An exact algorithm for the Maximum Leaf Spanning Tree problem[☆]

Henning Fernau[a], Joachim Kneis[b,1], Dieter Kratsch[c,2], Alexander Langer[b,1], Mathieu Liedloff[d,2,*], Daniel Raible[a], Peter Rossmanith[b,1]

[a] *Universität Trier, FB 4—Abteilung Informatik, D-54286 Trier, Germany*
[b] *Department of Computer Science, RWTH Aachen University, Germany*
[c] *Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France*
[d] *Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, 45067 Orléans Cedex 2, France*

Abstract

Given an undirected graph with n vertices, the Maximum Leaf Spanning Tree problem is to find a spanning tree with as many leaves as possible. When parameterized in the number of leaves k, this problem can be solved in time $O(4^k \text{poly}(n))$ using a simple branching algorithm introduced by a subset of the authors [16]. Daligault, Gutin, Kim, and Yeo [6] improved the branching and obtained a running time of $O(3.72^k \text{poly}(n))$. In this paper, we study the problem from an exponential time viewpoint, where it is equivalent to the Connected Dominating Set problem. Here, Fomin, Grandoni, and Kratsch showed how to break the $\Omega(2^n)$ barrier and proposed an $O(1.9407^n)$-time algorithm [11]. Based on some useful properties of [16] and [6], we present a branching algorithm whose running time of $O(1.8966^n)$ has been analyzed using the Measure-and-Conquer technique. Finally we provide a lower bound of $\Omega(1.4422^n)$ for the worst case running time of our algorithm.

Keywords: Exponential time algorithms, graphs algorithms, spanning tree, maximum leaf spanning tree

[☆] An extended abstract of this paper was presented at the conference "International Workshop on Parameterized and Exact Computation (IWPEC 2009)", Copenhagen, Denmark [10].

[*] Corresponding author.

Email addresses: fernau@uni-trier.de (Henning Fernau), kneis@cs.rwth-aachen.de (Joachim Kneis), kratsch@univ-metz.fr (Dieter Kratsch), langer@cs.rwth-aachen.de (Alexander Langer), mathieu.liedloff@univ-orleans.fr (Mathieu Liedloff), raible@informatik.uni-trier.de (Daniel Raible), rossmani@cs.rwth-aachen.de (Peter Rossmanith)

[1] Partially supported by the DFG under grant RO 927/7.

[2] Partially supported by ANR Blanc AGAPE (ANR-09-BLAN-0159-03).

1. Introduction

The Maximum Leaf Spanning Tree (MLST) problem, which asks to find for a given graph a spanning tree with as many leaves as possible, is one of the classical NP-complete problems [14]. Ongoing research on this topic is motivated by the fact that variants of this problem occur frequently in real life applications. For example, some broadcasting problems in network design ask to minimize the number of broadcasting nodes, which must be connected to a single root. This translates nicely to finding a spanning tree with many leaves and few internal nodes.

In the sense of exact algorithms the Maximum Leaf Spanning Tree problem is equivalent to the Connected Dominating Set problem, where one should find a minimum set of vertices $C \subseteq V$ of the input graph G such that the subgraph G induced by C is connected and C is a dominating set of G. It is easy to see that the internal nodes of a spanning tree with k leaves are a connected dominating set of size $|V| - k$ and vice versa. Connected Dominating Set is a fundamental problem in connected facility location and studied intensively in computer science and operations research [15, 19] and it is a also central problem in wireless networking, see e.g. [5, 17, 20].

Known results. In the field of exact exponential time algorithms, there is only the paper by Fomin, Grandoni, and Kratsch [11] in which they present a branching algorithm and use the Measure & Conquer analysis technique to establish a running time of $O(1.9407^n)$, where n is the number of vertices of the input graph. This result was the first improvement over the $\Omega(2^n)$ barrier achieved by trivial enumeration. By way of contrast, there is a long research history for this problem in the field of parameterized complexity, see [1, 7, 9, 3, 8, 2, 4] (in chronological order). The currently fastest algorithm builds and improves on the one by Kneis, Langer, and Rossmanith [16] with a runtime bounded by $O(4^k \text{poly}(n))$: in [6], Daligault, Gutin, Kim, and Yeo achieved a run time of $O(3.72^k \text{poly}(n))$. These ideas for improvements are also used in our exact algorithm.

Our results. In the next sections we present an exact algorithm solving the MLST problem in time $O(1.8966^n)$, thereby considerably improving upon the algorithm of [11]. Our algorithm is based on the parameterized ones presented in [6, 16], which basically repeatedly branches on leaves of a subtree of the graph in order to decide whether it can remain a leaf or must become an internal node of the spanning tree. If we analyze the running time as a function of n, we find that branching on nodes of small degree (with two possible successors) becomes the worst case resulting in a bad running time. This resembles the worst case of the parameterized algorithm, and the changes in [6] are based on improving exactly this case. We use a similar approach for our exact algorithm. We mark nodes as leaves as early as possible even when they are not yet attached to an internal node. To analyse our algorithm, we use the so-called Measure-and-Conquer technique which aim to balance in the analysis the bad cases against the

better cases, i.e., the better cases "lend" some running time to the bad cases for an overall improvement. This approach requires a rather complicated measure and an involved analysis of the various cases that can occur in recursive calls of the algorithm. As usual for the Measure-and-Conquer technique, the measure has then been optimized by a computer program. We verified the measure by an independent computer program that simulates the recursive calls of the algorithm.

2. Preliminaries

Let $G = (V, E)$ be a simple, undirected graph. We denote by n the number of its vertices and by m the number of its edges. Given a vertex $v \in V$, the set of its neighbors is defined by $N(v) = \{ u \in V \mid \{u, v\} \in E \}$. The closed neighborhood of v is $N[v] = \{v\} \cup N(v)$. Given a subset $S \subseteq V$, we define $N(S)$ as the set $\bigcup_{v \in S} N(v) \setminus S$ and for a $X \subseteq V$, we define $N_X(S) = N(S) \cap X$. We write $H \subseteq G$ if H is a subgraph of G.

Let $T = (V_T, E_T)$ be a tree and V_T its set of nodes. A tree $T = (V_T, E_T)$ is a *subtree* of G (or a *tree in* G) if $T \subseteq G$. The subtree T of G is a *spanning tree* of G if $V_T = V$. As usual, a node of degree 1 in a tree T is called a *leaf* and all other nodes are called *internal* nodes. Assuming that the input graph G has $n \geq 3$ vertices, each spanning tree of G contains at least one internal node. Once we fix some arbitrary node as the root and an internal node of the tree, we can also speak of *parents* and *children* of nodes within this tree. A spanning tree is a maximum leaf spanning tree (MLST) if there is no spanning tree with a larger number of leaves.

In the following, it is helpful to identify a tree $T = (V_T, E_T)$ with the bipartition of V_T into the sets of internal nodes and the set of leaves, denoted by internal(T) and leaves(T), respectively. Although there might be multiple subtrees of G sharing the same bipartition into the set of internal nodes and the set of leaves, either both are subtrees of some optimal solution of MLST or none of them is.

Branching Algorithms, Search Trees and Measure-and-Conquer. The design and analysis of our algorithm heavily relies on the concepts of branching algorithm, branching vector, branching number, and the Measure-and-Conquer analysis of branching algorithms. For convenience, we provide a short summary on these concepts and their use in our work. For detailed information we refer to [13] which contains a chapter on branching algorithms and one on Measure-and-Conquer, and also to [12, 18].

The algorithm presented in this paper is a *branching algorithm* and thus a recursive algorithm. Such an algorithm searches for a solution by calling and solving subproblems respectively branches recursively. We say that it branches into subproblems, or it k-branches into k subproblems. If the algorithm is called recursively for one subproblem then this is called a reduction. In this way we speak of reduction rules and branching rules. For example, the main idea of our algorithm is to branch into two subproblems: for a chosen particular vertex of

the input graph it is an internal node of the spanning tree in one branch and it is a leaf of the spanning tree in the other branch.

The execution of a branching algorithm on some input is typically illustrated by a *search tree* which is essentially the tree of the recursive calls of this execution. Every subproblem is assigned to a node of the search tree, and the original problem with the input is assigned to the root of the search tree. Finally when branching on a problem assigned to node x into $k \geq 2$ subproblems we assign each subproblem or branch to a child of x.

Typically, and this is also the case in our algorithm, the running time on every node or every subproblem is polynomial. This requires in particular that the time for a sequence of reductions is bounded by a polynomial. Under this assumption, which is satisfied for our algorithm, the running time of the branching algorithm on some input graph G is bounded by a polynomial times the number of leaves in the search tree obtained when executing the algorithm on G. Let $T(n)$ be the maximum number of leaves of the search tree for any execution of the algorithm on an input of length n. To estimate the running time we need to upper bound the function $T(n)$.

Let us describe how to analyse the running time for our graph algorithm $\mathcal{M}$. The instances of our subproblems are annotated graphs (see Section 3). We choose a *measure* μ that assings to every possible instance a real number. This measure is used to analyse the running time of the algorithm, and its choice is crucial for the analysis. The goal is to prove that the number of leaves of the search tree is at most $\alpha^{\mu(G)}$ for any input graph G, where $\alpha = 1.8966$ for our algorithm $\mathcal{M}$.

To do this we need to consider all branching rules. Assume that the algorithm on input $\mathcal{I}$ (an annotated graph) branches into k subinstances $\mathcal{I}_1, \ldots, \mathcal{I}_k$. This implies $T(\mu(\mathcal{I})) \leq T(\mu(\mathcal{I}_1)) + \cdots + T(\mu(\mathcal{I}_k))$. Let $\Delta_i = \mu(\mathcal{I}) - \mu(\mathcal{I}_i) > 0$ for $1 \leq i \leq k$ be the decrease of the measure. The tuple $(\Delta_1, \ldots, \Delta_k)$ is called a *branching vector* and the unique positive real solution of the equality $1 = z^{-\Delta_1} + \cdots + z^{-\Delta_k}$ is called its *branching number*. It now suffices to show that the branching number is at most $\alpha = 1.8966$, or in other words, $1 \geq \alpha^{-\Delta_1} + \cdots + \alpha^{-\Delta_k}$ $(*)$. This type of analysis has to be undertaken for all branching rules, recurrences and branching vectors obtained.

By a careful case analysis and the use of computer programs, we generated all recurrences and branching vectors such as to minimize the number of leaves in the search tree via the choice of the measure. In this way, we established that the maximum number of leaves in a search tree for an input graph on n vertices is 1.8966^n. Having achieved our choice of the measure and the corresponding value $\alpha = 1.8966$, it is easy to verify that they fulfill all requirements. Furthermore, one may present then the results in the manner of an inductive proof over the measure, as it is done in this paper. For example, by the induction hypothesis for the $\mathcal{I}_i$ and using $(*)$, we get that

$$T(\mu(\mathcal{I})) \leq T(\mu(\mathcal{I}_1)) + \cdots T(\mu(\mathcal{I}_k)) \leq 1.8966^{\mu(\mathcal{I}_1)} + \cdots + 1.8966^{\mu(\mathcal{I}_k)} \leq 1.8966^{\mu(\mathcal{I})}.$$

Finally, since $\mu(\mathcal{I}) \leq n$ we can upper bound the running time on a graph with

n vertices by $O(1.8966^n \text{poly}(n))$ where $\text{poly}(n)$ is a polynomial. In Section 5, we present the details of our running time analysis.

3. A New Exact Algorithm

In this section, we introduce an exact algorithm to solve the following annotated version of the MLST problem. Let $G = (V, E)$ be the input graph to the MLST problem and consider a partition $V = \text{Free} \cup \text{FL} \cup \text{BN} \cup \text{LN} \cup \text{IN}$ of the vertex set into the sets of *free vertices* (Free), *floating leaves* (FL), *branching nodes* (BN), *leaf nodes* (LN), and *internal nodes* (IN). For input $G, \text{IN}, \text{BN}, \text{LN}, \text{FL}$ the annotated MLST problem is to find a maximum leaf spanning tree that respects the annotations $\text{IN}, \text{BN}, \text{LN}, \text{FL}$, i.e., the vertices in IN are internal nodes of the spanning tree and the vertices in $\text{LN} \cup \text{FL}$ are leaves.

Definition 1. Let $G = (V, E)$ be a graph, and let $\text{IN}, \text{BN}, \text{LN}, \text{FL} \subseteq V$ be disjoint sets of vertices and $T \subseteq G$ be a tree. We say T *extends* $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ iff $\text{IN} \subseteq \text{internal}(T)$, $\text{LN} \subseteq \text{leaves}(T)$, $\text{BN} \subseteq \text{internal}(T) \cup \text{leaves}(T)$, and $\text{FL} \cap \text{internal}(T) = \emptyset$.

The key idea of the algorithm is to recursively build a subtree $T = (V_T, E_T) \subseteq G$ with $V_T = \text{IN} \cup \text{BN} \cup \text{LN}$, $\text{internal}(T) = \text{IN}$ and $\text{leaves}(T) = \text{BN} \cup \text{LN}$, which might in some branch of the search tree eventually turn into a spanning tree T' of G that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$. Vertices in LN will always remain leaves in subsequent calls. The branching nodes in BN are leaves of the current subtree T, but might be promoted to internal nodes or leaves in recursive calls of the algorithm. Vertices in $\text{FL} \subseteq V \setminus V_T$ are fixed to be leaves, but they are still "floating around". This means that they have not yet been attached to T, and therefore their parent node in the solution is unknown at the moment. See Figure 1 for an example.

The recursive construction of the spanning tree exploits the following crucial observation used in [16]. Roughly speaking, if there are $v \in \text{BN}$ and an optimal spanning tree T with $v \in \text{internal}(T)$, then there also is a solution T' where the $\text{Free} \cup \text{FL}$-neighbors of v in G are children of v in T'.

Lemma 1 ([16]). *Let $G = (V, E)$ be a graph and let $\text{IN}, \text{BN}, \text{LN}, \text{FL} \subseteq V$ be disjoint sets of vertices. Let $T \subseteq G$ be a spanning tree of G that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ and has k leaves. If there is $v \in \text{BN}$ with $v \in \text{internal}(T)$, then there also is a k-leaf spanning tree T' of G that extends $\big(\text{IN} \cup \{v\}, (\text{BN} \setminus \{v\}) \cup N_{\text{Free}}(v) \cup N_{\text{FL}}(v), \text{LN}, \text{FL} \setminus N_{\text{FL}}(v)\big)$.*

The lemma can be proved by a simple exchange argument for the edges connecting the neighbors of v in T and T'. Thus, when Algorithm $\mathcal{M}$ branches such that some vertex $x \in \text{BN}$ becomes an internal node, then all of its neighbors will instantly be attached to the tree such that each one has the node x as a neighbor in the tree.

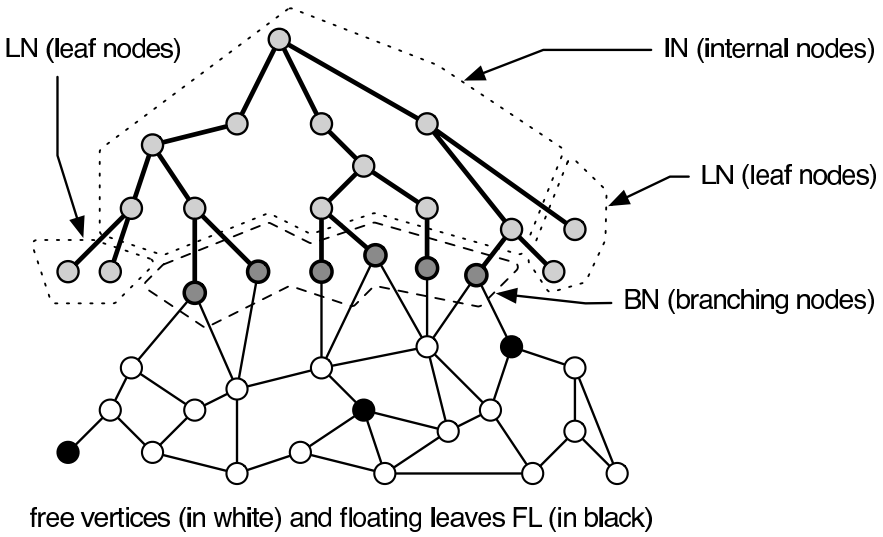

Figure 1: An example of a graph with a subtree with corresponding sets of vertices IN, BN, LN(describing the subtree), as well as FL and Free.

If $T \subseteq G$ is a tree such that $N(\text{internal}(T)) \subseteq \text{internal}(T) \cup \text{leaves}(T)$ (where $N(\text{internal}(T))$ denote the neighborhood of $\text{internal}(T)$ in G), we call T *inner-maximal*. Note that by the considerations above the algorithm maintains inner-maximal trees.

To avoid cluttered notation, we introduce the following notation to denote changes to the annotations $\text{IN}, \text{BN}, \text{LN}, \text{FL}$ and the recursive branches of the algorithm.

Definition 2. Let $G = (V, E)$ be a graph, let $\text{IN}, \text{BN}, \text{LN}, \text{FL} \subseteq V$ be disjoint sets of vertices, and let $x_1, \ldots, x_l \in V$. By $x_1 \to X_1, \ldots, x_l \to X_l$, where each X_i is one of IN, BN, LN or FL, we denote the operation of moving each x_i to the respective set X_i, and additionally, if $X_i = \text{IN}$, of moving all $y \in N_{\text{Free}}(x_i)$ to BN and all $y \in N_{\text{FL}}(x_i)$ to LN. The notation is extended to $Y \to X$, where $Y \subseteq V$, in a straightforward manner. To solve an instance, our algorithm considers subinstances obtained from $G, \text{IN}, \text{BN}, \text{LN}, \text{FL}$ by applying a set of operations of the form $x \to X$. We write

$$\langle x_{1,1} \to X_{1,1}, \ldots, x_{1,l_1} \to X_{1,l_1} \mid\mid \ldots \mid\mid x_{k,1} \to X_{k,1} \ldots, x_{k,l_k} \to X_{k,l_k} \rangle,$$

to express that the algorithm branches into k subinstances, where in the jth call, $1 \leq j \leq k$, the algorithm considers the subinstance obtained from $G, \text{IN}, \text{BN}, \text{LN}, \text{FL}$ by applying the operations $x_{j,1} \to X_{j,1}$ to $x_{j,l_j} \to X_{j,l_j}$.

For any $v \in V \setminus (\text{IN} \cup \text{LN})$, we define its *degree* $d(v)$ as $d(v) = |N(v) \cap (\text{Free} \cup \text{FL})|$ if $v \in \text{BN}$, as $d(v) = |N(v) \cap (\text{Free} \cup \text{FL} \cup \text{BN})|$ if $v \in \text{Free}$, and as $d(v) = |N(v) \cap (\text{Free} \cup \text{BN})|$ if $v \in \text{FL}$.

A vertex $v \in \text{Free} \cup \text{FL}$ is *unreachable*, if there is no path $uv_1 \dots v_t v$, where $t \geq 0$, $u \in \text{BN}$ and $v_i \in \text{Free}$ for all $1 \leq i \leq t$. We note that if $\text{BN} = \emptyset$, then every vertex in $\text{FL} \cup \text{Free}$ is unreachable.

Our algorithm uses the following reduction rules.

Definition 3. Let $G = (V, E)$ be a graph and let $\text{IN} \cup \text{BN} \cup \text{LN} \cup \text{FL} \cup \text{Free}$ be a partition of V. We define the following reduction rules:

(R1) If there exist two adjacent vertices $u, v \in V$ such that $u, v \in \text{FL}$ or $u, v \in \text{BN}$, then remove the edge $\{u, v\}$ from G.

(R2) If there exists a node $v \in \text{BN}$ with $d(v) = 0$, then move v into LN.

(R3) If there exists a free vertex v with $d(v) = 1$, then move v into FL.

(R4) If there exists a free vertex v with no neighbors in $\text{Free} \cup \text{FL}$, then move v into FL.

(R5) If there exists a triangle $\{x, y, z\}$ in G with x a free vertex and $d(x) = 2$, then move x into FL.

(R6) If there exists a node $u \in BN$ which is a cut vertex in G, then apply rule $u \to \text{IN}$.

(R7) If there exist two adjacent vertices $u, v \in V$ such that $u \in \text{LN}$ and $v \in V \setminus \text{IN}$, then remove the edge $\{u, v\}$ from G.

The correctness of the reduction rules is easy to prove. This is detailed in the following lemma.

Lemma 2. *Let $G = (V, E)$ be a graph and let $\text{IN}, \text{BN}, \text{LN}, \text{FL} \subseteq V$ be disjoint sets of vertices. Let T be a tree with a maximum number of leaves that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$. Assume that a reduction rule, say (Ri), $i = 1, 2, \dots 7$ is applied to the instance $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ from G. Then any tree T' with maximum number of leaves that extends the instance obtained from $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ by applying Reduction Rule (Ri) has the same number of leaves as T.*

Proof. Suppose that T is a subtree of G corresponding to $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, and that T' is any spanning tree of G such that T' extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$.

(R1) Let $u, v \in \text{FL}$. Then both are leaves of T' and cannot be adjacent in T' thus we may remove the edge $\{u, v\}$ from G. If $u, v \in \text{BN}$ then they cannot be adjacent in T', otherwise there would be a cycle in T' consisting of a path from u to v in T and the edge $\{u, v\}$. Consequently the edge $\{u, v\}$ can safely be removed from G.

(R2) Let $v \in \text{BN}$ and $d(v) = 0$. Then there is no free vertex adjacent to v. Hence no vertex can be a child of v in T' and thus v is a leaf of T'. Hence, we can safely put v into LN.

(R3) Let v be a free vertex with $d(v) = 1$. Then v has degree 1 in T', and thus v is a leaf of T'. Hence we can safely put v into FL.

(R4) Let v be a free vertex with no neighbors in Free $\cup$ FL. Hence v can only be adjacent to vertices, more precisely one vertex, of BN in T'. Thus v is a leaf of T'. Hence we can safely put v into FL.

(R5) Let $\{x, y, z\}$ be a triangle in G such that x is a free vertex and $d(x) = 2$. Assume that x is an internal node of T' extending $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$. Hence, x is adjacent to y and z in T', and thus one of them is the parent of x in T', say y. Then, the spanning tree T'' of G obtained from T' by removing $\{x, y\}$ and adding $\{y, z\}$ has never fewer leaves than T'. Thus, without loss of generality, x is a leaf in any spanning tree of G which extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ and has as many leaves as possible. Consequently, we can safely put x into FL.

(R6) Let $u \in BN$ be a cut vertex of G. Then the unique path from the root to the branching node u in the tree T passes through one component of $G-u$. Hence, the vertex u needs to be an internal node of T'. Consequently, we can safely put u into IN. Notice that the free neighbors of u are then put into BN and the floating leaves neighbors into LN.

(R7) Let $u, v \in V$ be adjacent vertices in G such that $u \in \text{LN}$ and $v \in V \setminus \text{IN}$. Since u is a leaf of T and T', u is only adjacent to one internal node x of T', which is the parent in a subtree corresponding to $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, i.e., $x \in \text{IN}$. Hence, it is not adjacent to any $v \in V \setminus \text{IN}$ in T'. Therefore, edge $\{u, v\}$ can be safely removed from G. □

Note that only for Rule (R5) we need to require that T' is a spanning tree with maximum number of leaves. For all other reduction rules it suffices to require that T' is a spanning tree.

We call an instance $(G, \text{IN}, \text{BN}, \text{LN}, \text{FL})$ a *reduced instance* if no reduction rule can be applied to it. Note that the only reduction rule changing the graph G is Rule (R1). In an instance to which Rule (R1) cannot be applied, and in particular in a reduced instance a simpler definition of degree can be used: $d(v) = |N(v) \cap (\text{Free} \cup \text{FL} \cup \text{BN})|$ for all $v \in (\text{BN} \cup \text{Free} \cup \text{FL})$.

The halting conditions and the branching rules are described in Algorithm $\mathcal{M}$ (see Figure 2). Their correctness is shown in the following Section. The running time analysis is provided in Section 5.

4. Correctness of the Algorithm

Throughout this section, we suppose that the given instance is reduced, which means that none of the Reduction Rules (R1) to (R7) can be applied to the instance.

The following lemma will ease the forthcoming correctness proof. It enables us to turn some vertices into additional floating leaves in some special cases. A similar technique has been already used in [6].

Algorithm $\mathcal{M}$
Input: A graph $G = (V, E)$, $\mathrm{IN}, \mathrm{BN}, \mathrm{LN}, \mathrm{FL} \subseteq V$

Reduce G according to the reduction rules.
if there is some unreachable $v \in \mathrm{Free} \cup \mathrm{FL}$ **then** return 0
if $V = \mathrm{IN} \cup \mathrm{LN}$ **then** return $|\mathrm{LN}|$
Choose a vertex $v \in \mathrm{BN}$ of maximum degree.
if $d(v) \geq 3$ **or** ($d(v) = 2$ and $N_{\mathrm{FL}}(v) \neq \emptyset$) **then**
    $\langle v \to \mathrm{LN} \mid\mid v \to \mathrm{IN} \rangle$ (B1)
else if $d(v) = 2$ **then**
    Let $\{x_1, x_2\} = N_{\mathrm{Free}}(v)$ such that $d(x_1) \leq d(x_2)$.
    if $d(x_1) = 2$ **then**
        Let $\{z\} = N(x_1) \setminus \{v\}$
        if $z \in \mathrm{Free}$ **then**
            $\langle v \to \mathrm{LN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{IN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{LN} \rangle$ (B2)
        else if $z \in \mathrm{FL}$ **then** $\langle v \to \mathrm{IN} \rangle$
    else if $(N(x_1) \cap N(x_2)) \setminus \mathrm{FL} = \{v\}$ **and** $\forall z \in (N_{\mathrm{FL}}(x_1) \cap N_{\mathrm{FL}}(x_2))$, $d(z) \geq 3$ **then** (B3)
        $\langle v \to \mathrm{LN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{IN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{LN}, x_2 \to \mathrm{IN} \mid\mid$
        $v \to \mathrm{IN}, x_1 \to \mathrm{LN}, x_2 \to \mathrm{LN},$
        $N_{\mathrm{Free}}(\{x_1, x_2\}) \to \mathrm{FL}, N_{\mathrm{BN}}(\{x_1, x_2\}) \setminus \{v\} \to \mathrm{LN} \rangle$
    else $\langle v \to \mathrm{LN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{IN} \mid\mid v \to \mathrm{IN}, x_1 \to \mathrm{LN}, x_2 \to \mathrm{IN} \rangle$ (B4)
else if $d(v) = 1$ **then**
    Let $P = (v = v_0, v_1, \ldots, v_k)$ be a maximum path such that
        $d(v_i) = 2$, $1 \leq i \leq k$, $v_1, \ldots, v_k \in \mathrm{Free}$.
    Let $z \in N(v_k) \setminus V(P)$.
    if $z \in \mathrm{FL}$ **and** $d(z) = 1$ **then** $\langle v_0, \ldots, v_k \to \mathrm{IN}, z \to \mathrm{LN} \rangle$
    else if $z \in \mathrm{FL}$ **and** $d(z) > 1$ **then** $\langle v_0, \ldots, v_{k-1} \to \mathrm{IN}, v_k \to \mathrm{LN} \rangle$
    else if $z \in \mathrm{BN}$ **then** $\langle v \to \mathrm{LN} \rangle$
    else if $z \in \mathrm{Free}$ **then** $\langle v_0, \ldots, v_k \to \mathrm{IN}, z \to \mathrm{IN} \mid\mid v \to \mathrm{LN} \rangle$ (B5)

Figure 2: An algorithm for Maximum Leaf Spanning Tree. The notation $\langle v \to \mathrm{IN} \mid\mid v \to \mathrm{LN} \rangle$ describes the corresponding branches, e.g., in this case v either becomes an internal node or a leaf (see Definition 2). To compute a solution for Maximum Leaf Spanning Tree of a given graph $G = (V, E)$, algorithm $\mathcal{M}$ must be called for each vertex $v \in V$ with $\mathrm{IN} = \{v\}$, $\mathrm{BN} = N(v)$ and $\mathrm{LN} = \mathrm{FL} = \emptyset$ as stated in Lemma 4.

Lemma 3. *Let $G = (V, E)$ be a graph, T a tree in G and $v \in \text{leaves}(T)$ such that $N(v) \setminus V_T = \{x_1, x_2\}$. If every optimal spanning tree $T' \supseteq T$ is such that v is an internal node and each x_i is a leaf in T', then there is also some optimal spanning tree where additionally each $w \in N(\{x_1, x_2\}) \setminus (\text{internal}(T) \cup \{v\})$ is a leaf.*

Proof. Since the instance is reduced and thus (R3) and (R5) cannot be applied, we have $N(x_2) \setminus \{v, x_1\} \neq \emptyset$. Let $T' \supseteq T$ be an optimal spanning tree as above, but some vertex in $(N(x_1) \cup N(x_2)) \setminus (\text{internal}(T) \cup \{v\})$ must be an internal node in all optimal solutions, say a neighbor w of x_1. Modify T' as follows. First, connect x_1 through w instead of v, which does not change the number of leaves, because w is already an internal node. Then connect x_2 through some other neighbor $u \in N(x_2) \setminus (\text{internal}(T) \cup \{v\})$ instead of v. This possibly destroys a leaf, u, but at the same time v becomes a leaf, so that the total number of leaves remains the same, a contradiction. □

Lemma 4. *Algorithm $\mathcal{M}$ can be used to solve the* Maximum Leaf Spanning Tree *problem for a graph $G = (V, E)$ if $|V| \geq 3$. Initially, for each $v \in V$ call algorithm $\mathcal{M}$ with $\text{IN} = \{v\}$, $\text{BN} = N(v)$ and $\text{LN} = \text{FL} = \emptyset$.*

Proof. The reduction rules update a partition $\mathcal{P} = (\text{Free}, \text{IN}, \text{BN}, \text{LN}, \text{FL})$ to a partition $\mathcal{P}' = (\text{Free}', \text{IN}', \text{BN}', \text{LN}', \text{FL}')$ so that any maximum leaf spanning tree T' that extends $\mathcal{P}'$ has at least as many leaves as any spanning tree T extending $\mathcal{P}$. Note that given some disjoint subsets IN, BN, LN, FL, the subset Free is uniquely determined by $V \setminus (\text{IN} \cup \text{BN} \cup \text{LN} \cup \text{FL})$. Thus, we omit the explicit notation of the set Free.

In the following, $(\text{IN} \cup \text{BN} \cup \text{LN} \cup \text{FL})_{x_1 \to X_1, \ldots, x_l \to X_l}$ denotes the partition $(\text{Free}', \text{IN}', \text{BN}', \text{LN}', \text{FL}')$ obtained from $(\text{Free}, \text{IN}, \text{BN}, \text{LN}, \text{FL})$ by the algorithm in the $x_1 \to X_1, \ldots, x_l \to X_l$ branch. In particular, whenever Algorithm $\mathcal{M}$ decides that some nodes $X \subseteq \text{BN} \cup \text{Free}$ become internal nodes, all nodes in $N(X) \cap \text{Free}$ become new branching nodes (BN) and all nodes in $N(X) \cap \text{FL}$ become leaves (LN). Hence, Algorithm $\mathcal{M}$ always computes an inner-maximal tree. It thus remains to show that if there is some spanning tree T with k leaves that extends the current $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, then Algorithm $\mathcal{M}$ calls itself with an new $(\text{IN}', \text{BN}', \text{LN}', \text{FL}')$ such that there is some spanning tree T' with k leaves that extends $(\text{IN}', \text{BN}', \text{LN}', \text{FL}')$, as well.

We prove this by induction. For the base step, notice that since $|V| \geq 3$, any spanning tree has at least one internal node. So, the initial branch will also consider a vertex v that is internal node of some maximum leaf spanning tree T^*. By Lemma 1, some maximum leaf spanning tree T^{**} will then extend $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ with $\text{IN} = \{v\}$, $\text{BN} = N(v)$ and $\text{LN} = \text{FL} = \emptyset$.

Now let T be a spanning tree with k leaves that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, and let $v \in \text{BN}$ be of maximum degree.

- If $d(v) \geq 3$ or $d(v) = 2$ and $N_{\text{FL}}(v) \neq \emptyset$, then Algorithm $\mathcal{M}$ calls itself recursively in (B1). Since v is either an internal node or a leaf in any spanning tree, T extends either $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}}$ or $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{LN}}$.

- Consider $d(v) = 2$, i.e., $N_{\text{Free}}(v) = \{x_1, x_2\}$, $d(x_1) \leq d(x_2)$, and first discuss the case when $d(x_1) = 2$, corresponding to (B2) in the algorithm. If $N(x_1) \setminus \{v\} = \{z\}$, such that $z \in \text{FL}$, we do not need to branch, since x_1 must be somehow connected to the tree in any solution extending $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, and v is the only choice. Thus, $v \to \text{IN}$.
  If otherwise $z \in \text{Free}$, then T either extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{LN}}$, or $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{LN}}$, or $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{IN}}$, because if v is not a leaf in T, then it is an internal node and x_1 is either a leaf or an internal node. Note that $z \in \text{BN}$ is not possible since an application of Reduction Rule (R4) would be done on vertex x_1, and vertex x_1 would be in FL.

- In the case where $d(v) = 2$, $3 \leq d(x_1) \leq d(x_2)$ and $N(x_1) \cap N(x_2) \cap (\text{Free} \cup \text{BN}) = \{v\}$, the algorithm branches on all possibilities whether v, x_1 and x_2 are internal nodes or leaves. This means we consider (in principle) the following complete branch:

$$\begin{aligned}&\langle v \to \text{LN} \mid\mid v \to \text{IN}, x_1 \to \text{IN} \mid\mid \\ &\quad v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{IN} \mid\mid v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{LN}\rangle\end{aligned}$$

  If there is some $z \in (N_{\text{FL}}(x_1) \cap N_{\text{FL}}(x_2))$ with $d(z) \leq 2$, not both x_1 and x_2 can be leaves and we skip the last branch (which yields (B4)). Otherwise, Lemma 3 guarantees that in the last branch that all other neighbors of x_1 and x_2 are leaves in some optimal solution, as well. Hence, there is a tree that extends either
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{LN}}$, or
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{IN}}$, or
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{IN}}$, or
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{LN}, N_{\text{Free}}(\{x_1, x_2\}) \to \text{FL}, N_{\text{BN}}(\{x_1, x_2\}) \setminus \{v\} \to \text{LN}}$.

- In the case where $d(v) = 2$, $3 \leq d(x_1) \leq d(x_2)$ and $N(x_1) \cap N(x_2) \cap (\text{Free} \cup \text{BN}) \neq \{v\}$, we can assume that if v is an internal node in every optimal solution, either x_1 or x_2 is an internal node as well. Otherwise, we could connect x_1 and x_2 to $z \in (N(x_1) \cap N(x_2)) \setminus \text{FL}$ instead of connecting them to v, which might destroy the leaf z, that must be connected somehow else, but yields the new leaf v. Since z is either a branching node or a free node, this is still allowed. Hence, there is also some optimal solution that extends
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{LN}}$, or
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{IN}}$, or
  $(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{IN}, x_1 \to \text{LN}, x_2 \to \text{IN}}$.
  This (again) corresponds to case (B4) in our algorithm.

- Finally, if $d(v) = 1$, let $P = (v = v_0, v_1, \ldots, v_k)$ be a maximum path such that $d(v_i) = 2$, $1 \leq i \leq k$, $v_1, \ldots, v_k \in \text{Free}$ and let $z \in (N(v_k) \setminus V(P))$, as described in Algorithm $\mathcal{M}$. If $z \in \text{FL}$ and $d(z) = 1$, all vertices in P must be internal nodes in any spanning tree that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$, because there is no other way to connect z, cf. Reduction Rule (R6). If

otherwise $d(z) > 1$, there is always an inner-maximal solution where v_k is a leaf by a simple exchange argument. Namely, assume that T is an optimal tree that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ such that z is attached to v_k in T. If there exists a vertex u, $u \neq v_k$, such that $u \in (N(z) \cap \text{internal}(T))$ then the tree T' obtained from T such that z is attached to u instead of v_k and v_k is a leaf has one more leaves than T, contradicting the optimality of T. Assume now that such a vertex u does not exist and $N(z) \cap \text{internal}(T) = \{v_k\}$. Let u be a neighbor of z. Thus $u \in \text{leaves}(T)$. Consider again the tree T' obtained from T by attaching z to u instead of v_k. Thus v_k is a leaf in T' (recall that the degree of $d(v_k) = 2$) and u is an internal node of T'. As a consequence, the number of leaves in T' is equal to the number of leaves in T. So we can assume that there is an optimal tree that extends $(\text{IN}, \text{BN}, \text{LN}, \text{FL})$ such that v_k is a leaf (and z is in FL).

If on the other hand $z \in \text{BN}$, then the nodes in P must either be connected through v or through z, and hence we can just decide to make v a leaf, again by a simple exchange argument.

Now assume $z \in \text{Free}$. Since (w.l.o.g.) T is inner-maximal we know by [16] that there is some inner-maximal T' that extends either
$(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v \to \text{LN}}$, or
$(\text{IN}, \text{BN}, \text{LN}, \text{FL})_{v, v_1, \ldots, v_k, z \to \text{IN}}$ in this case.

Since this concludes a complete distinction of all possible values of $d(v)$, the claim follows by induction. □

5. Analysis of the Running Time

To analyze the running time of our algorithm, we use the Measure-and-Conquer technique. We first define a suitable measure $\mu(I)$ of an annotated instance $I = (G, \text{IN}, \text{BN}, \text{LN}, \text{FL})$. We then consider each possible input instance and prove in a sequence of lemmata that for this measure we obtain a branching number less than 1.8966. Together, these lemmata therefore establish Theorem 11 by a simple induction proof as outlined in Section 2. This theorem shows that the worst-case running-time of Algorithm $\mathcal{M}$ is upper bounded by $O(1.8966^n)$. As running-times obtained via Measure-and-Conquer tends to be overestimate, Theorem 12 provides a lower-bound on the worst-case running-time. Namely we contruct a family of graphs on which Algorithm $\mathcal{M}$ needs $\Omega(3^{n/3}) = \Omega(1.4422^n)$ time.

To analyze the running time, we use the following measure on the size of an instance $I = (G, \text{IN}, \text{BN}, \text{LN}, \text{FL})$:

$$\mu(I) = \sum_{i=1}^{n} \epsilon_i^{\text{BN}} |\text{BN}_i| + \sum_{i=2}^{n} \epsilon_i^{\text{Free}} |\text{Free}_i| + \sum_{i=2}^{n} \epsilon_i^{\text{FL}} |\text{FL}_i|,$$

where BN_i (resp. Free_i and FL_i) denotes the set of vertices in BN (resp. Free and FL) with degree i, and the values of the ϵ's are chosen in $[0, 1]$ so that $\mu(I) \leq n$, more precisely:

- $\epsilon_0^{\text{Free}} = \epsilon_1^{\text{Free}} = 0$, $\epsilon_2^{\text{Free}} = 0.731975$, $\epsilon_3^{\text{Free}} = 0.946609$, and $\epsilon_i^{\text{Free}} = 1$ for all $i \geq 4$;
- $\epsilon_0^{\text{BN}} = 0$, $\epsilon_1^{\text{BN}} = 0.661662$, $\epsilon_i^{\text{BN}} = 0.730838$ for all $i \geq 2$;
- $\epsilon_0^{\text{FL}} = \epsilon_1^{\text{FL}} = 0$, $\epsilon_2^{\text{FL}} = 0.331595$, $\epsilon_3^{\text{FL}} = 0.494066$, and $\epsilon_i^{\text{FL}} = 0.628886$ for all $i \geq 4$.

Lemma 5. *Let $G = (V, E)$ be a graph and let* $\text{Free} \cup \text{BN} \cup \text{LN} \cup \text{FL} \cup \text{IN}$ *be a partition of V. Moreover let $v \in \text{BN}$ such that $d(v) \geq 3$ or $d(v) = 2$ and there is some $u \in N_{\text{FL}}(v)$. Then branching according to (B1) yields a branching number less than* 1.8966.

Proof. By Reduction Rules (R3) and (R6), we have $d(u) \geq 2$ for all $u \in N_{\text{Free}\cup\text{FL}}(v)$.

1. In the first branch, v becomes a leaf. Therefore, the degree of all nodes in $N_{\text{Free}\cup\text{FL}}(v)$ decreases by one, as the edge to v is removed. This implies a change in the measure of at least
$$\Delta_1 = \epsilon_{d(v)}^{\text{BN}} + \sum_{x \in N_{\text{Free}}(v)} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{Free}}) + \sum_{y \in N_{\text{FL}}(v)} (\epsilon_{d(y)}^{\text{FL}} - \epsilon_{d(y)-1}^{\text{FL}}).$$
2. In the second branch, v is added to the internal nodes. Thus, all nodes in $N_{\text{Free}}(v)$ are added to the branching nodes. This reduces the degree of all these nodes by at least one, since the edge to v is not counted anymore. Moreover, all nodes in $N_{\text{FL}}(v)$ are now leaf nodes. Thus, the measure decreases by at least
$$\Delta_2 = \epsilon_{d(v)}^{\text{BN}} + \sum_{x \in N_{\text{Free}}(v)} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{BN}}) + \sum_{y \in N_{\text{FL}}(v)} \epsilon_{d(y)}^{\text{FL}}.$$

Since higher degrees only imply a higher change, it is now sufficient to test all combinations where $d(v) = 3$ or $d(v) = 2$ and there is some $u \in N_{\text{FL}}(v)$. For all other nodes $u \in N_{\text{Free}\cup\text{FL}}(v)$, we can similarly assume $2 \leq d(u) \leq 5$. The worst case occurs when $d(v) = 3$ and v has three free neighbors of degree at least five. The corresponding branching vector $(1.538324, 0.730838)$ has a branching number smaller than 1.8966. □

Lemma 6. *Let $G = (V, E)$ be a graph and let* $\text{Free} \cup \text{BN} \cup \text{LN} \cup \text{FL} \cup \text{IN}$ *be a partition of V. Moreover let $v \in \text{BN}$ such that $d(v) = 2$ and there is some $x_1 \in N_{\text{Free}}(v)$ with $d(x_1) = 2$ and the remaining $z \in N(x_1) \setminus \{v\}$ is contained in* Free. *Then branching according to (B2) yields a branching number less than* 1.8966.

Proof. By Reduction Rule (R5), we know that $z \neq x_2$. Moreover, (R3) implies $d(z) \geq 2$.

1. Again, v becomes leaf in the first branch. Similar to Lemma 5, this implies a change in the measure of at least

$$\begin{aligned}\Delta_1 &= \epsilon_2^{\mathrm{BN}} + (\epsilon_2^{\mathrm{Free}} - \epsilon_1^{\mathrm{FL}}) + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{Free}}) \\ &= \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{Free}} + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{Free}}),\end{aligned}$$

   because x_1 becomes a floating leaf of degree one and the degree of x_2 decreases by one.
2. In the second branch, both v and x_1 become internal nodes, which implies that z and x_2 become branching nodes. Again, $d(z)$ and $d(x_2)$ decrease by one. The measure decreases by at least

$$\Delta_2 = \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{Free}} + (\epsilon_{d(z)}^{\mathrm{Free}} - \epsilon_{d(z)-1}^{\mathrm{BN}}) + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{BN}}).$$

3. In the third branch, v becomes an internal node and x_1 becomes a leaf connected to v. Thus, x_2 is now a branching node and $d(x_2)$ decreases. Moreover, $d(z)$ decreases by one as well. This implies that the measure is reduced by at least

$$\Delta_3 = \epsilon_2^{\mathrm{BN}} + \epsilon_2^{\mathrm{Free}} + (\epsilon_{d(z)}^{\mathrm{Free}} - \epsilon_{d(z)-1}^{\mathrm{Free}}) + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{BN}}).$$

Since $d(v) = d(x_1) = 2$, we need to try all possible combinations of $d(z)$ and $d(x_2)$, both between 2 and 5. The worst case occurs when $d(z) = d(x_2) = 5$. The corresponding branching vector $(1.462813, 1.731975, 2.001137)$ has branching number 1.8965. □

Lemma 7. *Let $G = (V, E)$ be a graph and let* $\mathrm{Free} \cup \mathrm{BN} \cup \mathrm{LN} \cup \mathrm{FL} \cup \mathrm{IN}$ *be a partition of V. Moreover let $v \in \mathrm{BN}$ such that $N_{\mathrm{Free}}(v) = \{x_1, x_2\}$ with $3 \leq d(x_1) \leq d(x_2)$ and let $(N(x_1) \cap N(x_2)) \setminus \mathrm{FL} = \{v\}$. Finally, assume $x_1 \notin N(x_2)$. Then branching according to (B3) yields a branching number less than* 1.8966.

Proof. 1. In the first branch, v becomes a leaf, which yields

$$\Delta_1 = \epsilon_2^{\mathrm{BN}} + (\epsilon_{d(x_1)}^{\mathrm{Free}} - \epsilon_{d(x_1)-1}^{\mathrm{Free}}) + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{Free}}).$$

2. In the second branch, v and x_1 become internal nodes. As a consequence, x_2 becomes a branching leaf and its degree decreases by one. Furthermore, the degree of all nodes in $N_{\mathrm{Free}\cup\mathrm{FL}}(x_1)$ decreases by one. We gain at least

$$\begin{aligned}\Delta_2 = \epsilon_2^{\mathrm{BN}} + \epsilon_{d(x_1)}^{\mathrm{Free}} + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{BN}}) + \sum_{x \in N_{\mathrm{Free}}(x_1)} (\epsilon_{d(x)}^{\mathrm{Free}} - \epsilon_{d(x)-1}^{\mathrm{BN}}) \\ + \sum_{y \in N_{\mathrm{FL}}(x_1)} \epsilon_{d(y)}^{\mathrm{FL}} + \sum_{z \in N_{\mathrm{BN}}(x_1) \setminus \{v\}} (\epsilon_{d(z)}^{\mathrm{BN}} - \epsilon_{d(z)-1}^{\mathrm{BN}}).\end{aligned}$$

3. In the third branch, v and x_2 become internal nodes, while x_1 becomes a leaf. Thus, the degree decreases by one for all nodes in $N_{\text{Free}\cup\text{FL}}(x_1)$, as well as for all nodes in $N_{\text{BN}}(x_2) \setminus \{v\}$. Moreover, all nodes in $N_{\text{Free}}(x_2)$ become branching nodes and all nodes in $N_{\text{FL}}(x_2)$ become leaves. Since $(N(x_1) \cap N(x_2)) \setminus \text{FL} = \emptyset$, the measure decreases by at least

$$\begin{aligned}\Delta_3 = \epsilon_2^{\text{BN}} + \epsilon_{d(x_1)}^{\text{Free}} + \epsilon_{d(x_2)}^{\text{Free}} + \sum_{x \in N_{\text{Free}}(x_1)} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{Free}}) \\ + \sum_{y \in N_{\text{FL}}(x_1) \setminus N(x_2)} (\epsilon_{d(y)}^{\text{FL}} - \epsilon_{d(y)-1}^{\text{FL}}) + \sum_{z \in N_{\text{BN}}(\{x_1,x_2\}) \setminus \{v\}} (\epsilon_{d(z)}^{\text{BN}} - \epsilon_{d(z)-1}^{\text{BN}}) \\ + \sum_{x' \in N_{\text{Free}}(x_2)} (\epsilon_{d(x')}^{\text{Free}} - \epsilon_{d(x')-1}^{\text{BN}}) + \sum_{y' \in N_{\text{FL}}(x_2)} \epsilon_{d(y')}^{\text{FL}}.\end{aligned}$$

4. In the last branch, v becomes an internal node, x_1 and x_2 become leaves, and all nodes in $N_{\text{Free}}(\{x_1, x_2\})$ become floating leaves. Moreover, all nodes in $N_{\text{BN}}(\{x_1, x_2\}) \setminus \{v\}$ become leaves as well and finally, the degree decreases by at least one for all $u \in N_{\text{FL}}(\{x_1, x_2\})$. This implies that the measure decreases by at least

$$\begin{aligned}\Delta_4 = \epsilon_2^{\text{BN}} + \epsilon_{d(x_1)}^{\text{Free}} + \epsilon_{d(x_2)}^{\text{Free}} + \sum_{x \in N_{\text{Free}}(\{x_1,x_2\})} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{FL}}) \\ + \sum_{y \in N_{\text{FL}}(\{x_1,x_2\}) \setminus (N(x_1) \cap N(x_2))} (\epsilon_{d(y)}^{\text{FL}} - \epsilon_{d(y)-1}^{\text{FL}}) \\ + \sum_{y \in \text{FL} \cap N(x_1) \cap N(x_2)} (\epsilon_{d(y)}^{\text{FL}} - \epsilon_{d(y)-2}^{\text{FL}}) \\ + \sum_{z \in N_{\text{BN}}(\{x_1,x_2\}) \setminus \{v\}} \epsilon_{d(z)}^{\text{BN}}.\end{aligned}$$

Again, we have to compute all possible neighborhoods. This requires us to test all $3 \leq d(x_1) \leq d(x_2) \leq 5$, all $1 \leq d(u) \leq 2$ for all $u \in N_{\text{BN}}(\{x_1, x_2\})$, all $2 \leq d(u) \leq 5$ for each $u \in N_{\text{FL}}(\{x_1, x_2\})$ and finally all $2 \leq d(u) \leq 5$ for all $u \in N_{\text{Free}}(\{x_1, x_2\})$.

Note that we can assume that all floating leaves in $N(x_i)$ are of degree at least two. Otherwise, the branches that turn x_i into a leaf node yield new instances that will be solved in polynomial time, because they are obvious "No" instances. This is detected by our algorithm, since such floating leaves would become unreachable due to Reduction Rule (R7). Thus, in such a case the exponential parts of the running time only depend on the other branches, which yields a much better time bound, even if some floating leaves are of degree one. Similarly, we can assume that floating leaves of degree two are not contained in $N(x_1) \cap N(x_2)$, because otherwise the last branch (both, x_1 and x_2 are in LN) is found to be a "No" instance in polynomial time.

It turns out that the largest branching number in this case is smaller than 1.8506 with a branching vector $(0.730838, 2.476690, 3.216207, 8.218955)$ in the worst case that occurs when $d(x_1) = d(x_2) = 5$, $N_{\text{Free}}(x_1) = \{u\}$ with $d(u) = 5$, $N_{\text{FL}}(x_1) = \emptyset$, $N_{\text{BN}}(x_1) = \{u_1, u_2, u_3\}$ with $d(u_1) = d(u_2) = d(u_3) = 2$, $N_{\text{Free}}(x_2) = \{w\}$ with $d(w) = 2$, $N_{\text{FL}}(x_2) = \emptyset$, and $N_{\text{BN}}(x_2) = \{w_1, w_2, w_3\}$ with $d(w_1) = d(w_2) = d(w_3) = 2$. □

Lemma 8. *Let $G = (V, E)$ be a graph and let* $\text{Free} \cup \text{BN} \cup \text{LN} \cup \text{FL} \cup \text{IN}$ *be a partition of V. Moreover let $v \in$ BN such that $N_{\text{Free}}(v) = \{x_1, x_2\}$ with $3 \leq d(x_1) \leq d(x_2)$ and let $(N(x_1) \cap N(x_2)) \setminus \text{FL} = \{v\}$. Finally, let $x_1 \in N(x_2)$. Then branching according to (B3) yields a branching number less than* 1.8966*.*

Proof. The proof is very similar to the previous lemma, we only need to make sure that the edge between x_1 and x_2 is not counted twice.

Observe that we branch exactly as in Lemma 7, but obtain slightly different results, because there are fewer neighbors of x_1 and x_2, but x_2 is now affected whenever we decide whether x_1 is an internal node or a leaf. Analogously, we obtain the following branches.

1. In the first branch, v becomes a leaf and as above, we gain at least
$$\Delta_1 = \epsilon_2^{\text{BN}} + (\epsilon_{d(x_1)}^{\text{Free}} - \epsilon_{d(x_1)-1}^{\text{Free}}) + (\epsilon_{d(x_2)}^{\text{Free}} - \epsilon_{d(x_2)-1}^{\text{Free}}).$$
2. In the second branch, v and x_1 become internal nodes. As a consequence, x_2 becomes a branching node and its degree decreases by two, as the edge to x_1 and the edge to v are not counted anymore. For all other vertices in $N_{\text{Free}\cup\text{FL}}(x_1)$, the degree decreases by one and they turn into either branching nodes or leaves. This implies a loss in the measure of at least
$$\Delta_2 = \epsilon_2^{\text{BN}} + \epsilon_{d(x_1)}^{\text{Free}} + (\epsilon_{d(x_2)}^{\text{Free}} - \epsilon_{d(x_2)-2}^{\text{BN}}) + \sum_{x \in N_{\text{Free}}(x_1)\setminus\{x_2\}} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{BN}}) + \sum_{y \in N_{\text{FL}}(x_1)} \epsilon_{d(y)}^{\text{FL}} + \sum_{z \in N_{\text{BN}}(x_1)\setminus\{v\}} (\epsilon_{d(z)}^{\text{BN}} - \epsilon_{d(z)-1}^{\text{BN}}).$$
3. In the third branch, v and x_2 become internal nodes and x_1 becomes a leaf. This case is identical to the third branch in Lemma 7, except that x_1 and x_2 each have one neighbor less. We gain at least
$$\begin{aligned}\Delta_3 = \epsilon_2^{\text{BN}} &+ \epsilon_{d(x_1)}^{\text{Free}} + \epsilon_{d(x_2)}^{\text{Free}} + \sum_{x \in N_{\text{Free}}(x_1)\setminus\{x_2\}} (\epsilon_{d(x)}^{\text{Free}} - \epsilon_{d(x)-1}^{\text{Free}}) \\ &+ \sum_{y \in N_{\text{FL}}(x_1)\setminus N(x_2)} (\epsilon_{d(y)}^{\text{FL}} - \epsilon_{d(y)-1}^{\text{FL}}) + \sum_{z \in N_{\text{BN}}(x_1)\setminus\{v\}} (\epsilon_{d(z)}^{\text{BN}} - \epsilon_{d(z)-1}^{\text{BN}}) \\ &+ \sum_{y' \in N_{\text{FL}}(x_2)} \epsilon_{d(y')}^{\text{FL}} + \sum_{x' \in N_{\text{Free}}(x_2)\setminus\{x_1\}} (\epsilon_{d(x')}^{\text{Free}} - \epsilon_{d(x')-1}^{\text{BN}}) \\ &+ \sum_{z' \in N_{\text{BN}}(x_2)\setminus\{v\}} (\epsilon_{d(z')}^{\text{BN}} - \epsilon_{d(z')-1}^{\text{BN}}).\end{aligned}$$

4. The last branch, where v becomes an internal node and both x_1 and x_2 become leaves is again similar to the last branch in Lemma 7, except that there are fewer neighbors of x_1 and x_2. The measure decreases by at least

$$\Delta_4 = \epsilon_2^{\mathrm{BN}} + \epsilon_{d(x_1)}^{\mathrm{Free}} + \epsilon_{d(x_2)}^{\mathrm{Free}} + \sum_{x \in N_{\mathrm{Free}}(\{x_1,x_2\}) \setminus \{x_1,x_2\}} (\epsilon_{d(x)}^{\mathrm{Free}} - \epsilon_{d(x)-1}^{\mathrm{FL}}) + \sum_{y \in N_{\mathrm{FL}}(\{x_1,x_2\}) \setminus (N(x_1) \cap N(x_2))} (\epsilon_{d(y)}^{\mathrm{FL}} - \epsilon_{d(y)-1}^{\mathrm{FL}}) + \sum_{y \in \mathrm{FL} \cap N(x_1) \cap N(x_2)} (\epsilon_{d(y)}^{\mathrm{FL}} - \epsilon_{d(y)-2}^{\mathrm{FL}}) + \sum_{z \in N_{\mathrm{BN}}(\{x_1,x_2\}) \setminus \{v\}} \epsilon_{d(z)}^{\mathrm{BN}}.$$

Similar to Lemma 7, we have to test all $3 \leq d(x_1) \leq 5$ and $d(x_1) \leq d(x_2) \leq 5+1$, because the $d(x_2)$ decreases by two now. Moreover, we need to try all $1 \leq d(u) \leq 2$ for all $u \in N_{\mathrm{BN}}(\{x_1, x_2\})$, all $2 \leq d(u) \leq 5$ for each $u \in N_{\mathrm{FL}}(\{x_1, x_2\})$ and finally all $2 \leq d(u) \leq 5$ for all $u \in N_{\mathrm{Free}}(\{x_1, x_2\})$. However, this time we need to be careful, because one free neighbor of x_1 is x_2 and vice versa. Thus, there are fewer neighbors overall.

As argued in the proof of Lemma 7, it is sufficient for the analysis of the running time of our algorithm to assume that all floating leaves in $N(x_i)$ are of degree at least two and those that are in $N(x_1) \cap N(x_2)$ even of degree three.

The worst branching number is less than 1.8921 and the corresponding branching vector is $(0.784229, 2.338338, 3.007542, 6.025304)$ which is obtained when $d(x_1) = 4$, $d(x_2) = 5$, $N_{\mathrm{Free}}(x_1) = \{u_1, x_2\}$ with $d(u_1) = 5$, $N_{\mathrm{FL}}(x_1) = \emptyset$, $N_{\mathrm{BN}}(x_1) = \{u\}$ with $d(u) = 2$ $N_{\mathrm{Free}}(x_2) = \{x_1\}$, $N_{\mathrm{FL}}(x_2) = \emptyset$, and $N_{\mathrm{BN}}(x_2) = \{u_1, u_2, u_3\}$ with $d(u_1) = d(u_2) = d(u_3) = 2$. □

Lemma 9. *Let $G = (V, E)$ be a graph and let* $\mathrm{Free} \cup \mathrm{BN} \cup \mathrm{LN} \cup \mathrm{FL} \cup \mathrm{IN}$ *be a partition of V. Moreover let $v \in$* BN *such that $N_{\mathrm{Free}}(v) = \{x_1, x_2\}$ with $3 \leq d(x_1) \leq d(x_2)$ and let $(N(x_1) \cap N(x_2)) \setminus$* FL $\neq \{v\}$. *Then branching according to (B4) yields a branching number less than* 1.8966.

Proof. Similar to Lemma 7 and Lemma 8, x_1 and x_2 can possibly be neighbors.

1. In the first branch, v becomes a leaf. Similar to above, we obtain at least

$$\Delta_1 = \epsilon_2^{\mathrm{BN}} + (\epsilon_{d(x_1)}^{\mathrm{Free}} - \epsilon_{d(x_1)-1}^{\mathrm{Free}}) + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{Free}}).$$

2. In the second branch, v and x_1 become internal nodes. As a consequence, the degree decreases for all vertices in $N_{\mathrm{Free} \cup \mathrm{FL}}(\{v, x_1\})$ and these vertices turn into branching nodes or leaves, respectively. The measure decreases by at least

$$\Delta_2 = \epsilon_2^{\mathrm{BN}} + \epsilon_{d(x_1)}^{\mathrm{Free}} + (\epsilon_{d(x_2)}^{\mathrm{Free}} - \epsilon_{d(x_2)-1}^{\mathrm{BN}}) + \sum_{x \in N_{\mathrm{Free}}(x_1) \setminus \{x_2\}} (\epsilon_{d(x)}^{\mathrm{Free}} - \epsilon_{d(x)-1}^{\mathrm{BN}}) + \sum_{y \in N_{\mathrm{FL}}(x_1)} \epsilon_{d(y)}^{\mathrm{FL}} + \sum_{z \in N_{\mathrm{BN}}(x_1) \setminus \{v\}} (\epsilon_{d(z)}^{\mathrm{BN}} - \epsilon_{d(z)-1}^{\mathrm{BN}}).$$

Note that when $x_2 \in N(x_1)$, $d(x_2)$ decreases even more. However, this estimation is good enough to obtain the claimed bounds.

3. In the last branch, v and x_2 become internal nodes and x_1 becomes a leaf. As usual, the measure decreases by at least

$$\Delta_3 = \epsilon_2^{\mathrm{BN}} + \epsilon_{d(x_1)}^{\mathrm{Free}} + \epsilon_{d(x_2)}^{\mathrm{Free}} + \sum_{x \in N_{\mathrm{Free}}(x_1) \setminus \{x_2\}} (\epsilon_{d(x)}^{\mathrm{Free}} - \epsilon_{d(x)-1}^{\mathrm{Free}}) + \sum_{y \in N_{\mathrm{FL}}(x_1)} (\epsilon_{d(y)}^{\mathrm{FL}} - \epsilon_{d(y)-1}^{\mathrm{FL}}) + \sum_{z \in N_{\mathrm{BN}}(x_1) \setminus \{v\}} (\epsilon_{d(z)}^{\mathrm{BN}} - \epsilon_{d(z)-1}^{\mathrm{BN}}).$$

In all three cases, we only analyzed how the neighbors of x_1 are affected and omitted the neighbors of x_2. Thus, we do not have to distinguish between vertices in $N(x_1) \setminus N(x_2)$ and $N(x_1) \cap N(x_2)$. Similar to previous lemmata, we can safely assume that $d(u) \geq 2$ for all floating leaves $u \in N(x_1)$. In order to compute all possible branching vectors, we need to test all $3 \leq d(x_1) \leq d(x_2) \leq 5$. Furthermore, we need to try all $1 \leq d(u) \leq 2$ for all $u \in N_{\mathrm{BN}}(x_1)$, all $2 \leq d(u) \leq 5$ for each $u \in N_{\mathrm{FL}}(x_1)$ and finally all $2 \leq d(u) \leq 5$ for all $u \in N_{\mathrm{Free}}(x_1)$.

The worst case occurs when $d(x_1) = d(x_2) = 5$, $N_{\mathrm{Free}}(x_1) = \{u_1, u_2\}$ with $d(u_1) = d(u_2) = 5$, $N_{\mathrm{FL}}(x_1) = \emptyset$, and $N_{\mathrm{BN}}(x_1) = \{u_1' u_2'\}$ with $d(u_1') = d(u_2') = 2$. The corresponding branching vector $(0.730838, 2.407514, 2.869190)$ has branching number 1.8966. □

Lemma 10. *Let $G = (V, E)$ be a graph and let* $\mathrm{Free} \cup \mathrm{BN} \cup \mathrm{LN} \cup \mathrm{FL} \cup \mathrm{IN}$ *be a partition of V. Moreover let $v \in$* BN *such that $d(v) = 1$. Then branching according to (B5) yields a branching number less than* 1.8966*.*

Proof. Let $v_1, \ldots, v_k$ and $z \in V$ as described in Algorithm $\mathcal{M}$ and recall that $d(z) \geq 3$ and $z \in$ Free.

1. In the first branch, v becomes an internal node (as well as all $v_1, \ldots, v_k$ and z do). This implies that the measure decreases by at least

$$\Delta_1 = \epsilon_1^{\mathrm{BN}} + k\epsilon_2^{\mathrm{Free}} + \epsilon_{d(z)}^{\mathrm{Free}}.$$

2. In the other branch, v becomes a leaf. If $k = 0$, then the degree of z will decrease, and if $k > 0$, the vertex v_1 becomes a floating leaf of degree one. Therefore, we gain at least

$$\Delta_2 = \epsilon_1^{\mathrm{BN}} + \min(\epsilon_{d(z)}^{\mathrm{Free}} - \epsilon_{d(z)-1}^{\mathrm{Free}}, \epsilon_2^{\mathrm{Free}}).$$

The worst case occurs when $d(z) = 5$ and $k = 0$. The corresponding branching vector $(1.661662, 0.661662)$ has a branching number less than 1.8966. □

The lemmata of this section consider all possible cases of the branching algorithm and they guarantee by an inductive proof over the measure that the search tree of an execution of the branching algorithm on a graph of n vertices contains at most 1.8966^n leaves. Together with Lemma 4, they also guarantee the correctness of our algorithm. Thus we can conclude our main result.

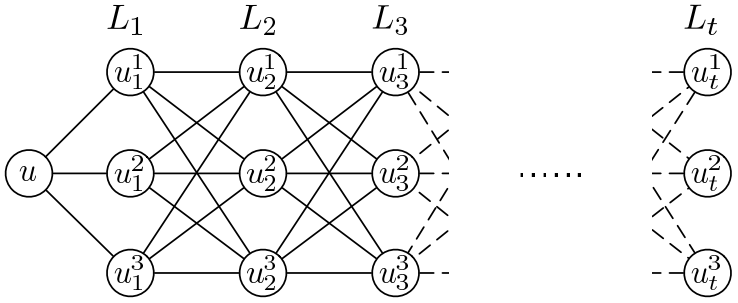

Figure 3: Graph G_t with t levels.

Theorem 11. *The given algorithm solves the* Maximum Leaf Spanning Tree *problem in time* $O(1.8966^n)$.

It is known that the current tools for the time analysis of branching algorithms, even Measure-and-Conquer, might overestimate the worst-case running time. The following theorem gives a lower bound on the worst-case running time of our algorithm. We recall that the algorithm of Fomin et al. [11] solving the problem MLST has a worst-case running time upper bounded by $O(1.9407^n)$ and a lower bound of $\Omega(1.3195^n)$.

Theorem 12. *Our algorithm solving the problem MLST has a worst-case running time lower bounded by* $\Omega(3^{n/3}) = \Omega(1.4422^n)$.

Proof. Consider the graphs $G_t = (V_t, E_t)$ for integers $t \geq 1$, constructed as follows (see also Figure 3). The vertex set V_t is defined as $\{u\} \cup \bigcup_{i=1}^{t} L_i$ where $L_i = \{u_i^1, u_i^2, u_i^3\}$. The edge set E_t consists of all edges between any two sets L_i and L_{i+1}, $1 \leq i < t$ and all edges between u and the vertices of L_1. Thus the graph consists of a vertex u and a collection of t independent sets L_i (called "levels"), $1 \leq i \leq t$, such that each vertex of level L_i is adjacent to all vertices of the next level L_{i+1} and the vertex u is adjacent to each vertices of L_1.

Suppose that the algorithm is initially called for vertex u; hence u becomes an internal node and all vertices $N(u) = L_1$ become branching nodes. Clearly, none of the Reduction Rules (R1) to (R7) can be applied. Thus algorithm $\mathcal{M}$ chooses a $v \in$ BN of maximum degree. Since u_1^1, u_1^2 and u_1^3 have degree 3, w.l.o.g., assume that u_1^1 is chosen by (B1). In the branch $u_1^1 \to$ IN, all vertices of $N_{\text{Free}}(u_1^1)$ are set to branching nodes and u_1^2 and u_1^3 are set to leaf nodes by subsequent applications of (R1) and (R2). Thus the new set of branching nodes is L_2 and Free $= \bigcup_{i=3}^{t} L_i$ (subproblem Π_1). In the branch $u_1^1 \to$ LN, we have BN $= \{u_1^2, u_1^3\}$ and no reduction rules can be applied. Then in this branch, suppose that u_1^2 is chosen by (B1). Again, either $u_1^2 \to$ IN, and u_1^3 is moved to LN by subsequent applications of (R1) and (R2), and the new set of branching nodes is L_2 and Free $= \bigcup_{i=3}^{t} L_i$ (subproblem Π_2); or $u_1^2 \to$ LN and the remaining vertex in BN is $\{u_1^3\}$. In such a case, algorithm $\mathcal{M}$ applies Rule

(R7) on the edges $\{u_1^1, u_2^j\}$ and $\{u_1^2, u_2^j\}$ for $j \in \{1, 2, 3\}$. Thus u_1^3 becomes a cut vertex in the graph. By subsequent application of (R6), vertex u_1^3 is put in IN, BN $= L_2$ and Free $= \bigcup_{i=3}^{t} L_i$ (subproblem Π_3); Note also that all subproblems Π_1, Π_2 and Π_3 have the same sets BN (i.e., L_2) and Free.

By construction of G_t, the same arguments recursively apply to all further levels (except L_t) and algorithm $\mathcal{M}$ branches in a similar way. Here we summarize the subsequent applications of (B1) on the vertices of L_2: $\langle u_2^1 \to \text{IN}, u_2^2 \to \text{LN}, u_2^3 \to \text{LN}, L_3 \to \text{BN} \mid\mid u_2^1 \to \text{LN}, u_2^2 \to \text{IN}, u_2^3 \to \text{LN}, L_3 \to \text{BN} \mid\mid u_2^1 \to \text{LN}, u_2^2 \to \text{LN}, u_2^3 \to \text{IN}, L_3 \to \text{BN}\rangle$.

Inductively, it can easily be shown that such a branching applies to each of the first $t-1$ levels (due to (R2) and makes the vertices of L_t leaf nodes as soon as a vertex of L_{t-1} is set to be an internal node). As a consequence, the worst-case running time is lower bounded by $\Omega(3^{t-1}) = \Omega(3^{n/3})$. □

[1] H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

[2] P. Bonsma. *Sparse cuts, matching-cuts and leafy trees in graphs.* PhD thesis, University of Twente, the Netherlands, 2006.

[3] P. S. Bonsma, T. Brüggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In *Proceedings of the 28th Conference on Mathematical Foundations of Computer Science (MFCS)*, vol. 2747 of LNCS, pages 259–268. Springer, 2003.

[4] P. S. Bonsma and F. Zickfeld. Spanning trees with many leaves in graphs without diamonds and blossoms. In *Proceedings of the of 8th Latin American Theoretical Informatics Symposium (LATIN)*, vol. 4957 of LNCS, pages 531–543. Springer, 2008.

[5] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):908–920, 2004.

[6] 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.

[7] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *Feasible Mathematics II*, pages 219–244. Birkhäuser, 1995.

[8] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In *Proceedings of the 1st Workshop Algorithms and Complexity in Durham (ACiD)*, pages 1–41. King's College Publications, 2005.

[9] 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 *Proceedings of the 20th IARCS*

Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pages 240–251. Springer, 2000.

[10] 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. In *Proceedings of 4th International Workshop on Parameterized and Exact Computation (IWPEC)*, vol. 5917 of LNCS, pages 161–172. Springer, 2009.

[11] F. V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008.

[12] F. V. Fomin, F. Grandoni, and D. Kratsch. A Measure & Conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5): (2009), article 25.

[13] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms.* Springer, 2010.

[14] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, 1979.

[15] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. Proceedings of the *35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372. ACM Press, 2003.

[16] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, vol. 5369 of LNCS, pages 270–281. Springer, 2008.

[17] W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 112–122. ACM Press, 2002.

[18] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[19] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica* 40(4):245–269, 2004.

[20] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 7–14. ACM Press, 1999.