a) RR**(10.a)**: Before the rule applies

(b) RR**(10.a)**: Tree T'

(c) RR**(10.a)**: Tree T^*

(d) RR**(10.b)**: Before the rule applies

(e) RR**(10.b)**: Tree T'

(f) RR**(10.b)**: Tree T^*

Fig. 2: Bold edges are from T. Dotted edges may be present or not.

1. *If there is a* $z \in ((N(x_1) \cap N(x_2)) \setminus \mathcal{N})$*, then* $|leaves(T')| \geq |leaves(T^*)|$.
2. *If* $co(v) = v$*,* $y \in N(x_2) \setminus \mathcal{N}$*,* $z \in N(x_1) \setminus \mathcal{N}$ *with* $lab_{T^*}(z) =$ *IN, then* $|leaves(T')| \geq |leaves(T^*)|$.
3. *If* $co(v) \neq v$ *and if there is a* $z \in ((N(x_1) \cup N(x_2)) \setminus \mathcal{N})$ *with* $lab_{T^*}(z) =$ *IN, then* $|leaves(T')| \geq |leaves(T^*)|$.

Proof:

1. Firstly, suppose $co(v) = v$. Consider $T^+ := (T^* \setminus \{v\,x_1, v\,x_2\}) \cup \{z\,x_1, z\,x_2\}$, see Figures 3(a) and 3(b). We have $lab_{T^+}(v) =$ LN and z can be the only vertex besides v where $lab_{T^+}(z) \neq lab_{T^*}(z)$. Thus, z could be the only vertex with $lab_{T^+}(z) =$ IN and $lab_{T^*}(z) =$ LN. Therefore, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$. Secondly, if $co(v) \neq v$ then consider $T^{\#} := (T^* \setminus \{v\,co(v), co(v)\,x_2\}) \cup \{z\,x_1, z\,x_2\}$ instead of T^+.
2. Consider $T^+ := (T^* \setminus \{v\,x_1, v\,x_2\}) \cup \{z\,x_1, y\,x_2\}$, see Figures 3(c) and 3(d). We have $lab_{T^+}(v) =$ LN and at most for y we could have $lab_{T^*}(y) =$ LN and $lab_{T^+}(y) =$ IN. Hence, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$.
3. W.l.o.g., $z \in N(x_1) \setminus \mathcal{N}$. Consider $T^{\#} := (T^* \setminus \{v\,co(v)\}) \cup \{z\,x_1\}$. We have $lab_{T^{\#}}(v) =$ LN and therefore $|leaves(T')| \geq |leaves(T^{\#})| \geq |leaves(T^*)|$.

□

Lemma 9 *Let $T \subseteq E$ be a given tree such that $v \in BN_T$ and $N_{\overline{V}_T}(co(v)) = \{x_1, x_2\}$. Let T', T^* be optimal spanning trees under the restrictions $T' \succ T$, $T^* \succ T$, $lab_{T'}(v) =$ LN, $lab_{T^*}(v) = lab_{T^*}(co(v)) =$ IN and $lab_{T^*}(x_1) =$ LN.*

1. *If $co(v) = v$, $\{x_1, x_2\} \in E$, $N(x_2) \setminus FL = \{v, x_1\}$ and if there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) =$ IN, then $|leaves(T')| \geq |leaves(T^*)|$.*

2. *If $co(v) \neq v$, $N(x_2) \setminus FL \subseteq \{co(v), x_1\}$ and if there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) =$ IN, then $|leaves(T')| \geq |leaves(T^*)|$.*

3. *If $lab_{T^*}(x_2) =$ IN, $d_{T^*}(x_2) = 2$, $E_{\overline{V}_T}(v)$ is not an edge cut-set in $G[V]$ nor $G[V \setminus FL]$. if in addition there is a $z \in N(x_1) \setminus \mathcal{N}$ with $lab_{T^*}(z) =$ IN, then $|leaves(T')| \geq |leaves(T^*)|$.*

Proof:

1. Consider $T^+ := (T^* \setminus \{v\,x_1, v\,x_2\}) \cup \{x_1\,x_2, z\,x_1\}$. Note that $lab_{T^+}(v) =$ LN and x_1 is the only vertex where we have $lab_{T^*}(x_1) =$ LN and $lab_{T^+}(x_1) =$ IN. Observe that T^+ is indeed a spanning tree. It is impossible that we have created a cycle, because x_1 is the only non-FL neighbor of x_2 in T^+. Thus, $|leaves(T')| \geq |leaves(T^+)| \geq |leaves(T^*)|$.

2. Consider $T^\# := (T^* \setminus \{v\,co(v)\}) \cup \{z\,x_1\}$ instead of T^+ from item 1.

3. First assume $co(v) = v$. Consider $T^+ := (T^* \setminus \{v\,x_1, v\,x_2\}) \cup \{z\,x_1\}$, see Figures 3(e) and 3(f). Due to Lemma 4 T^+ is a forest consisting of two trees T_1^+ and T_2^+, where v and x_2 have become leaf nodes. Thus $|leaves(T^+)| - 2 = |leaves(T^*)|$. As $E_{\overline{V}_T}(v)$ is not a edge cut-set there is some $e \in E \setminus (T^+ \cup E_V(v))$ such that $T^{++} := T^+ \cup \{e\}$ is connected. Furthermore, T^+ has at most two leaf nodes more than T^{++} as the addition of e might turn at most two leaf nodes into internal nodes. Thus as $lab_{T^{++}}(v) =$ LN, $|leaves(T')| \geq |leaves(T^{++})| \geq |leaves(T^*)|$. Note also that e can be choosen in a way that its addition will not turn any floating leaf of the current labeling of T^+ into an internal vertex as otherwise **(3)** applies.

   If $co(v) \neq v$ then consider $T^\# := (T^* \setminus \{v\,co(v), co(v)\,x_2\}) \cup \{z\,x_1\}$ instead of T^+. As $E_{\overline{V}_T}(v)$ is not an edge cut-set in $G[V]$, $\{v, co(v)\}$ is not a bridge. Thus, there is an $e \in E \setminus (T^\# \cup \{\{v, co(v)\}\})$ such that $|leaves(T^\# \cup \{e\})| \geq |leaves(T^*)|$ and $T^\# \cup \{e\}$ is a spanning tree where v is LN-node. Note also that for the same reasons as in the case $co(v) = v$, the edge e can be choosen such that any floating leaf is preserved.

□

In [24] the bottleneck case was when branching on a vertex $v \in BN$ with at most two non-tree neighbors, that is $d_{\overline{V}_T}(v) \leq 2$. In the last two lemmas we considered this case. If the bottleneck case also matches the conditions of Lemma 8 or 9 we either can skip some recursive call or decrease the yet to be defined measure by an extra amount. Otherwise we show that the branching behavior is more beneficial. This is a substantial ingredient for achieving a better running time upper bound.

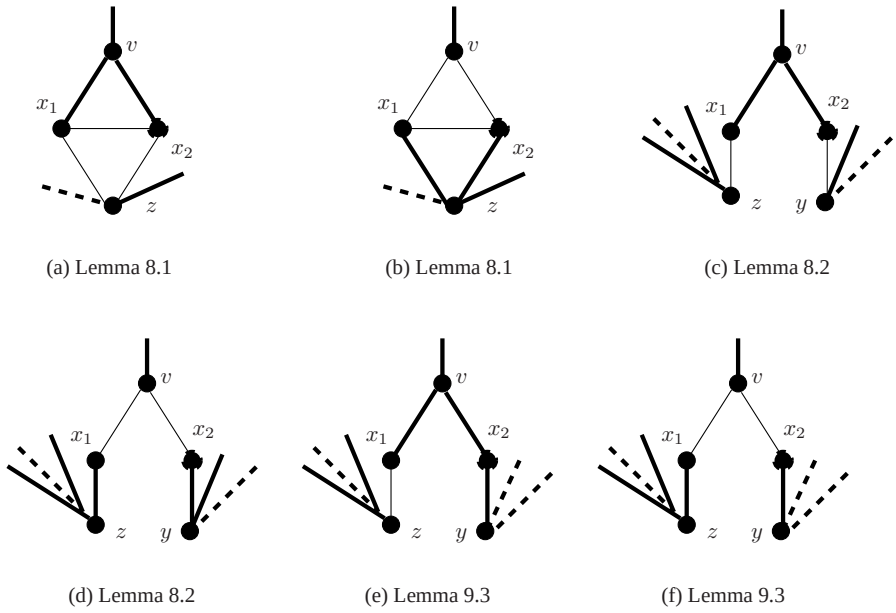
Algorithm 1: **An algorithm solving** k-LEAF SPANNING TREE

Data: A graph $G(V, E)$, $k \in \mathbb{N}$ and a tree $T \subseteq E$.
Result: YES if there is a spanning tree with at least k leaves and NO otherwise.

if $\kappa(G) \leq 0$ *or* $|BN| + |LN| \geq k$ **then**
  return YES
else if $G[V \setminus FL]$ *is not connected or* $BN = \emptyset$ **then**
  return NO
else
  Apply the reduction rules exhaustively
  Choose a vertex $v \in$ BN of maximum degree
  if $d_{\overline{V}_T}(co(v)) \geq 3$ **then**
    $\langle v \in \text{LN}; v, co(v) \in \text{IN}\rangle$ (B1)
  else
    (Note that now we must have $N_{\overline{V}_T}(co(v)) = \{x_1, x_2\}$)
    Choose v according to the following priorities:
    case $(\{x_1, x_2\} \subseteq FL)$ **or** *(B2.a)*
    $(x_1 \in FREE \;\&\; d_{\overline{V}_T \setminus \mathcal{N}}(x_1) = 0)$ **or** *(B2.b)*
    $(x_1 \in FREE \;\&\; N_{\overline{V}_T \setminus \mathcal{N}}(x_1) = \{z\} \;\&\; (d_{\overline{V}_T \setminus \mathcal{N}}(z) \leq 1$ *or* $z \in FL))$ *(B2.c)*
      $\langle v \in \text{LN}; v, co(v) \in \text{IN}\rangle$ (B2)
    case $x_1 \in FREE, x_2 \in FL$ *or* *(B3.a)*
    $x_1, x_2 \in FREE, N_{\overline{FL}}(x_2) \subseteq \{x_1, co(v)\}$ **or** *(B3.b)*
    $x_1, x_2 \in FREE \;\&\; d_{\overline{V}_T \setminus \mathcal{N}}(x_2) = 1$ *(B3.c)*
      $\langle v \in \text{LN}; v, co(v) \in \text{IN}, x_1 \in \text{LN}; v, co(v), x_1, co(x_1) \in \text{IN}\rangle$ (B3)
      and apply `makeleaves`(x_1, x_1) in the 2nd branch
    case $x_1, x_2 \in FREE \;\&\; \exists z \in \bigcap_{i=1,2} N_{\overline{FL} \setminus \mathcal{N}}(x_i)$
      $\langle v \in \text{LN}; v, co(v), x_2, co(x_2) \in \text{IN}, x_1 \in \text{LN}; v, co(v), x_1, co(x_1) \in \text{IN}\rangle$ (B4)
    otherwise
      $\langle v \in \text{LN}; v, co(v) \in \text{IN}, x_1, x_2 \in \text{LN}; v, co(v), x_2, co(x_2) \in \text{IN}, x_1 \in$ LN$; v, co(v), x_1, co(x_1) \in \text{IN}\rangle$ (B5)
      and apply `makeleaves`(x_1, x_2) in the 2nd branch

Procedure `makeleaves`(x_1, x_2)

begin
  $\forall u \in [(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}] \cap$ FREE set $u \in$ FL;
  $\forall u \in [(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}] \cap$ BN set $u \in$ LN;
end



(a) Lemma 8.1 (b) Lemma 8.1 (c) Lemma 8.2

(d) Lemma 8.2 (e) Lemma 9.3 (f) Lemma 9.3

Fig. 3: Bold edges are from T. Dotted edges may be present or not.

3 The Algorithm

We are now ready to present Algorithm 1.

We mention that if the answer YES is returned a k-leaf spanning tree can be constructed easily. This will be guaranteed by Lemma 10. For the sake of a short presentation of the different branchings, we introduce the following notation $\langle b_1; b_2; \ldots; b_n \rangle$, called a *branching*. Here the entries b_i are separated by semicolons and stand for the different *parts of the branching*. They will express how the label of some vertices change. For example: $\langle v \in \text{LN}; v, co(v) \in \text{IN} \rangle$. This stands for a binary branching where in the first part we set $lab(v) = \text{LN}$ and in the second $lab(v) = lab(co(v)) = \text{IN}$. Note that the non-standard run time measure κ will be defined later.

3.1 Correctness

3.1.1 Branchings

When we set $lab(v) = \text{IN}$, then we also set $T \leftarrow T \cup \{\{u, v\} \in E \mid u \notin V_T\}$. This is justified by Lemma 4. If we set $lab(v) = \text{LN}$, then we delete $\{\{u, v\} \in E \mid \{u, v\} \notin T\}$ as these edges will never appear in any solution.

In every branching of our algorithm, the possibility that $lab(v) = \text{LN}$ is considered. This recursive call must be possible, as otherwise **(3)** would have been triggered before. Now consider the case $co(v) \neq v$. If the recursive call for $lab(v) = \text{LN}$ does not succeed, then we consider $lab(v) = \text{IN}$. Due to Lemma 6 we immediately can also set $lab(co(v)) = \text{IN}$. This fact is used throughout the branchings (B1)-(B5).

Nonetheless, in the branchings (B1), (B2), (B3) and (B5) every possibility for v, x_1 and x_2 is considered in one part of the branching, i.e., these are exhaustive branchings. But in (B3) and (B5) the procedure `makeleaves()` is invoked and (B4) is not exhaustive. We will go through each subcase and argue that this is correct in a way that at least one optimal solution is preserved.

$B3.a)$ In the second part of the branch, every vertex in $(N(x_1) \cup N(x_2)) \setminus \{\mathcal{N}\}$ can be assumed to be a leaf node in the solution. Otherwise, due to Lemmas 8.2 and 8.3, a solution, which is no worse than the neglected one, can be found in the first part of the branch when we set $lab(v) = \text{LN}$. Therefore the application of `makeleaves(`x_1, x_1`)` is correct.

$B3.b)$ There is a $z_1 \in N_{\overline{\text{FL}}\setminus\mathcal{N}}(x_1)$ (by **(3)**). If $co(v) = v$ then we must have $\{x_1, x_2\} \in E$ due to **(3)**. Thus, either Lemma 9.1 or 9.2 applies (depending on whether $co(v) = v$ or not). As in the previous item it follows that the application of `makeleaves(`x_1, x_1`)` is correct.

$B3.c)$ Let $z_1 \in N_{\overline{\text{FL}}\setminus\mathcal{N}}(x_1)$ and $\tilde{T} \succ T$ be an optimal spanning tree solution extending T such that $lab_{\tilde{T}}(v) = \text{lab}_{\tilde{T}}(co(v)) = \text{IN}$ and $lab_{\tilde{T}}(x_1) = \text{LN}$. If $lab_{\tilde{T}}(x_2) = \text{LN}$, then Lemmas 8.2 and 8.3 apply. This means in the branch setting $v, co(v) \in \text{IN}, x_1 \in \text{LN}$, we can assume that vertices in $(N(x_1 \cup N(x_2))) \setminus \mathcal{N}$ are leaves, i.e., we can adjoin them to FL or LN. If $lab_{\tilde{T}}(x_2) = \text{IN}$, then we must have $d_{\tilde{T}}(x_2) = 2$. Thus, Lemma 9.3 applies. This can be seen as follows.
Let us recall the situation: $lab_{\hat{T}}(v) = lab_{\hat{T}}(co(v)) = \text{IN}$, $lab_{\hat{T}}(x_1) = \text{LN}$ and $lab_{\hat{T}}(x_2) = \text{IN}$. Thus, $d_{\hat{T}}(x_2) = 2$ due to the precondition of $B3.c)$. $E_{\overline{V}_T}(v)$ is not an edge cut-set in $G[V]$ nor in $G[V \setminus \text{FL}]$ due to **(3)**. Therefore, Lemma 9.3 applies here.
Thus, in the second part of the branch we can assume that every vertex in $N(x_1) \setminus \mathcal{N}$ is a leaf. Any solution which violates this assumption can be substituted by a no worse one where $v \in \text{LN}$ which is assured by Lemma 9.3. This justifies calling `makeleaves(`x_1, x_1`)`.

$(B4)$ does not consider the possibility that $lab(v) = \text{IN}$ and $lab(x_1) = lab(x_2) = \text{LN}$. Here we refer to Lemma 8.1, which states that a no worse solution can be found when we set $lab(v) = \text{LN}$.

$(B5)$ If $co(v) = v$, then by reduction rule **(3)** we have $N(x_i) \setminus \{v\} \neq \emptyset$. If $co(v) \neq v$ then it follows that $\bigcup_{i=1}^{2} N(x_i) \setminus \mathcal{N} \neq \emptyset$ analogously. Then due to Lemmas 8.2 and 8.3 an application of `makeleaves(`x_1, x_2`)` is valid. Note that Lemmas 8.2 and 8.3 can also be read with exchanged roles between x_1 and x_2.

3.1.2 The Measure

To derive an upper-bound on the running time for our algorithm, we use the measure

$$\kappa(G) := k - \omega_f \cdot |\text{FL}| - \omega_b \cdot |\text{BN}| - |\text{LN}| \text{ with } \omega_b = 0.5130 \text{ and } \omega_f = 0.4117.$$

$\kappa(G)$ is defined by a tree T and a labeling (which both are to be built up by our algorithm). Thus, we use a subscript when we are referring to this, i.e., $\kappa(G)_T$.

Parameterized Measure & Conquer poses two special problems:

- When our algorithm returns YES, then T might still not be a spanning tree. We first have to attach all the floating leaves to T. It is possible that a branching node turns into an internal node and

thus $\kappa(G)$ increases. The next important lemma shows that if we take all the floating leaves and branching nodes into account, then $\kappa(G)$ decreases.

- The application of reduction rules never increase the measure, at least in the case when this application is exhaustive.

Lemma 10 *If for a given labeling $\kappa(G) \leq 0$, then a spanning tree $\hat{T}$ with $|leaves(\hat{T})| \geq k$ can be constructed in polynomial time.*

Proof: Delete the vertices in FL_T and compute a depth-first spanning tree DT for the remaining graph starting from T. Then simply attach the vertices from FL_T to one of its neighbors in DT. This way we obtain a spanning tree $\hat{T} \succ T$. Let $\mathrm{LBN} = \mathrm{BN}_T \cap \mathrm{LN}_{\hat{T}}$ and $\mathrm{IBN} = \mathrm{BN}_T \setminus \mathrm{LBN}$. For $c \in \mathrm{IBN}$, let T_c be the subtree rooted at c in $\hat{T}$. Clearly, $|leaves(T_c)| \geq 1$ (✷). Observe that each vertex $v \in \mathrm{FL}_T \cup \mathrm{LBN}$ now decreases $\kappa(G)_{\hat{T}}$ by one due to the labeling under $\hat{T}$. Thus, $\kappa(G)_{\hat{T}}$ was decreased by $(1-\omega_f)$ or $(1-\omega_b)$, resp., with respect to $\kappa(G)_T$ due to turning v into a leaf node. The following chain of inequalities shows that $|leaves(\hat{T})| \geq k$.

$$\begin{aligned} k - |\mathrm{LN}_{\hat{T}}| = \kappa(G)_{\hat{T}} &\leq \kappa(G)_T - |\mathrm{LBN}| \cdot (1-\omega_b) + |\mathrm{IBN}| \cdot \omega_b - |\mathrm{FL}| \cdot (1-\omega_f) \\ &\leq \kappa(G)_T + |\mathrm{IBN}| \cdot \omega_b - \sum\nolimits_{c \in \mathrm{IBN}} |leaves(T_c)| \cdot (1-\omega_f) \\ &\leq \kappa(G)_T + |\mathrm{IBN}| \cdot (\omega_b + \omega_f - 1) \leq \kappa(G)_T \leq 0. \qquad \text{(by (✷))} \end{aligned}$$

□

Next we consider the interaction of the reduction rules with the measure.

Lemma 11 *An exhaustive application of the reduction rules never increases $\kappa(G)$.*

Proof: Rule **(2)** decreases $\kappa(G)$ by $1-\omega_b$ or ω_f. The deletion of an edge from $E \setminus T$ can cause a change in $\kappa(G)$ only in one way: a branching node can become a leaf node. Then $\kappa(G)$ is decreased by $1-\omega_b$. If one of **(1)**, **(5)**, **(6)** and **(7)** applies, then exactly such an edge is deleted. If **(10)** applies, then a vertex $v \in \mathrm{BN}$ becomes a leaf node. Thus, one or more edges are deleted and therefore $\kappa(G)$ is not increasing. In **(3)** there is a vertex $u \in \mathrm{BN}$ which becomes internal. Thus, $\kappa(G)$ increases by ω_b. Moreover, there is a second vertex $q \in N_{\overline{V}_T}(u)$. If q is free, then it becomes a branching node and $\kappa(G)$ decreases by ω_b. Thus, $\kappa(G)$ does not change if we sum up both amounts. If q is a floating leaf, then it becomes a leaf node. Thus $\kappa(G)$ decreases by $1-\omega_f-\omega_b > 0$. As in **(4)** an edge with with two free vertices is contracted such that the resulting vertex is also free, $\kappa(G)$ remains the same. In **(8)** a free vertex becomes a floating leaf and a vertex h with degree one is deleted. As we have $h \in \mathrm{FL}$, $\kappa(G)$ does not change. Rule **(9.a)** can be analyzed analogously. In **(9.b)** a free vertex turns in a floating leaf and $\kappa(G)$ decreases by ω_f. □

3.2 Running Time Analysis

In our algorithm the changes of $\kappa(G)$ are caused by triggering reduction rules or by branching. In the first case Lemma 11 ensures that $\kappa(G)$ will never increase. In the second case we have reductions of $\kappa(G)$ of the following type. When a vertex $v \in \mathrm{BN}$ becomes a leaf node (i.e., we set $lab(v) = \mathrm{LN}$) then $\kappa(G)$ will drop by an amount of $(1-\omega_b)$. On the other hand when v becomes an internal node (i.e., we

set $lab(v) = \text{IN}$) then $\kappa(G)$ will increase by ω_b. This is due to v not becoming a leaf. Moreover, the free neighbors of v become branching nodes and the floating leaves become leaf nodes, due to Lemma 4. Therefore $\kappa(G)$ will be decreased by ω_b and $1 - \omega_f$, respectively. We point out that the weights ω_b and ω_f have to be chosen such that $\kappa(G)$ will not increase in any part of a branching of our algorithm.

Analyzing the Different Branching Cases
(B1) Let $i := |N_{\overline{V}_T}(co(v)) \cap \text{FL}|$ and $j := |N_{\overline{V}_T}(co(v)) \cap \text{FREE}|$. Note that $i + j \geq 3$. Then the branching vector is: $(1 - \omega_b, i \cdot (1 - \omega_f) + j \cdot \omega_b - \omega_b)$.

(B2) a) The branching vector is $(1 - \omega_b, 2 \cdot (1 - \omega_f) - \omega_b)$.

b) When we set $lab(v) = lab(co(v)) = \text{IN}$ the vertex x_1 will become a leaf node due to **(1)**. The branching vector is $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$. The vertex x_2 contributes an amount of $\min\{1 - \omega_f, \omega_b\}$ depending on whether $x_2 \in \text{FL}$ or $x_2 \in \text{BN}$.

c) Firstly, suppose that $z \in FL$.

1. $d(z) = 1$:
   (a) $co(v) = v$: If $\{x_1, x_2\} \notin E$ then either **(9.a)** or **(3)** applies (depending on whether $d_{V_T}(x_1) > 1$). If $\{x_1, x_2\} \in E$, then there is some $z_1 \in N_{V_T}(x_1) \setminus \{v\}$, as otherwise **(8)** applies. But then **(10.a)** applies.
   (b) $co(v) \neq v$: If $d_{V_T}(x_1) = 0$, then either **(8)** or **(4)** applies (depending on whether $\{x_1, x_2\} \in E$). If $d_{V_T}(x_1) > 0$, then **(10.b)** applies.
2. $d(z) \geq 2$: After setting $lab(v) = lab(co(v)) = \text{IN}$ and applying **(1)** exhaustively we have $d(x_1) = 2$ and $x_1, x_2 \in \text{BN}$ afterwards. Observe that adding an edge to T does not create a bridge. The same holds for rule **(1)** (Lemma 3). Thus, rules **(3)** or **(4)** are not triggered before the rules with lower priority. As **(2)** and **(5)** do not change the local setting with respect to x_1 and z **(6.b)** will delete $\{x_1, z\}$, leading to a $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$ branch.

The case that $z \in \text{FREE}$ can be seen by similar arguments. Observe that $d_G(z) \geq 2$ by **(2)**. In the part where we set $v, co(v) \in \text{IN}$ we obtain a tree $T' := T \cup E_{\overline{V}_T}(co(v))$. Then rule **(1)** will delete edges incident to x_1 such that $d_V(x_1) = 2$ and $x_1 \in BN_{T'}$.
1. If $d_{T'}(z) \geq 2$, then **(7)** deletes $\{x_1, z\}$.
2. If $d_{T'}(z) = 1$, then by $d_G(z) \geq 2$ and $d_{\overline{V}_T \setminus \mathcal{N}}(z) \leq 1$ we deduce that $d_G(z) = 2$ before setting $v, co(v) \in \text{IN}$. Note that $x_2 \notin N(x)$ as $x_2 \in V_{T'}$. Hence, **(6.a)** deletes $\{x_1, z\}$ afterwards.
Hence we have a $(1 - \omega_b, 1 + \min\{1 - \omega_f, \omega_b\} - \omega_b)$ branch. We point out that it is guaranteed that no reduction rule triggered after **(1)** of lower priority than **(6)** and **(7)** will change the local situation with respect to x_1 and z (note Lemma 3).

Remark 12 *From this point on, w.l.o.g., for a free vertex x_i, $i = 1, 2$, we have:*

1. $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) \geq 2$ *or*
2. $N_{\overline{V}_T \setminus \mathcal{N}}(x_i) = \{z_i\}$ *such that* $d_{\overline{V}_T \setminus \mathcal{N}}(z_i) \geq 2$ *and* $z_i \notin FL$.

If $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) = 0$, then (B2.b) would apply. If $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) = 1$ and 2. fails then case (B2.c) applied.

Remark 13 *Note that if* 2. *applies to* x_i*, then when we set* $lab(v)) = lab(co(v)) =$ *IN, we have that* $co(x_i) = z_i$ *after the application of the reduction rules. In this sense (with a slight abuse of notation) we set* $co(x_i) = x_i$ *if case* 1. *applies and* $co(x_i) = z_i$ *if case* 2. *applies.*

(B3) If x_i *is free let* $fl_i := |(N(co(x_i)) \setminus \mathcal{N}) \cap FL|$ *and* $fr_i := |(N(co(x_i)) \setminus \mathcal{N}) \cap FREE|$.

Due to Remark 12 we have $fl_i + fr_i \geq 2$.

a) Note that we must have that $S := N_{\overline{\text{FL}}}(x_1) \setminus \mathcal{N} \neq \emptyset$ due to **(3)**. If $co(v) = v$, then also $N(x_i) \setminus \{v\} \neq \emptyset$ $(i = 1, 2)$ due to **(3)**. Hence, in the second branch for every $q \in S$ we get ω_f if $lab(q) =$ FREE as we set $lab(q) =$ FL in `makeleaves(`x_1, x_1`)`. If $lab(q) =$ BN, we set $lab(q) =$ LN in `makeleaves(`x_1, x_1`)` and receive $1 - \omega_b$. We have the following reduction in $\kappa(G)$ for the different parts of the branching.

$v \in$ LN: $1 - \omega_b$.

$v \in$ IN, $co(v) \in$ IN, $x_1 \in$ LN: $1 + \min\{\omega_f, 1 - \omega_b\} + 1 - \omega_f - \omega_b$

$v, co(v), x_1, co(x_1) \in$ IN: $1 - \omega_f + fl_1(1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b$.

Remark 14 *Note that from this point we have that* x_1 *and* x_2 *are free.*

b) Note that we must have that $S := N_{\overline{\text{FL}}}(x_1) \setminus \mathcal{N} \neq \emptyset$ due to **(3)**. Analogously as in *a)* we obtain $\min\{\omega_f, 1 - \omega_b\}$ in addition from $N(x_1) \setminus \mathcal{N}$ in the second part of the branch. Thus, we have the branching vector

$$(1 - \omega_b, 1 + \min\{\omega_f, 1 - \omega_b\} + \omega_b - \omega_b, \omega_b + fl_1(1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b). (\diamondsuit)$$

Remark 15 *Observe that from now on there is always a vertex* $z_i \in N_{\overline{FL} \setminus \mathcal{N}}(x_i)$ *(*$i = 1, 2$*) due to the previous case.*

c) This entails the same branch as in ($\diamondsuit$).

(B4) Due to Lemma 8.1 a

$(1 - \omega_b, 1 + fr_2 \cdot \omega_b + fl_2 \cdot (1 - \omega_f) - \omega_b, \omega_b + fr_1 \cdot \omega_b + fl_1 \cdot (1 - \omega_f) - \omega_b)$ branch

can be derived.

(B5) In this case $\bigcap_{i=1,2} N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_i) = \emptyset$ is true, as otherwise (B4) applies. This means for $i = 1, 2$ there are two different vertices $z_i \in N_{\overline{\text{FL}} \setminus \mathcal{N}}(x_i)$ (Remark 15). Due to (B3.c) we have $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) \geq 2$. Thus, in the second part we additional get $(fr_1 + fr_2) \cdot \omega_f$. If $fr_i = 0$ we get an amount of $1 - \omega_b$ by Remark 15.

$v \in$ LN: $1 - \omega_b$.

$v, co(v) \in$ IN, $x_1, x_2 \in$ LN: $2 + (fr_1 + fr_2) \cdot \omega_f + (\max\{0, 1 - fr_1\} + \max\{0, 1 - fr_2\}) \cdot (1 - \omega_b) - \omega_b$

$v, co(v), x_2, co(x_2) \in$ IN, $x_1 \in$ LN: $1 + fl_2 \cdot (1 - \omega_f) + fr_2 \cdot \omega_b - \omega_b$

$v, co(v), x_1, co(x_1) \in$ IN: $\omega_b + fl_1 \cdot (1 - \omega_f) + fr_1 \cdot \omega_b - \omega_b$.

We have calculated the branching number for every mentioned recursion such that $2 \leq fr_i + fl_1 \leq 5$, $i = 1, 2$, with respect to ω_b and ω_f. The branching number of any other recursion is upper-bounded by one of these. We mention the bottleneck cases which attain the given running time:
Cases *(B3.b)/(B3.c)* and Case *(B5)* such that $d_{\overline{V}_T \setminus \mathcal{N}}(x_i) = 2$ and *a)* $fl_1 = fl_2 = 0$, $fr_1 = fr_2 = 2$, *b)* $fr_1 = 2$, $fl_1 = 0$, $fr_2 = fl_2 = 1$; compared to [13] case *(B5)* has been improved. We can find at least two vertices which are turned from FREE vertices to floating leaves or from branching nodes to leaf nodes. In [13] only one vertex with this property can be found in the worst case. Thus, due to the previous case analysis and the fact that the reduction rules can be executed in polynomial time, we can state our main result:

Theorem 16 k-LEAF SPANNING TREE *can be solved in time* $\mathcal{O}^*(3.4575^k)$*, using polynomial space.*

4 An Exact Exponential Time Analysis

We stated the running time of Alg 1 in terms of k where k is the number of leaves in the spanning tree.

Exponential Time Analysis F. V. Fomin, F. Grandoni and D. Kratsch [21] gave an exact exponential-time algorithm with a running time of $\mathcal{O}^*(1.9407^n)$. Based on and re-analyzing the parameterized algorithm of Kneis, Langer and Rossmanith [24], this was recently improved to $\mathcal{O}^*(1.8966^n)$, see [19]. We can show further (slight) improvements by re-analyzing our parameterized algorithm. Therefore, we define a new measure:

$$\tau := n - \tilde{\omega}_f \cdot |\text{FL}| - \tilde{\omega}_b \cdot |\text{BN}| - |\text{LN}| - |\text{IN}| \text{ with } \tilde{\omega}_b = 0.2726 \text{ and } \tilde{\omega}_f = 0.5571.$$

The remaining task is quite easy. We only have to adjust the branching vectors we derived with respect to $\kappa(G)$ to τ. This leads to Table 3.

Note that in cases *(B3) b)* and *(B3) c)*, we were making a case distinction based on $co(x_1)$. If $co(x_1) \neq x_1$ then in the branch where we set $v, co(v), x_1, co(x_1) \in$ IN we can decrease τ by one more as $co(x_1)$ can be counted additionally. If $co(x_1) = x_1$ then $fr_1 \cdot \tilde{\omega}_f + \max\{0, 1 - fr1\} \cdot (1 - \tilde{\omega}_b)$ is the least amount by which τ is reduced due to the application of `makeleaves(`x_1, x_1`)`.
It can be checked that every branching number of the above recursions is upper bounded by $\mathcal{O}^*(1.89615^\tau)$.

Theorem 17 MAXIMUM LEAF SPANNING TREE *can be solved in* $\mathcal{O}^*(1.89615^n)$*, using polynomial space.*

5 Conclusions

Parameterized Measure & Conquer Amortized search tree analysis, also known as *Measure & Conquer*, is a big issue in exact, non-parameterized algorithmics. Although search trees play an important role in exact parameterized algorithmics, this kind of analysis has been rather seldom applicable. Good examples are the papers of Fernau and Raible [20], which deals with MAXIMUM ACYCLIC SUBGRAPH, the analysis of 3-HITTING-SET in Wahlström's PhD Thesis [35] and the amortized analysis of cubic vertex cover by Chen, Kanj and Xia [12]. This paper contributes to this topic. Let us emphasize the difference to the, say, non-parameterized *Measure & Conquer* and to this case. Usually if a measure μ, which is used to derive an upper bound of the form $c^{|V|}$, is decreased to zero then we immediately have a solution.

(B1)	:	$(1-\tilde{\omega}_b, i\cdot(1-\tilde{\omega}_f)+j\cdot\tilde{\omega}_b+1-\tilde{\omega}_b)$.
(B2) a)	:	$(1-\tilde{\omega}_b, 2\cdot(1-\tilde{\omega}_f)+1-\tilde{\omega}_b)$.
(B2) b)/c)	:	$(1-\tilde{\omega}_b, 1+\min\{1-\tilde{\omega}_f, \tilde{\omega}_b\}+1-\tilde{\omega}_b)$.
(B3) a)	:	$(1-\tilde{\omega}_b, 1+\min\{\tilde{\omega}_f, 1-\tilde{\omega}_b\}+1-\tilde{\omega}_f+1-\tilde{\omega}_b,$ $fl_1\cdot(1-\tilde{\omega}_f)+fr_1\cdot\tilde{\omega}_b+1-\tilde{\omega}_b+1-\tilde{\omega}_f+1)$.
(B3) b)/c) $co(x_1)\neq x_1$	:	$(1-\tilde{\omega}_b, 1+\tilde{\omega}_f+\tilde{\omega}_b+1-\tilde{\omega}_b,$ $\tilde{\omega}_b+fl_1\cdot(1-\tilde{\omega}_f)+fr_1\tilde{\omega}_b+1-\tilde{\omega}_b+2)$.
(B3) b)/c) $co(x_1)=x_1$	:	$(1-\tilde{\omega}_b, 1+fr_1\cdot\tilde{\omega}_f+\max\{0,1-fr1\}\cdot(1-\tilde{\omega}_b)+\tilde{\omega}_b+1-\tilde{\omega}_b,$ $\tilde{\omega}_b+fl_1\cdot(1-\tilde{\omega}_f)+fr_1\cdot\tilde{\omega}_b+1-\tilde{\omega}_b+1)$.
(B4)	:	$(1-\tilde{\omega}_b, 1+fr_2\cdot\tilde{\omega}_b+fl_2\cdot(1-\tilde{\omega}_f)+1-\tilde{\omega}_b+1,$ $\tilde{\omega}_b+fl_1\cdot(1-\tilde{\omega}_f)+fr_1\cdot\tilde{\omega}_b+1+1-\tilde{\omega}_b$.
(B5)	:	$(1-\tilde{\omega}_b, 2+1-\tilde{\omega}_b+(fr_1+fr_2)\cdot\tilde{\omega}_f+$ $(\max\{0,1-fr_1\}+\max\{0,1-fr_2\})\cdot(1-\tilde{\omega}_b),$ $2+1-\tilde{\omega}_b+fl_2\cdot(1-\tilde{\omega}_f)+fr_2\cdot\tilde{\omega}_b,$ $\tilde{\omega}_b+1-\tilde{\omega}_b+1+fl_1\cdot(1-\tilde{\omega}_f)+fr_1\cdot\tilde{\omega}_b)$

Tab. 3: The modified branching vectors in the different cases of our analysis.

Almost all time this is quite clear because then the instance is polynomial-time solvable. Now if the parameterized measure $\kappa(G)$ is smaller than zero then in general a hard sub-instance remains. Also $\kappa(G)$ has been decreased due to producing floating leaves, which are not attached to the tree yet. Thus, it is crucial to have Lemma 10, which ensures that a k-leaf spanning tree can be indeed constructed. Beyond that, it is harder to show that no reduction rules ever increase $\kappa(G)$. As vertices which have been counted already partly (e.g., because they belong to FL $\cup$ BN) can be deleted, $\kappa(G)$ can even increase temporally. Concerning the traditional approach this is a straight-forward task and is hardly ever mentioned. It is still a challenge to find further parameterized problems where this, say, parameterized Measure & Conquer paradigm can be applied.

As there is a linear kernel [16] we can first kernelize the input graph in polynomial time. The kernel size is no more than $3.75k$. Thus, we can run our algorithm on the kernel, which yields a running time upper-bound of $\mathcal{O}(3.4575^k + poly(n))$. Notice that the right choice for ω_f and ω_b is quite crucial. For example, setting $\omega_f = \omega_b = 0.5$ only shows a running time upper-bound of $\mathcal{O}^*(3.57^k)$. To find these beneficial values, a local search procedure was executed on a computer. It is worth pointing out that our algorithm is quite explicit. This means that its statement is to some extent lengthy but on the other hand easier to implement. The algorithm does not use compact mathematical expressions which might lead to ambiguities in the implementation process.

Final Remarks A preliminary version of this paper appeared in [29]. We like to thank the referees for their scholarly work.

References

[1] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2009.

[2] D. Binkele-Raible. *Amortized Analysis of Exponential Time- and Parameterized Algorithms: Measure & Conquer and Reference Search Trees*. PhD thesis, Fachbereich IV, Universität Trier, Germany, 2010.

[3] D. Binkele-Raible, L. Brankovic, M. Cygan, H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, M. Pilipczuk, P. Rossmanith, and J. O. Wojtaszczyk. Breaking the 2^n-barrier for IRREDUNDANCE: Two lines of attack. *Journal of Discrete Algorithms*, 9:214–230, 2011.

[4] D. Binkele-Raible and H. Fernau. A new upper bound for Max-2-SAT: A graph-theoretic approach. *Journal of Discrete Algorithms*, 8:388–401, 2010.

[5] D. Binkele-Raible and H. Fernau. An exact exponential time algorithm for POWER DOMINATING SET. *Algorithmica*, 63:323–346, 2012.

[6] D. Binkele-Raible and H. Fernau. An exact exponential-time algorithm for the directed maximum leaf spanning tree problem. *Journal of Discrete Algorithms*, 15:43–55, 2012.

[7] D. Binkele-Raible, H. Fernau, S. Gaspers, and M. Liedloff. Exact and parameterized algorithms for MAX INTERNAL SPANNING TREE. *Algorithmica*, 65:95–128, 2013.

[8] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and MANETs. In *Handbook of Combinatorial Optimization*, volume B, pages 329–369. Springer, 2005.

[9] P. S. Bonsma, T. Brueggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In B. Rovan and P. Vojtáś, editors, *Mathematical Foundations of Computer Science 2003, MFCS*, volume 2747 of *LNCS*, pages 259–268. Springer, 2003.

[10] P. S. Bonsma and F. Zickfeld. A $3/2$-approximation algorithm for finding spanning trees with many leaves in cubic graphs. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008*, volume 5344 of *LNCS*, pages 66–77. Springer, 2008.

[11] P. S. Bonsma and F. Zickfeld. Spanning trees with many leaves in graphs without diamonds and blossoms. In E. Sany Laber, C. F. Bornstein, L. Tito Nogueira, and L. Faria, editors, *LATIN 2008: Theoretical Informatics*, volume 4957 of *LNCS*, pages 531–543. Springer, 2008.

[12] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. *Algorithmica*, 43:245–273, 2005.

[13] J. Daligault, G. Gutin, E. J. Kim, and A. Yeo. FPT algorithms and kernels for the directed k-leaf problem. *Journal of Computer and System Sciences*, 76(2):144–152, 2010.

[14] R. J. Douglas. NP-completeness and degree restricted spanning trees. *Discrete Mathematics*, 105(1-3):41–47, 1992.

[15] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[16] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In H. Broersma, M. Johnson, and S. Szeider, editors, *Algorithms and Complexity in Durham ACiD 2005*, volume 4 of *Texts in Algorithmics*, pages 1–41. King's College Publications, 2005.

[17] M. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. Rosamond, and S. Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory of Computing Systems*, 45:822–848, 2009.

[18] M. R. Fellows, C. McCartin, F. A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: an improved FPT algorithm for Max Leaf Spanning Tree and other problems. In S. Kapoor and S. Prasad, editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS*, volume 1974 of *LNCS*, pages 240–251. Springer, 2000.

[19] H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, D. Raible, and P. Rossmanith. An exact algorithm for the Maximum Leaf Spanning Tree problem. *Theoretical Computer Science*, 412(45):6290–6302, 2011.

[20] H. Fernau and D. Raible. Exact algorithms for maximum acyclic subgraph on a superclass of cubic graphs. In S.-I. Nakano and Md. S. Rahman, editors, *Workshop on Algorithms and Computation: WALCOM*, volume 4921 of *LNCS*, pages 144–156. Springer, 2008.

[21] F. V. Fomin, F. Grandoni, and D. Kratsch. Solving Connected Dominating Set faster than 2^n. *Algorithmica*, 52:153–166, 2008.

[22] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

[23] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, 1998.

[24] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. *Algorithmica*, 61:882–897, 2011.

[25] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Proceedings, Part I*, volume 5555 of *LNCS*, pages 653–664. Springer, 2009.

[26] H.-I Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms*, 29:132–141, 1998.

[27] T. Milenković, V. Memišević, A. Bonato, and N. Pržulj. Dominating biological networks. *PLOSone*, 6(8), 2011.

[28] M. Rai, S. Verma, and S. Tapaswi. A power aware minimum connected dominating set for wireless sensor networks. *Journal of Networks*, 4(6), 2009.

[29] D. Raible and H. Fernau. An amortized search tree analysis for k-LEAF SPANNING TREE. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, editors, *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *LNCS*, pages 672–684. Springer, 2010.

[30] E. Sampathkumar and H. B. Walikar. The connected domination number of a graph. *Journal of Mathematical and Physical Sciences*, 13(6):607–613, 1979.

[31] N. Schwartges. Approximationsalgorithmen für das gerichtete Maximum-Leaf-Spanning-Tree-Problem. Master's thesis, Universität Würzburg, Germany, 2011.

[32] N. Schwartges, J. Spoerhase, and A. Wolff. Approximation algorithms for the maximum leaf spanning tree problem on acyclic digraphs. In R. Solis-Oba and G. Persiano, editors, *Approximation and Online Algorithms — 9th International Workshop, WAOA 2011*, volume 7164 of *LNCS*, pages 77–88. Springer, 2012.

[33] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms—ESA '98, 6th Annual European Symposium*, volume 1461 of *LNCS*, pages 441–452. Springer, 1998.

[34] M. T. Thai, F. Wang, D. Liu, S. Zhu, and D.-Z. Du. Connected dominating sets in wireless networks different transmission ranges. *IEEE Transactions on Mobile Computing*, 6:721–730, 2007.

[35] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007.