Algorithmica manuscript No.
(will be inserted by the editor)

A New Algorithm for Finding Trees with Many Leaves

Joachim Kneis · Alexander Langer · Peter Rossmanith

Received: date / Accepted: date

Abstract We present an algorithm that finds out-trees and out-branchings with at least k leaves in directed graphs. These problems are known as Directed Maximum Leaf Out-Tree and Directed Maximum Leaf Out-Branching, respectively, and—in the case of undirected graphs—as Maximum Leaf Spanning Tree. The run time of our algorithm is $O(4^k nm)$ on directed graphs and $O(poly(n) + 4^k k^2)$ on undirected graphs. This improves over the previously fastest algorithms for these problems with run times of $2^{O(k \log k)} poly(n)$ and $O(poly(n) + 6.75^k poly(k))$ respectively.

Keywords Graph algorithms · Algorithms

CR Subject Classification G.2.2 [Graph Theory]: Graph algorithms; F.1.3 [Complexity Measures and Classes]: Reducibility and completeness

1 Introduction

In this paper we consider the problems of finding trees and spanning trees in graphs, so that their number of leaves is maximal. To be more precise, given a graph or digraph G with n vertices and a number k, we must find out whether G contains a subtree, respectively a spanning tree, with at least k leaves.

For undirected graphs, the terms *tree* and *spanning tree* are well-known. There are, however, a couple of variants of how these terms can be translated to directed graphs. Here, we consider *out-tree*s and *out-branching*s: An out-tree is a rooted tree, such that every leaf can be reached from the root via a directed path within this tree, and an out-branching is an out-tree that spans all vertices (see, e.g., [4]).

Supported by the DFG under grant RO 927/7

Theoretical Computer Science, Dept. of Computer Science, RWTH Aachen University, Ahornstrasse 55, 52074 Aachen, Germany
E-mail: {kneis,langer,rossmani}@cs.rwth-aachen.de

Here, we consider the problem of finding an out-tree or and out-branching with at least k leaves in a given digraph in the realm of parameterized algorithms. Formally, a parameterized problem is a pair (L, κ), where $L \subseteq \Sigma^*$ is a language over a finite alphabet Σ and $\kappa\colon \Sigma^* \to \mathbf{N}$ is the parameterization. For an instance $x \in \Sigma^*$, we call $\kappa(x)$ the parameter of x. A parameterized problem (L, κ) is called *fixed parameter tractable* and said to be in the complexity class FPT if there is an algorithm that for all $x \in \Sigma^*$ decides whether $x \in L$ in time $f(\kappa(x))poly(|x|)$, where f is an arbitrary computable function. For further information, see the books by Downey and Fellows [17], by Flum and Grohe [23], or by Niedermeier [33].

Undirected graphs

The NP-hard [26] Maximum Leaf Spanning Tree problem has a number of practical applications. It is, for instance, used in network design, where a small number of (typically expensive) core routers provide the backbone network infrastructure for a large number of clients, see, e.g., [13, 31, 34, 37]. It is known to be APX-hard [25] and there exists a 2-approximation algorithm [36] running in linear time. On cubic graphs, a 3/2-approximation was found recently [11]. Fomin, Grandoni, and Kratsch [24] showed how to solve the Maximum Leaf Spanning Tree problem in time $O(1.9407^n)$. The parameterized version is defined as follows:

Maximum Leaf Spanning Tree (MLST)

Input: An undirected graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does G have a spanning tree with at least k leaves?

Fellows and Langston [20] observed that MLST $\in$ FPT using the Graph Minor Theory [35]. The first explicit algorithm is due to Bodlaender [5] using dynamic programming on tree-decompositions, which yields a linear time algorithm for MLST for any fixed k. Consequent improvements are due to Downey and Fellows [16] (running time bounded by $O(n + (2k)^{4k})$) and Fellows, McCartin, Rosamond, and Stege [21] ($O(n+14.23^k k)$). Results in extremal graph theory by Linial and Sturtevant [32] and Kleitman and West [29] were used by Bonsma, Brueggemann, and Woeginger [8] for an algorithm with a run time of $O(n^3+9.4815^k k^3)$. After several further rounds of improvement by Estivill-Castro, Fellows, Langston, and Rosamond [19] as well as Bonsma [6, 7], the previously best known algorithm for this problem is by Bonsma and Zickfeld [12] and runs in time $O(poly(n) + 6.75^k poly(k))$.

There is also a *small problem kernel* for this problem: In polynomial time an instance (G, k) of MLST is reduced to an equivalent instance (G', k') with $|G'| \leq f(k)$ and $k' \leq k$. Note that the existence of a small problem kernel for a parameterized problem implies that the respective problem is in FPT. The MLST problem admits a kernel of size $3.75k$ as shown by Estivill-Castro, Fellows, Langston, and Rosamond [19].

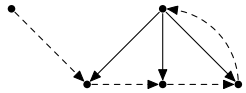

Fig. 1 A graph containing a 3-leaf out-tree, but no 3-leaf out-branching.

Directed Graphs

While it is easy to see that a k-leaf tree in an undirected graph can always be extended to a k-leaf spanning tree (see also Lemma 1), this is not the case for directed graphs that are not strongly connected (see the example depicted in Figure 1). In the directed case, we therefore have to distinguish between the following two variants:

Directed Maximum Leaf Out-Tree (DMLOT)

Input: A digraph $G = (V, A)$, a positive integer k
Parameter: k
Question: Does G contain an out-tree with at least k leaves?

Directed Maximum Leaf Spanning Out-Branching (DMLOB)

Input: A digraph $G = (V, A)$, a positive integer k
Parameter: k
Question: Does G have an out-branching with at least k leaves?

Since every undirected graph can be seen as a symmetric directed graph, these variants are NP- and APX-hard as well. DMLOB even remains NP-hard when restricted to acyclic digraphs by a reduction from the Set Cover problem [3], but can be approximated with an approximation factor of $O(\sqrt{\mathrm{OPT}})$ due to Drescher and Vetta [18], where OPT is the size of an optimal solution.

However, nothing was known about membership in FPT for a long time, since neither the graph minor theorem by Robertson and Seymour in its current shape, nor the method used by Bodlaender, nor the extremal results for undirected graphs are applicable for directed graphs. Alon, Fomin, Gutin, Krivelevich, and Saurabh [1, 2] (see also [3]) initiated the current line of research on maximum leaf problems for directed graphs. They proved an extremal result similar to the bound on undirected graphs by Kleitman and West: Every strongly connected directed graph with minimum in-degree 3 has an out-branching with at least $(n/4)^{1/3} - 1$ leaves. They furthermore showed that either a k-leaf out-tree exists, or the pathwidth of the underlying (undirected) graph is bounded by $2k^2$. This allows dynamic programming, so that an overall run time bound of $2^{O(k^2 \log k)} poly(n)$ for the DMLOT problem can be achieved, answering the long open question whether DMLOT is fixed parameter tractable. They could further improve this to $2^{O(k \log^2 k)} poly(n)$ and, if G is acyclic, to $2^{O(k \log k)} poly(n)$ [1]. Bonsma and Dorn [9] showed that DMLOB $\in$ FPT by extending the technique of Alon et al. Their approach is based on pathwidth and dynamic programming as well and yields a run time bound of $2^{O(k^3 \log k)} poly(n)$. In a subsequent paper [10], they proved that a run time of $2^{O(k \log k)} poly(n)$ suffices to solve both, DMLOT and DMLOB.

Our contribution

Recall that in the directed case a k-leaf out-tree cannot necessarily be extended to a k-leaf out-branching even if G does contain an out-branching (see Figure 1). In this paper, we use the fact that a k-leaf out-tree with root r *can* always be extended to a k-leaf out-branching if G does contain an out-branching rooted at r. This particularly holds true for undirected graphs, which can be seen as symmetric directed graphs.

We develop a new algorithm that — in contrast to the prior approaches based on extremal graph theory — grows an out-tree from the root and therefore solves DMLOT, DMLOB, and MLST. The algorithm recursively selects and tries two of the many possible ways to extend the tree. We prove that at least one of these recursive calls finds a k-leaf out-tree, if such a tree exists. The number of recursive calls can be bounded by $2^{2k} = 4^k$.

Organization

This paper is organized as follows: In Section 2, we introduce some notations. Section 3 contains the proof that each out-tree rooted at r can be extended to an out-branching rooted at r if such an out-branching exists. Our algorithm and a detailed analysis are presented in Section 4. In Section 5, we discuss subexponential lower bounds. A conclusion and some details on consequent research can be found in the final section.

2 Preliminaries

Let $G = (V, A)$ be a digraph, and let $n := |V|$ and $m := |A|$ be the number of its vertices and arcs, respectively. For $v \in V$, let $N^+(v) := \{\, u \in V \mid (v, u) \in A \,\}$ denote the set of all out-neighbors of v in G, $N^+[v] := N^+(v) \cup \{v\}$ the closed out-neighborhood of v, and for any $U \subseteq V$ we let $N^+(U) := \bigcup_{u \in U} N^+(u)$.

A rooted *out-tree* T is a tree in G, such that T has a root r, and each vertex in T can be reached by a unique directed path from r in T. A *k-leaf out-tree* is an out-tree with at least k leaves, and a *k-leaf out-branching* is a k-leaf out-tree that is also spanning all vertices, i.e., one that contains all vertices.

Let T be an out-tree in G. Then $V(T)$ denotes the set of vertices of T, and $A(T)$ the set of arcs of T. The root, leaves, and inner vertices of T are denoted by $\mathrm{root}(T)$, $\mathrm{leaves}(T)$ and $\mathrm{inner}(T) := V(T) \setminus \mathrm{leaves}(T)$, respectively.

For $v \in V(T)$, we let $N^+_{\overline{T}}(v) := N^+(v) \setminus V(T) = \{\, u \in V(G) \setminus V(T) \mid (v, u) \in A(G) \,\}$, i.e., all neighbors of v in G that are not already contained in T. Similarly, we let $A^+_{\overline{T}}(v) := \{\, (v, u) \mid u \in N^+_{\overline{T}}(v) \,\}$ and $N^+_{\overline{T}}(U) := N^+(U) \setminus V(T)$ for any $U \subseteq V$. For a set of arcs $F \subseteq A^+_{\overline{T}}(v)$, we let $T + F$ be the tree obtained from T by adding all arcs in F. For a single arc $e \in N^+_{\overline{T}}(v)$, we abbreviate

$T+\{e\}$ as $T+e$. Finally, for any $v \in V$, let $T_v := (N^+[v], \{ (v,u) \mid u \in N^+(v) \})$ be the star rooted at v that contains all out-neighbors of v in G.

Since undirected graphs can be seen as symmetric directed graphs, where every edge has a *reverse edge*, we only consider directed graphs unless explicitly stated. In the case of undirected graphs, we use $N(v)$ and $N_{\overline{T}}(v)$ instead of $N^+(v)$ and $N^+_{\overline{T}}(v)$. Also, out-trees and out-branchings in undirected graphs are denoted by *tree*s and *spanning tree*s.

Recall that our algorithm recursively grows an out-tree from the root until either enough leaves have been found or the out-tree cannot be extended further.

Definition 1 For out-trees $T \neq T'$, we say T' *extends* T, denoted by $T' \succeq T$, iff $\mathrm{root}(T') = \mathrm{root}(T)$ and T is an induced subgraph of T'. We write $T' \succ T$ when $T' \succeq T$ and $T' \neq T'$.

There might be vertices that have to be leaves in every k-leaf out-tree T' with $T' \succeq T$. Therefore, the algorithm further distinguishes between *red* leaves R of T that will be leaves in the final k-leaf out-tree T', and *blue* leaves B that are still allowed to become inner vertices in the future.

Definition 2 A *leaf-labeled* out-tree is a 3-tuple (T, R, B), such that T is an out-tree, $R \cup B = \mathrm{leaves}(T)$ and $R \cap B = \emptyset$.

We then only consider such extensions of T that respect our previous fixing of red leaves.

Definition 3 If (T, R, B) is a leaf-labeled out-tree and T' is an out-tree such that $T' \succeq T$ and $R \subseteq \mathrm{leaves}(T')$ (red leaves of T remain leaves in T'), we say T' is a *(leaf-preserving) extension* of (T, R, B), denoted by $T' \succeq (T, R, B)$. If furthermore $T' \succ T$, we write $T' \succ (T, R, B)$.

A leaf-labeled out-tree (T', R', B') *extends* a leaf-labeled out-tree (T, R, B), denoted by $(T', R', B') \succeq (T, R, B)$, iff $T' \succeq (T, R, B)$ and $R' \supseteq R$.

If $(T', R', B') \succeq (T, R, B)$ and additionally $T' \succ T$ or $R' \neq R$ or $B' \neq B$ holds, we write $(T', R', B') \succ (T, R, B)$.

In any recursive call, each inner vertex of the current out-tree T has all of its neighbors in $V(T)$. We call such out-trees inner-maximal.

Definition 4 Let $G = (V, A)$ be a digraph, and let T be an out-tree in G. We call T an *inner-maximal* out-tree if $N^+(\mathrm{inner}(T)) \subseteq V(T)$. A leaf-labeled out-tree (T, R, B) is called an *inner-maximal leaf-labeled out-tree*, if T is inner-maximal.

Note that if an out-tree T is inner-maximal, we can only extend it by choosing a leaf $v \in \mathrm{leaves}(T)$ with $N^+_{\overline{T}}(v) \neq \emptyset$ and letting $T' := T + F$ for some $\emptyset \neq F \subseteq A^+_{\overline{T}}(v)$. Moreover, the new T' is inner-maximal if and only if $T' = T + A^+_{\overline{T}}(v)$. In Section 4, we show that it suffices to consider only inner-maximal trees.

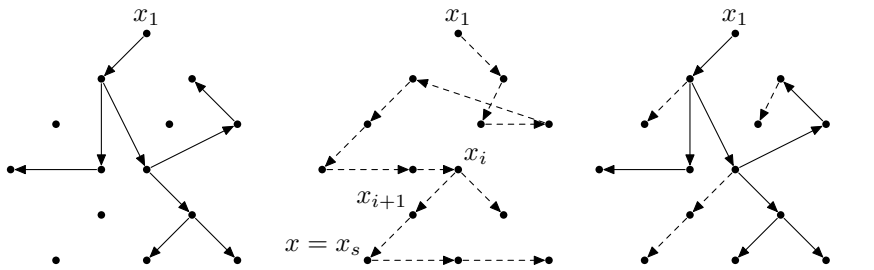

Fig. 2 How to extend a k-leaf out-tree into a k-leaf out-branching: For the ease of illustration, we do not show all the arcs in G. A 4-leaf out-tree with root x_1 is depicted in the first figure. The second figure shows an arbitrary out-branching rooted at x_1; we have chosen one with two leaves. We can extend the first out-tree with arcs from the out-branching so that all vertices are reached.

3 k-Leaf Out-Trees versus k-Leaf Out-Branchings

In this section, we show when and how k-leaf out-trees can be extended to k-leaf out-branchings. Note here that we allow that the resulting out-branching has more leaves than the originating k-leaf out-tree. While special case for undirected graphs, Lemma 1, is rather simple and can be considered folklore, Lemma 2 significantly eases our search for k-leaf out-branchings in directed graphs.

Lemma 1 *A connected, undirected graph $G = (V, E)$ contains a k-leaf tree iff G contains a k-leaf spanning tree. Furthermore, each k-leaf tree can be extended to a k-leaf spanning tree in time $O(m)$.*

Proof Let T be a tree in G with at least k leaves, and let $l := |V - V(T)|$ be the number of vertices that are not part of T. If $l = 0$, then T is a spanning tree with at least k leaves. If otherwise $l > 0$, choose $u \in V(T)$ and $v \notin V(T)$, such that u and v are adjacent. Let $T' := T + \{u, v\}$. It is easy to see that T' has at least as many leaves as T. Furthermore, this operation can efficiently be done with a breadth-first-search on G starting in $V(T)$, and hence after at most $O(n+m)$ steps a spanning tree with at least k leaves can be constructed from T. □

In the undirected case, it is therefore sufficient to search for an arbitrary tree with at least k leaves.

Lemma 1 is, however, not applicable for directed graphs (cf., Figure 1): It is easy to see that this digraph contains an out-tree with three leaves, but the unique out-branching contains only one leaf. If we fix the root of the trees, we obtain the following weaker result for directed graphs.

Lemma 2 *Let $G = (V, A)$ be a digraph. If G contains an out-branching rooted at x_1, then any k-leaf out-tree rooted at x_1 can be extended to a k-leaf out-branching of G in time $O(m)$.*

Algorithm 1 A fast algorithm for maximum leaf problems.

Algorithm MaxLeaf(G, T, R, B, k):
Input: Digraph $G = (V, A)$, an inner-maximal leaf-labeled out-tree (T, R, B), $k \in \mathbf{N}$
Output: Is there a k-leaf out-tree $T' \succeq (T, R, B)$?

```
01: if |R| + |B| ≥ k then return "yes"
02: if B = ∅ then return "no"
03: Choose u ∈ B.
// Try branch where u is a leaf
04: if MaxLeaf(G, T, R ∪ {u}, B \ {u}, k) then return "yes"
// If u is not a leaf, it must be an inner vertex in all extending solutions
05: B := B \ {u}
06: N := N_T^+(u)
07: T := T + { (u, u') | u' ∈ N }
// Follow paths, see Lemma 5
08: while |N| = 1 do
09:     Let u be the unique element of N
10:     N := N_T^+(u)
11:     T := T + { (u, u') | u' ∈ N }
12: done
// Do not branch if no out-neighbors left, see Observation 1
13: if N = ∅ then return "no"
14: return MaxLeaf(G, T, R, B ∪ N, k)
```

Proof Let T be a k-leaf out-tree with $\mathrm{root}(T) = x_1$ and let $l := |V - V(T)|$ be the number of vertices that are not in T. If $l = 0$, then T is an out-branching for G with at least k leaves. If $l > 0$, choose $x \in V - V(T)$ and consider a path $x_1, x_2, \ldots, x_s$ with $x_s = x$ from x_1 to x. Since G has an out-branching rooted at x_1, such a path must exist in G. Furthermore, $x \notin V(T)$ and hence there is $1 \leq i \leq s$ such that $x_i \in V(T)$ and $x_j \notin V(T)$ for each $j = i + 1, \ldots, s$. It is easy to see that by adding the path $x_i, \ldots, x_s$ to T, the number of leaves does not decrease. Repeating this procedure yields an out-branching for G that has at least k leaves. Again, this can be efficiently done with a breadth-first-search on G, which starts in T and takes time at most $O(n + m)$. See Figure 2 for an illustration. □

4 The Algorithm

In this section, we give Algorithm MaxLeaf(G, T, R, B, k), which given an inner-maximal leaf-labeled out-tree (T, R, B) recursively decides whether there is a k-leaf out-tree $T' \succeq (T, R, B)$ in G. Informally, the algorithm works as follows: Choose a blue vertex $u \in B$ and recursively test whether there is a solution where u is a leaf, or whether there is a solution where u is an inner vertex. In the first case, u is recolored red, so that u is preserved as a leaf in solutions T'. In the second case, u is considered an inner vertex, all of its outgoing arcs $A_T^+(u)$ are added to T and the vertices in $N_T^+(u)$ are colored blue.

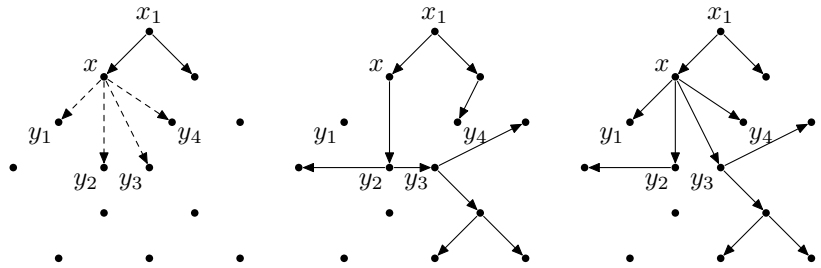

Fig. 3 The exchange argument (Lemma 4): The first figure shows a leaf-labeled out-tree (T, R, B) with $x \in B$. The out-neighborhood of x, $N^+_{\overline{T}}(x)$, is shown with dashed arcs. The second figure shows a 5-leaf out-tree $T' \succ (T, R, B)$, but different choices for arcs originating in x have been made: y_1 is not in T' at all, and different paths to y_3 and y_4, respectively, have been chosen. The third figure shows how the T' can be modified so that *all* $y \in N^+_{\overline{T}}(x)$ are children of x. This modification does not decrease the number of leaves in T': y_1 becomes a new leaf; no changes are made to the arc (x, y_2), y_3 remains inner vertex, and y_4 remains leaf, although it is now connected through x.

Lemma 4 guarantees that at least one of these two branches is successful, if a solution exists at all. In the special case that $|N^+_{\overline{T}}(u)| \leq 1$, the correctness of the algorithm follows from Lemma 5 and Observation 1. The algorithm can stop once no blue vertices are left, as shown by Lemma 3.

Lemma 3 *Let (T, R, B) be an inner-maximal leaf-labeled out-tree, and $T' \succ (T, R, B)$ a leaf-preserving extension of (T, R, B). Then $B \neq \emptyset$.*

Proof Since $T \neq T'$, there is an $x \in V(T')$ with $x \notin V(T)$. Let $x_1 := \text{root}(T) = \text{root}(T')$ and consider the path $x_1, \ldots, x_l$ with $x_l = x$ from x_1 to x in T. Since $x = x_l \notin V(T)$, there is some i such that $x_i \in V(T)$ and $x_{i+1} \notin V(T)$. We know that $x_i \in \text{leaves}(T) = R \cup B$, because T is inner-maximal. On the other hand, $x_i \in \text{inner}(T')$, and with $R \subseteq \text{leaves}(T')$, we have $x_i \in B$. □

The following lemma allows for a simple branching since for every leaf $x \in B$ we can just test the two cases whether x is a leaf or is an inner vertex. The latter case then implies that we have to enlarge the out-tree by $A^+_{\overline{T}}(x)$.

Lemma 4 *Let $G = (V, A)$ be a digraph, (T, R, B) a leaf-labeled out-tree, and $x \in B$.*

1. *If there is no k-leaf out-tree T', such that $T' \succeq (T, R, B)$ and $x \in \text{leaves}(T')$, then* all *k-leaf out-trees T' with $T' \succeq (T, R, B)$ have $x \in \text{inner}(T')$.*
2. *If there is a k-leaf out-tree T', such that $T' \succeq (T, R, B)$ and $x \in \text{inner}(T')$, then there is also a k-leaf out-tree $T'' \succeq (T + A^+_{\overline{T}}(x), R, N^+_{\overline{T}}(x) \cup B \setminus \{x\})$.*

Proof 1. Let T' be an out-tree such that $T' \succeq (T, R, B)$. Then T is an induced subgraph of T' and hence x is either a leaf or an inner vertex in T'.

2. Let T' be a k-leaf out-tree, such that $T' \succeq (T, R, B)$ and $x \in \text{inner}(T')$, i.e., $N^+_{\overline{T}}(x) \neq \emptyset$. Consider an arbitrary $y \in N^+_{\overline{T}}(x)$. If $y \notin V(T')$, then we can construct a k-leaf out-tree T'' from T' by adding the arc (x, y). If otherwise

$y \in V(T')$, but $(x, y) \notin A(T')$, consider the unique path $x_1, x_2, \ldots, x_i, y$ from $x_1 := \text{root}(T')$ to y in T'. We construct T'' by replacing the arc (x_i, y) with (x, y) in T'. Then, $|\text{leaves}(T')| \leq |\text{leaves}(T'')|$: x is inner vertex in T' by definition, and $y \in \text{leaves}(T')$ implies $y \in \text{leaves}(T'')$. Furthermore, the connectivity of T' remains intact.
Doing so iteratively for all neighbors y of x yields a k-leaf out-tree T'' with $A^+_{\overline{T}}(x) \subseteq A(T'')$. Therefore $T'' \succeq (T + A^+_{\overline{T}}(x), R, N^+_{\overline{T}}(x) \cup B \setminus \{x\})$. See Figure 3 for an example.

□

Now let T be an inner-maximal leaf-labeled out-tree that can be extended to a k-leaf out-tree. Whenever we end up with a blue vertex x in T that has exactly one out-neighbor $y \in N^+_{\overline{T}}(x)$, we do not need to branch on both x and y: If we already know that x cannot be a leaf in any out-tree with k leaves, we also know that y cannot be a leaf in such an out-tree. Otherwise we could obtain a k-leaf out-tree by removing y, so that x becomes a leaf. The following lemma gives a formal formulation of this observation, which allows the algorithm to *follow* all paths it encounters.

Lemma 5 *Let $G = (V, A)$ be a digraph, (T, R, B) a leaf-labeled out-tree and $x \in B$ with $N^+_{\overline{T}}(x) = \{y\}$. If there is no k-leaf out-tree that extends $(T, R \cup \{x\}, B \setminus \{x\})$, then there is no k-leaf out-tree that extends $(T + (x, y), R \cup \{y\}, B \setminus \{x\})$.*

Proof Let T' be a k-leaf out-tree that extends $(T + (x, y), R \cup \{y\}, B \setminus \{x\})$. Since T is an induced subgraph of T' and $N^+_{\overline{T}}(x) = \{y\}$, x has exactly the child y in T'. Furthermore, $y \in \text{leaves}(T')$ by the choice of T'.

Let T'' be the tree obtained from T' by removing y. Then T'' extends (T, R, B) because $R \subseteq \text{leaves}(T') \subseteq \text{leaves}(T'') \cup \{y\}$ by the choice of T' and $y \notin R$, since $R \subseteq \text{leaves}(T) \subseteq V(T)$ and of course $y \notin V(T)$. Moreover, x becomes a leaf in T''. Therefore, $T'' \succeq (T, R \cup \{x\}, B \setminus \{x\})$. □

Similarly, any blue vertex x with an empty $N^+_{\overline{T}}(x)$ can be assumed to be a leaf.

Observation 1 Let $G = (V, A)$ be a digraph, (T, R, B) a leaf-labeled out-tree and $x \in B$ with $N^+_{\overline{T}}(x) = \emptyset$. If there is a k-leaf out-tree that extends (T, R, B), then there is a k-leaf out-tree that extends $(T, R \cup \{x\}, B \setminus \{x\})$.

We are now able to prove the correctness of Algorithm MaxLeaf.

Lemma 6 *Let $G = (V, A)$ be a digraph, $r \in V$, $|N^+(r)| > 1$ and $k \in \mathbf{N}$. Algorithm* MaxLeaf *returns "yes" when called on* MaxLeaf$(G, T_r, \emptyset, N^+(r), k)$ *if and only if G does contain a k-leaf out-tree rooted at r.*

Proof We first show that all subsequent calls to MaxLeaf are always given an inner-maximal leaf-labeled out-tree: The star T_r is inner-maximal, and hence $(T_r, \emptyset, N^+(r))$ is an inner-maximal leaf-labeled out-tree.

Let (T, R, B) be the inner-maximal out-tree given as argument to MaxLeaf. The algorithm chooses $x \in B$ and either fixes it as a leaf or as an inner vertex. If x becomes a leaf, then $(T, R \cup \{x\}, B \setminus \{x\}) \succ (T, R, B)$ is inner-maximal. If otherwise x becomes an inner vertex, an out-tree T' is obtained from T by adding $A^+_{\overline{T}}(x)$ and therefore particularly by adding $N^+_{\overline{T}}(x)$. Since $N(x) \subseteq V(T')$ and

$$N(\text{inner}(T')) = N(\text{inner}(T)) \cup N(x) \subseteq V(T) \cup N(x) = V(T'),$$

the new out-tree T' is inner-maximal, and so is $(T', R, N^+_{\overline{T}}(x) \cup B \setminus \{x\})$. This step might be repeated l times while $|N^+_{\overline{T}}(x)| = 1$, so that we obtain a sequence of leaf-labeled out-trees

$$(T, R, B) \prec (T', R', B') \prec \cdots \prec (T^{(l+1)}, R^{(l+1)}, B^{(l+1)}),$$

each of them being inner-maximal for the same reason. Therefore, MaxLeaf is called with an inner-maximal leaf-labeled out-tree $(T^{(l+1)}, R^{(l+1)}, B^{(l+1)})$. Thus, all recursive calls are valid.

By induction over $\prec$ we now prove the following claim:

If (T, R, B) is an inner-maximal leaf-labeled out-tree, MaxLeaf(G, T, R, B, k) answers correctly whether a k-leaf out-tree T' with $T' \succeq (T, R, B)$ exists.

For the induction basis, we consider the cases where the algorithm is called with arguments (T, R, B) such that $|R| + |B| \geq k$ or $B = \emptyset$. In the former case, T is already a k-leaf out-tree and the algorithm correctly returns "yes", and in the latter case "no" is the correct answer by Lemma 3.

Now let the algorithm be called with arguments (T, R, B), such that $|R| + |B| < k$ and $B \neq \emptyset$, and assume the claim already holds for all (T', R', B') with $(T, R, B) \prec (T', R', B')$. Let $u \in B$.

Since $|R|+|B| < k$, the algorithm only answers "yes" if one of the recursive calls returns "yes". Hence, by induction "no" is returned when there is no k-leaf out-tree extending (T, R, B).

We therefore now assume there is a k-leaf out-tree extending (T, R, B). If there is a k-leaf out-tree T' with

$$T' \succeq (T, R \cup \{u\}, B \setminus \{u\}) \succ (T, R, B),$$

then the recursive call in line 4 correctly returns "yes" by induction.

If there is no such out-tree, then by Lemma 4 there is a k-leaf out-tree $T'' \succeq T'$, where $T' := (T + A^+_{\overline{T}}(u), R, N^+_{\overline{T}}(u) \cup B \setminus \{u\})$. If $|N^+_{\overline{T}}(u)| \geq 2$,

$$T'' \succeq T' \succ (T, R, B)$$

and the recursive call MaxLeaf(G, T', k) in Line 14 correctly returns "yes" by induction, since the while-loop is not entered.

If $|N^+_{\overline{T}}(u)| = 1$, the while-loop is executed at least once. Let $l \geq 1$ denote the number of iterations. The jth iteration of the while-loop implicitly defines a new inner-maximal leaf-labeled out-tree (T_j, R_j, B_j) for $1 \leq j \leq l$, where $v_0 = u$, $(T_0, R_0, B_0) = (T, R, B)$ and for all $0 \leq i \leq l - 1$,

1. $(T_{i+1}, R_{i+1}, B_{i+1}) = (T_i + (v_i, v_{i+1}), R_i, N^+_{\overline{T_i}}(v_i) \cup B_i \setminus \{v_i\})$,
2. $N^+_{\overline{T_i}}(v_i) = \{v_{i+1}\}$, and
3. $|N^+_{\overline{T_l}}(v_l)| \neq 1$.

Note that the algorithm does not update the set B in each step, as every change would be overwritten in the next step of the while-loop. It is sufficient to use an updated B in the recursive call after the loop. Since $T'' \succ (T, R, B)$ and $T'' \not\succeq (T, R \cup \{u\}, B \setminus \{u\})$, Lemma 4 guarantees

$$T'' \succ (T + (v_0, v_1), R, \{v_1\} \cup B_0 \setminus \{u\}) = (T_1, R_1, B_1)$$

and by Lemma 5, $T'' \not\succeq (T_1, R_1 \cup \{v_1\}, B_1 \setminus \{v_1\})$. Thus, we obtain $T'' \succeq (T_l, R_l, B_l) \succ (T, R, B)$ inductively.

Moreover, $N^+_{\overline{T_l}}(v_l) \neq \emptyset$, because otherwise $T'' \succ (T_l, R_l \cup \{v_l\}, B_l \setminus \{v_l\})$ by Observation 1 and then $T' \succeq (T, R \cup \{u\}, B \setminus \{u\})$ again inductively by Lemma 5, a contradiction. Therefore, the algorithm does not return "no" in Line 13.

Hence the algorithm recursively calls itself as MaxLeaf(G, T_l, R_l, B_l, k), where $T'' \succeq (T_l, R_l, B_l) \succ (T, R, B)$. By the induction hypothesis for (T_l, R_l, B_l), the algorithm correctly returns "yes", which concludes the proof of the claim.

Now let T be a k-leaf out-tree T with root$(T) = r$ and consider $T' = (\{r\}, \emptyset)$. Then $(T', \emptyset, \{r\}) \prec T$ is a inner-maximal leaf-labeled out-tree, and by Lemma 4 there is also a k-leaf out-tree $T'' \succ (T_r, \emptyset, N^+(r))$. The correctness of Lemma 6 now follows from the claim above. □

Lemma 7 *Let $G = (V, A)$ be a digraph and $v \in V$. The number of recursive calls of Algorithm* MaxLeaf *when called as* MaxLeaf$(G, T_v, \emptyset, N^+(v), k)$ *for $v \in V$ is bounded by $O(2^{2k-|N^+(v)|}) = O(4^k)$.*

Proof Consider a potential function $\Phi(k, R, B) := 2k - 2|R| - |B|$. When called with a leaf-labeled out-tree (T, R, B), the algorithm recursively calls itself at most two times: In Line 4, some vertex $u \in B$ is fixed as a leaf and the algorithm calls itself as MaxLeaf$(G, T, R \cup \{u\}, B \setminus \{u\}, k)$. The potential decreases by $\Phi(k, R, B) - \Phi(k, R \cup \{u\}, B \setminus \{u\}) = 1$. The while loop in Lines 8–12 does not change the size of B. If, however, Line 14 of the algorithm is reached, we have $|N| \geq 2$. Here, the recursive call is MaxLeaf$(G, T', R, B \setminus \{u\} \cup N, k)$ for some out-tree T', and hence the potential decreases by $\Phi(k, R, B) - \Phi(k, R, B \setminus \{u\} \cup N) \geq 1$.

Note that $\Phi(k, R, B) \leq 0$ implies $|R + B| \geq k$. Since the potential decreases by at least 1 in each recursive call, the height of the search tree is therefore at most $\Phi(k, R, B) \leq 2k$. For arbitrary inner-maximal leaf-labeled out-trees (T, R, B), the number of recursive calls is hence bounded by $2^{\Phi(k,R,B)}$.

In the very first call, we already have $|B| = |N^+(v)|$. Hence we obtain a bound of $2^{\Phi(k,\emptyset,N^+(v))} = O(2^{2k-|N^+(v)|}) = O(4^k)$. □

Theorem 1 MLST *can be solved in time* $O(poly(n) + 4^k \cdot k^2)$.

Proof Let $G = (V, E)$ be an undirected graph. As Estivill-Castro et al. have shown [19], there is a problem kernel of size $3.75k = O(k)$ for MLST, which can be computed in a preprocessing that requires time $poly(n)$. Hence, $n = O(k)$.

Without loss of generality, we assume G is connected and $k > 2$ (the cases where $k \leq 2$ are trivial, but the algorithm fails on a single edge for $k = 2$). We do not know, which vertex $v \in V$ suffices as a root, so we need to iterate over possible roots. This already yields an algorithm with running time $O(poly(n) + 4^k poly(k))$ by Lemma 7. The following more precise analysis shows how to bound the run time by $O(poly(n) + 4^k \cdot k^2)$.

Since $k > 2$, it is easy to see that either some $v \in V$ or one of its neighbors is root of some k-leaf spanning tree, if any k-leaf spanning tree T exists at all: If $v \in \text{leaves}(T)$, the unique predecessor u of v in T is an inner vertex $u \in \text{inner}(T)$. If furthermore v has minimum degree, the cost to test all $u \in N(v) \cup \{v\}$ disappears in the run time estimation, as will be shown in the next paragraph.

Therefore, let $v \in V$ be a vertex of minimum degree d. We need to call MaxLeaf with arguments $(G, T_u, R, N(u), k)$ for all $u \in N(v) \cup \{v\}$: If G contains a k-leaf tree, at least one of those u is a root of some k-leaf tree and the respective call to MaxLeaf returns "yes" by Lemma 6. Otherwise each call returns "no". By Lemma 7, the total number of recursive calls is bounded by

$$O(2^{\Phi(k,\emptyset,N(v))}) + \sum_{u \in N(v)} O(2^{\Phi(k,\emptyset,N(u))}) = O\big((d+1)2^{2k-d}\big) = O\Big(4^k \frac{d+1}{2^d}\Big).$$

It remains to show that the number of operations in each recursive call is bounded by $O(n^2) = O(k^2)$. We can assume the sets V, E, $V(T)$, $E(T)$, R, and B are realized as doubly-linked lists with additional pointers from each vertex to its position in the respective lists, so that membership tests and insert and delete operations only require constant time each.

Hence Lines 1–3 and computing the new sets in Lines 4 and 5 takes constant time. Computing $N_{\overline{T}}(u)$ and the new tree T takes time $O(k)$, since u has only up to k neighbors, which are tested for membership in $V(T)$ in constant time. The while loop is executed at most once per vertex $u \in V$. Each execution of the while loop takes time $O(k)$ due to the computation of $N_{\overline{T}}(u)$. Concatenating N to B in Line 14 takes constant time, but updating the B-membership flag for each $v \in N$ takes up to k steps.

At this point we have shown that the overall number of operations required to decide whether G contains a rooted k-leaf tree is bounded by $O(poly(n) + 4^k \cdot k^2)$. By Lemma 1, each k-leaf tree can be extended to a spanning tree with at least k leaves, so the decision problem MLST can be solved in the same amount of time. □

Note that Algorithm MaxLeaf can easily be modified to return a k-leaf spanning tree in G within the same run time bound. In this case, an additional

$O(m)$ postprocessing is required to extend the k-leaf tree to a k-leaf spanning tree.

Theorem 2 DMLOT *and* DMLOB *can be solved in time* $O(4^k nm)$.

Proof Let $G = (V, A)$ be a digraph. We first consider DMLOT: If G contains a k-leaf out-tree rooted at r, MaxLeaf$(G, T_r, \emptyset, N^+(r), k)$ returns "yes" by Lemma 6. Otherwise, MaxLeaf$(G, T_v, \emptyset, N^+(v), k)$ returns "no" for all $v \in V$. We do not know r, so we need to iterate over all $v \in V$. By Lemma 7, the total number of recursive calls is therefore bounded by

$$\sum_{v \in V} O(2^{\Phi(k, \emptyset, N^+(v))}) = O(n \cdot 2^{2k}) = O(4^k n).$$

In a single recursive call to MaxLeaf, each edge is considered at most once when computing the respective sets $N^+_{\overline{T}}(u)$ in lines 6 and 10. Hence the time spent in the while loop is at most $O(m)$. Therefore, the overall run time to solve DMLOT is bounded by $O(4^k \cdot nm)$.

To prove the run time bound for DMLOB, the algorithm must be slightly modified in Line 1. Here, it may only return "yes" if the leaf-labeled out-tree (T, R, B) can be extended to a k-leaf out-branching. By Lemma 2, each k-leaf out-tree that shares the same root with some out-branching *can* be extended to a k-leaf out-branching in time $O(m)$. Thus the run time remains bounded by $O(4^k \cdot nm)$. □

5 Subexponential Lower Bounds

It furthermore turns out that $2^{O(k)}poly(n)$ algorithms for the problems studied in this paper are in some sense optimal, i.e., subexponential time algorithms are unlikely. More precisely, we show that the Exponential Time Hypothesis (ETH) fails if MLST can be solved in time $2^{o(k)}poly(n)$. The ETH is the well-known assumption

$$\text{3-SAT} \notin \text{DTIME}(2^{o(n)}),$$

where n denotes the number of variables, see, e.g., Impagliazzo, Paturi, and Zane [28] and Flum and Grohe [23]. Our proof is based on the following equivalent assumption, cf. [28] or [23, Corollary 16.23].

Lemma 8 (Impagliazzo et al. [28]) *The* Vertex Cover *problem cannot be solved in time* $2^{o(|V|+|E|)}$ *unless the ETH fails.*

We will use the following linear size reduction from Vertex Cover to MLST.

Lemma 9 *Let* (G, k) *with* $G = (V, E)$ *be an input instance for* Vertex Cover. *We can construct an equivalent instance* (G', k') *with* $G' = (V', E')$ *for* MLST *in polynomial time such that*

- $|V'| = O(|V| + |E|)$,
- $|E'| = O(|V| + |E|)$, *and*
- (G, k) *is a "yes"-instance iff* (G', k') *is a "yes"-instance.*

Proof Given the input instance (G, k) with $G = (V, E)$ for Vertex Cover, we construct an instance (G', k') for MLST as follows. Without loss of generality, assume $|V| > 2$. We let G' be the bipartite graph with vertices $V \cup E \cup \{x_1, x_2\}$, where $x_1, x_2 \notin V \cup E$. In G', there is an edge between $v \in V$ and $e \in E$ iff $v \in e$. Furthermore, G' contains the edge $\{x_1, x_2\}$ and edges $\{x_2, v\}$ for every $v \in V$. We set $k' = |V| + |E| - k + 1$.

Clearly, $|V'| = |V| + |E| + 2$ and $|E'| = 2|E| + 1 + |V|$. Furthermore, G has a vertex cover of size at most k if and only if G' has a spanning tree with at least k' leaves. Note that we search for a spanning tree where at most $k + 1$ nodes are *not* leaves. Since x_2 is an internal node in any spanning tree, the remaining k internal nodes of a spanning tree for G' will correspond to nodes of a vertex cover for G.

Let $U \subseteq V$ with $|U| \leq k$ be a vertex cover of G. We construct a spanning tree T for G' as follows: Firstly, T contains the edge $\{x_1, x_2\}$ and the edges $\{x_2, v\}$ for each $v \in V$. Secondly, for each $e = \{u, v\} \in E$, the tree T contains one of the edges $\{\, \{e, u'\} \in E' \mid u' \in e \cap U \,\}$, which is non-empty since U is a vertex cover for G. Therefore, T is a spanning tree for G', and x_1 and all nodes in E and $V \setminus U$ are leaves in T.

Conversely, let T be an optimal spanning tree for G'. First note that x_1 is a leaf and x_2 is an internal node in any spanning tree for G'. Secondly, we can assume that each node $e \in E$ is a leaf in T by the following simple exchange argument. Let $e = \{u, v\} \in \text{inner}(T)$. By construction, u and v are the only neighbors of e in G'. Let w.l.o.g. u be the unique node in $\{u, v\}$ that is contained in the unique path between x_2 and e in T. We can then obtain a new optimal spanning tree T' from T by replacing the edge $\{v, e\}$ with $\{x_2, v\}$. The tree T' is still connected, and $|\text{leaves}(T)| \leq |\text{leaves}(T')|$, since $x_2 \in \text{inner}(T)$. Let $U \subseteq \text{inner}(T) \cap V$. Then U is a vertex cover of size at most k for G: For each edge $e \in E$ we have $|e \cap U| \geq 1$, since T is a spanning tree. Furthermore, x_1 and all $e \in E$ are leaves in T, and x_2 is an inner node. This means, at least $|V| - k$ nodes in V are leaves in T, which then yields the bound for $|U|$. □

Theorem 3 MLST *cannot be solved in time* $2^{o(n+m)}$ *unless the ETH fails.*

Proof Assume that MLST can be solved in time $2^{o(n+m)}$. The linear size reduction from Lemma 9 then implies that we can also solve Vertex Cover in time $2^{o(n+m)}$. By Lemma 8, this implies the ETH fails. □

Since we can assume $k \leq |V|$ for all instances of MLST we obtain the following corollary.

Corollary 1 MLST *cannot be solved in time* $2^{o(k)}\text{poly}(n)$ *unless the ETH fails.*

Conclusion

We solve open problems [10, 27] on whether there exist $c^k poly(n)$-time algorithms for the k-leaf out-tree and k-leaf out-branching problems on directed graphs. Our algorithms for DMLOT and DMLOB have a run time of $O(4^k nm)$, which is a significant improvement over the previously best bound of $2^{O(k \log k)} poly(n)$. Unless the Exponential Time Hypothesis fails, this cannot be improved further to $2^{o(k)} poly(n)$ in the general case, but nothing is known about other graph classes such as planar (di-)graphs.

Since the undirected case is easier, has a linear size problem kernel, and the root of some k-leaf tree can be found faster, we can solve MLST in time $O(poly(n) + 4^k k^2)$, where $poly(n)$ is the time to compute the problem kernel of size $3.75k$. This improves over the currently best algorithm with a run time of $O(poly(n) + 6.75^k poly(k))$.

6 Consequent Research

We used problem kernels only for the MLST problem on undirected graphs. On directed graphs, no polynomial time computable problem kernel of size polynomial in k for the DMLOT and DMLOB problems is known. Very recently, Fernau, Fomin, Lokshtanov, Raible, Saurabh, and Villanger [22] showed that no such kernelization is possible unless the polynomial hierarchy collapses to the third level. If, however, we are looking for an out-branching with a given root, then a kernel of size $O(k^2)$ exists as shown by Daligault and Thomassé [15].

Recently, Daligault, Gutin, Kim, and Yeo improved our algorithm and the analysis and obtained a time complexity of $O(3.72^k poly(n))$ for the DMLOB problem [14], and Koutis and Williams [30] even found a $O^*(2^k)$ randomized algorithm for MLST.

Acknowledgements We would like to thank the anonymous referees for their valuable comments, which greatly helped to improve the presentation of this paper.

References

1. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Better algorithms and bounds for directed maximum leaf problems. In: Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), no. 4855 in Lecture Notes in Computer Science, pp. 316–327. Springer(2007)
2. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Parameterized algorithms for directed maximum leaf problems. In: Proceedings of the 34th International Colloquium on Automata, Languages, and Programming (ICALP), no. 4596 in Lecture Notes in Computer Science, pp. 352–362. Springer(2007)
3. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Spanning directed trees with many leaves. SIAM J. Discrete Math. **23**(1), 466–476 (2008)
4. Bang-Jensen, J., Gutin, G.Z.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer Monographs in Mathematics. Springer(2009)

5. Bodlaender, H.L.: On linear time minor tests with depth-first search. J. Algorithms **14**(1), 1–23 (1993). DOI http://dx.doi.org/10.1006/jagm.1993.1001
6. Bonsma, P.: Sparse cuts, matching-cuts and leafy trees in graphs. Ph.D. thesis, University of Twente, the Netherlands (2006)
7. Bonsma, P.: Spanning trees with many leaves in graphs with minimum degree three. SIAM J. Discrete Math. **22**(3), 920–937 (2008)
8. Bonsma, P.S., Brüggemann, T., Woeginger, G.J.: A faster fpt algorithm for finding spanning trees with many leaves. In: Proceedings of the 28th Conference on Mathematical Foundations of Computer Science (MFCS), *Lecture Notes in Computer Science*, vol. 2747, pp. 259–268. Springer(2003)
9. Bonsma, P.S., Dorn, F.: An FPT algorithm for directed spanning k-leaf (2007). `http://arxiv.org/abs/0711.4052`
10. Bonsma, P.S., Dorn, F.: Tight Bounds and Faster Algorithms for Directed Max-Leaf Problems. In: Proceedings of the 16th European Symposium on Algorithms (ESA), *Lecture Notes in Computer Science*, vol. 5193, pp. 222–233. Springer(2008)
11. Bonsma, P.S., Zickfeld, F.: A 3/2-Approximation Algorithm for Finding Spanning Trees with Many Leaves in Cubic Graphs. In: Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), no. 5344 in Lecture Notes in Computer Science. Springer(2008)
12. Bonsma, P.S., Zickfeld, F.: Spanning trees with many leaves in graphs without diamonds and blossoms. In: Proceedings of the 8th Symposium on Latin American Theoretical Informatics (LATIN), no. 4957 in Lecture Notes in Computer Science, pp. 531–543. Springer(2008)
13. Dai, F., Wu, J.: 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)
14. Daligault, J., Gutin, G., Kim, E.J., Yeo, A.: FPT algorithms and kernels for the directed k-leaf problem. J. Comput. Syst. Sci. **76**(2), 144–152 (2010)
15. Daligault, J., Thomassé, S.: On finding directed trees with many leaves. In: Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC), no. 5917 in Lecture Notes in Computer Science, pp. 86–97. Springer(2009)
16. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: P. Clote, J. Remmel (eds.): Feasible Mathematics II, pp. 219–244. Boston: Birkhäuser (1995)
17. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer-Verlag (1999)
18. Drescher, M., Vetta, A.R.: An approximation algorithm for the maximum leaf spanning arborescence problem. ACM Transactions on Algorithms (2009). Accepted for publication
19. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: Proceedings of the 1st Workshop on Algorithms and Complexity in Durham (ACiD), pp. 1–41 (2005)
20. Fellows, M.R., Langston, M.A.: On well-partial-ordering theory and its applications to combinatorial problems in VLSI design. SIAM Journal on Discrete Mathematics**5**, 117–126 (1992)
21. Fellows, M.R., McCartin, C., Rosamond, F.A., Stege, U.: Coordinatized kernels and catalytic reductions: An improved fpt algorithm for max leaf spanning tree and other problems. In: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 240–251. Springer-Verlag (2000)
22. Fernau, H., Fomin, F.V., Lokshtanov, D., Raible, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. In: Proceedings of the 26th Symposium on Theoretical Aspects of Computer Science (STACS), *Dagstuhl Seminar Proceedings*, vol. 09001, pp. 421–432. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2009)
23. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag (2006)
24. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2^n. Algorithmica **52**(2), 153–166 (2008)
25. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Information Processing Letters**52**(1), 45–49 (1994). DOI http://dx.doi.org/10.1016/0020-0190(94)90139-2

26. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
27. Gutin, G., Razgon, I., Kim, E.J.: Minimum Leaf Out-Branching Problems. In: Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM), no. 5034 in Lecture Notes in Computer Science, pp. 235–246. Springer(2008)
28. Impagliazzio, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences **63**(4):512–530 (2001)
29. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM Journal on Discrete Mathematics**4**(1), 99–106 (1991). DOI http://dx.doi.org/10.1137/0404010
30. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. In: Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP), Part I, *Lecture Notes in Computer Science*, vol. 5555, pp. 653–664. Springer (2009)
31. Liang, W.: 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), pp. 112–122. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/513800.513815
32. Linial, N., Sturtevant, D.: (1987). Unpublished result
33. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
34. Park, M.A., Willson, J., Wang, C., Thai, M., Wu, W., Farago, A.: A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In: Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), pp. 22–31. ACM, New York, NY, USA (2007)
35. Robertson, N., Seymour, P.D.: Graph minors XIII: The disjoint paths problem. Journal of Combinatorial Theory, Series B **63**, 65–110 (1995)
36. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Proceedings of the 6th European Symposium on Algorithms (ESA), no. 1461 in Lecture Notes in Computer Science, pp. 441–452. Springer (1998)
37. Thai, M., Wang, F., Liu, D., Zhu, S., Du, D.: Connected dominating sets in wireless networks with different transmission ranges. IEEE Transactions on Mobile Computing**6**(7), 721–730 (2007)