A New Algorithm for Finding Trees with Many Leaves*

Joachim Kneis, Alexander Langer, Peter Rossmanith

Dept. of Computer Science, RWTH Aachen University, Germany

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

1 Introduction

In this paper we consider the graph theoretical problems of finding trees and spanning trees in graphs, so that their number of leaves is maximal. To be more precise, given a graph G and a number k, we are to find a (spanning) tree with at least k leaves. For undirected graphs, the terms *tree* and *spanning tree* are well-known. These terms translate to *out-tree* and *spanning out-tree* on directed graphs. Here, a (spanning) out-tree is a rooted tree, such that every leaf (every node of G) can be reached from the root via a directed path within this tree.

Being a problem that has many practical applications, e.g., in network design [10, 18, 21, 24], it is already widely studied with regard to its complexity and approximability. All versions are APX-hard [15] and there is a polynomial time 2-approximation for undirected graphs [23] and a 3-approximation in almost linear time [20]. On cubic graphs, a 3/2-approximation was found recently [8].

In the area of parameterized algorithms, the Maximum Leaf Spanning Tree problem is very prominent. Parameterized complexity theory is an approach to explore whether hard problems can be solved exactly with a run time that comes close to polynomial time on well-behaved instances. Formally, a parameterized problem L is a set of pairs (I, k) where I is an *instance* and k the *parameter*. A parameterized problem L is called *fixed parameter tractable* and belongs to the complexity class FPT if there is an algorithm that decides membership of L in time $f(k)poly(|I|)$, where f is an arbitrary function. If the parameter is small, such an algorithm can be quite efficient in spite of the NP-hardness of the problem — in particular if f is a moderately exponential function.

The parameterized version of the undirected problem is defined as follows:

* Supported by the DFG under grant RO 927/7-1

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?

It is long known that MLST $\in$ FPT because a graph G contains a k-leaf spanning tree iff G has a $K_{1,k}$ (a k-star) as a minor [13]. However, this uses the graph minor theorem from Robertson and Seymour [22] and only proves the existence of an algorithm with running time $f(k)|V|^3$. The first explicit algorithm is due to Bodlaender [3], who uses the fact that G does contain a $K_{1,k}$ as a minor if its treewidth is larger than w_k, a value that depends on k. The algorithm hence tests if the treewidth of G is bigger than w_k. In this case, the algorithm directly answers "yes". Otherwise, it uses dynamic programming on a small tree decomposition of G. The overall run time is roughly $O((17k^4)!\,|G|)$. In the following years, the run time of algorithms deciding MLST was improved further to $O((2k)^{4k}\,poly(|G|))$ by Downey and Fellows [11], and to $O(|G| + 14.23^k k)$ by Fellows, McCartin, Rosamond, and Stege [14]. The latter was the first algorithm with an exponential cost function $2^O(k) \cdot poly(|G|)$ and the first algorithm that employs a *small problem kernel*: 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.

Bonsma, Brueggemann, and Woeginger [5] use an involved result from extremal graph theory by Linial and Sturtevant [19], and Kleitman and West [17] to bound the number of nodes that can possibly be leaves by $4k$. A brute force check for each k-subset of these $4k$ nodes yields a run time bound of $O(|V|^3 + 9.4815^k k^3)$. A new problem kernel of size $3.75k$ by Estivill-Castro, Fellows, Langston, and Rosamond [12] improves the exponential factor of this algorithm to 8.12^k [4]. The currently best known algorithm for MLST is due to Bonsma and Zickfeld [9], who reduce the instances to graphs without certain subgraphs called diamonds and blossoms that admit a better extremal result, obtaining a run time bound of $O(poly(|V|) + 6.75^k poly(k))$.

In the directed case, we have to distinguish between the following variants:

Directed Maximum Leaf Out-Tree (DMLOT)

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

Directed Maximum Leaf Spanning Out-Tree (DMLST)

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

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, this is not the case for directed graphs that are not strongly connected (see Figure 1 [6]).

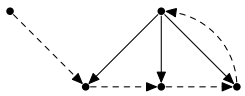

Fig. 1. A graph containing a 3-leaf out-tree, but no 3-leaf spanning tree.

For both of these problems, membership in FPT was discovered only recently, since neither the graph minor theorem by Robertson and Seymour in its current shape, nor the method used by Bodlaender, nor the extremal results by Kleitman-West are applicable for directed graphs.

In the case of DMLOT, Alon, Fomin, Gutin, Krivelevich, and Saurabh [2] proved an extremal result for directed graphs, so that either a k-leaf out-tree exists, or the pathwidth of the underlying graph is bounded by $2k^2$. This allows dynamic programming, so that an overall run time bound of $2^{O(k^2 \log k)} poly(|V|)$ can be achieved, answering the long open question whether DMLOT is fixed parameter tractable. They could further improve this to $2^{O(k \log^2 k)} poly(|V|)$ and, if G is acyclic, to $2^{O(k \log k)} poly(|V|)$ [1].

The more important question, if DMLST $\in$ FPT, remained open. Only very recently, Bonsma and Dorn [6] were able to answer this question in the affirmative. Their approach is based on pathwidth and dynamic programming as well and yields a run time bound of $2^{O(k^3 \log k)} poly(|V|)$. In a subsequent paper [7], they proved that a run time of $2^{O(k \log k)} poly(|V|)$ suffices to solve both, DMLOT and DMLST.

Our contribution

Recall that in the directed case a k-leaf out-tree cannot necessarily be extended to a k-leaf spanning out-tree even if G does contain a spanning out-tree (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 spanning out-tree if G does contain a spanning out-tree rooted in r.

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 both DMLOT and DMLST. 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 tree, if such a tree exists. The number of recursive calls can be bounded by $2^{2k} = 4^k$. The same algorithm can be used to solve MLST.

2 Preliminaries

Let $G = (V, E)$ be a graph, and let $n := |V|$ and $m := |E|$ be the number of vertices and edges, respectively. If G is undirected, we call a (spanning) tree T in G a *k-leaf (spanning) tree* iff T has at least k leaves. If G is a directed graph, a

rooted out-tree T is a tree in G, such that T has a unique root $r = \text{root}(T)$, 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 spanning out-tree* is a k-leaf out-tree that is also a spanning out-tree.

In this paper, we do not distinguish between directed and undirected graphs except when explicitly stated. The respective results and the algorithm can easily be transferred from directed graphs to undirected graphs and vice versa — in particular, if undirected graphs are seen as symmetric directed graphs, where every edge has a *reverse edge*. Such a representation is commonly used by algorithmic graph libraries like LEDA. Edges are therefore denoted by (u, v), and we universally use the terms *tree* and *spanning tree*. Without loss of generality ($k > 2$), trees in undirected graphs are assumed to be rooted.

Let T be a tree in G. $V(T)$ denotes the set of nodes of T, $E(T)$ the set of edges of T. The root, leaves, and inner nodes of T are denoted by $\text{root}(T)$, $\text{leaves}(T)$ and $\text{inner}(T) := V(T) \setminus \text{leaves}(T)$, respectively. We denote by $N(v) := \{ u \in V \mid (v, u) \in E \}$ the set of all neighbors of $v \in V$, $N[v] := N(v) \cup \{v\}$, and for $U \subseteq V$ we let $N(U) := \bigcup_{u \in U} N(u)$. For a tree T and $v \in V$, we set $N_{\overline{T}}(v) := N(v) \setminus V(T)$. Similarly, $N_{\overline{T}}(U) := N(U) \setminus V(T)$ for $U \subseteq V$. For $v \in V$, let $T_v := (N[v], \bigcup_{u \in N(v)} \{(v, u)\})$ be the star rooted in v that contains all neighbors of v.

Recall that our algorithm grows a tree from the root. To do so, the algorithm further distinguishes between leaves of trees that will be leaves in the final k-leaf tree (R), and leaves that are still allowed to become inner nodes (B), when the tree is *extended* by the algorithm. This extension consists of the complete remaining neighborhood of the particular node. The resulting tree T will be such that each inner node has all of its neighbors in $V(T)$. We call such trees *inner-maximal* trees.

Definition 1. *Let $G = (V, E)$ be a graph, and let T be a tree. If $N(\text{inner}(T)) \subseteq V(T)$, we call T an* inner-maximal *tree. A* leaf-labeled *tree is a 3-tuple (T, R, B), such that T is a tree, and R and B form a partition of* $\text{leaves}(T)$. *(T, R, B) is an* inner-maximal leaf-labeled tree, *if T is inner-maximal.*

For trees $T \neq T'$, we say T' extends *T, denoted by $T' \succ T$, iff $\text{root}(T') = \text{root}(T)$ and T is an induced subgraph of T'. If (T, R, B) is a leaf-labeled tree and T' is a tree such that $T' \succ T$ and $R \subseteq \text{leaves}(T')$ (R-colored leaves of T remain leaves in T'), we say T' is an* (leaf-preserving) extension *of (T, R, B), denoted by $T' \succ (T, R, B)$. We say a leaf-labeled tree (T', R', B')* extends *a leaf-labeled tree (T, R, B), denoted by $(T', R', B') \succ (T, R, B)$, iff $T' \succ (T, R, B)$.*

Lemma 1. *Let (T, R, B) be an inner-maximal leaf-labeled 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 $x \in V(T')$ 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 $x_{i+1} \notin V(T)$. It is $x_i \in \text{leaves}(T) = R \cup B$, because T is inner-maximal. On the other hand, $x_i \in \text{inner}(T')$, and with $R \subseteq \text{leaves}(T')$, we have $x_i \in B$. □

3 *k*-Leaf Trees versus *k*-Leaf Spanning Trees

In this section, we show when and how k-leaf trees can be extended to k-leaf spanning trees. For this to work, remember that we consider trees with *at least* k leaves. In particular, we allow that the resulting spanning tree has more leaves than the originating k-leaf tree. While Lemma 2 can be considered folklore, Lemma 3 is a new contribution that significantly eases our search for k-leaf spanning trees in directed graphs.

Lemma 2. *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 $O(n + m)$.*

Proof. Let T be a tree in G with at least k leaves, and let $l := |V - V(T)|$ be the number of nodes 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 $v \in N_{\overline{T}}(V(T))$, 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. If an explicit k-leaf spanning tree is asked for, the k-leaf tree can then be extended to a spanning tree using an efficient postprocessing operation.

Lemma 2 is, however, not applicable for directed graphs (cf., Figure 1): It is easy to see that this graph contains an out-tree with three leaves, but the unique spanning out-tree contains only one leaf. If we fix the root of the trees, we obtain the following weaker result for directed graphs.

Lemma 3. *Let $G = (V, E)$ be a directed graph. If G contains a k-leaf spanning out-tree rooted in x_1, then any k-leaf out-tree rooted in x_1 can be extended to a k-leaf spanning out-tree of G in time $O(n + m)$.*

Proof. Let T be a k-leaf out-tree with $\text{root}(T) = x_1$ and let $l := |V - V(T)|$ be the number of nodes that are not in T. If $l = 0$, then T is a spanning out-tree 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 a spanning out-tree rooted in 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 a spanning out-tree 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. □

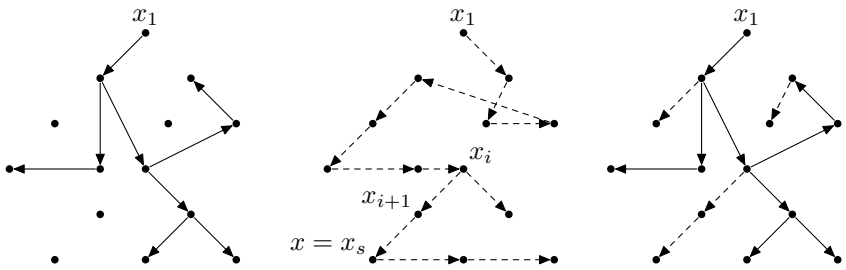

Fig. 2. How to extend a k-leaf out-tree into a k-leaf spanning out-tree: For the ease of illustration, we do not show all the edges in G. A 4-leaf out-tree with root x_1 is depicted in the first figure. The second figure shows an arbitrary spanning out-tree rooted in x_1, we chose one with two leaves. We can extend the first out-tree with edges from the spanning out-tree so that all nodes are covered.

4 The Algorithm

In this section, we introduce Algorithm 1, which given an inner-maximal leaf-labeled tree (T, R, B) recursively decides whether there is a k-leaf tree $T' \succeq (T, R, B)$. Informally, the algorithm works as follows: Choose a node $u \in B$ and recursively test whether there is a solution where u is a leaf, or whether there is a solution where u is an inner node. In the first case, u is moved from B to the set of fixed leaves R, so that u is preserved as a leaf in solutions T'. In the second case, u is considered an inner node and all of its outgoing edges to nodes in $N_{\overline{T}}(u)$ are added to T. The upcoming Lemma 4 guarantees that at least one of these two branches is successful, if a solution exists at all. In the special case that $|N_{\overline{T}}(u)| \leq 1$, we can skip the latter of the two branches by Lemma 5 and Corollary 1. Please note that the resulting algorithm is basically the same for directed and undirected graphs.

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

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

Proof. 1. This is clear, since x is either leaf or inner node in T'. 2. Let T' be a k-leaf tree, such that $T' \succeq (T, R, B)$ and $x \in \text{inner}(T')$. First note that $N_{\overline{T}}(x) \neq \emptyset$, because $x \in \text{inner}(T')$ and T is an induced subgraph of T'. Hence consider arbitrary $y \in N_{\overline{T}}(x)$. If $y \notin V(T')$, then we can construct a k-leaf tree T'' from T' by adding y and the edge (x, y). If $y \in V(T')$, but $(x, y) \notin E(T')$, consider the unique path $x_1, x_2, \ldots, x_i, y$ from $x_1 := \text{root}(T')$ to y in T'. We can now replace the edge (x_i, y) with (x, y) without decreasing the number of leaves in T': x is inner node 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

Algorithm 1 A fast algorithm for maximum leaf problems.

Algorithm MaxLeaf:
Input: Graph $G = (V, E)$, an inner-maximal leaf-labeled tree (T, R, B), $k \in \mathbf{N}$
Output: Is there a k-leaf 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 node in all extending solutions
05: B := B \ {u}
06: N := N_T̄(u)
07: T := T ∪ { (u, u') | u' ∈ N }
// follow paths, see Lemma 5
08: while |N| = 1 do
09:       Let u be the unique member of N.
10:       N := N_T̄(u)
11:       T := T ∪ { (u, u') | u' ∈ N }
12: done
// Do not branch if no neighbors left, see Corollary 1
13: if N = ∅ then return "no".
14: return MaxLeaf(G, T, R, B ∪ N, k)
```

neighbors y of x yields a k-leaf tree T'' with $\{ (x, y) \mid y \in N_{\overline{T}}(x) \} \subseteq E(T'')$. Therefore $T'' \succeq (T + \{ (x, y) \mid y \in N_{\overline{T}}(x) \}, R, N_{\overline{T}}(x) \cup B \setminus \{x\})$. See Figure 3 for an example. □

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

Proof. Let T' be a k-leaf tree that extends $(T + (x, y), R \cup \{y\}, B \setminus \{x\})$. Since in particular $T' \succ T + (x, y) \succ T$, y is the only child of x in T', and since $T' \succ (T + (x, y), R \cup \{y\}, B \setminus \{x\})$, y is leaf in T'. Hence y can be removed from T', obtaining a k-leaf tree T'' with $x \in \text{leaves}(T'')$, i.e., $T'' \succ (T, R \cup \{x\}, B \setminus \{x\})$. □

Corollary 1. *Let $G = (V, E)$ be a graph, (T, R, B) a leaf-labeled tree and $x \in B$ with $N_{\overline{T}}(x) = \emptyset$. If there is a k-leaf tree that extends (T, R, B), there is a k-leaf tree that extends $(T, R \cup \{x\}, B \setminus \{x\})$.*

Proof. Let T' be a k-leaf tree that extends (T, R, B). It is $x \in B \subseteq \text{leaves}(T)$. Since $N_{\overline{T}}(x) = \emptyset$, we have $N(x) \subseteq V(T) \subseteq V(T')$. For each $y \in N(x)$, there is $z \in V(T)$ with $(z, y) \in E(T)$. In particular, if $(x, y) \notin E(T)$, then $(x, y) \notin E(T')$, since T' is a tree and $E(T) \subseteq E(T')$. Hence $x \in \text{leaves}(T')$ and $T' \succ (T, R \cup \{x\}, B \setminus \{x\})$. □

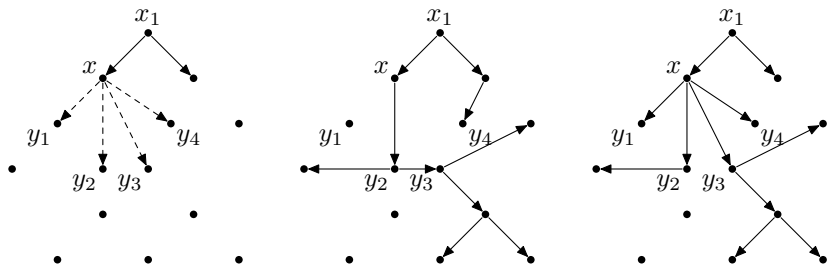

Fig. 3. The exchange argument (Lemma 4): The first figure shows a leaf-labeled tree (T, R, B) with $x \in B$. The neighborhood of x, $N_{\overline{T}}(x)$, is shown with dashed edges. The second figure shows a 5-leaf tree $T' \succ (T, R, B)$, but different choices for edges originating in x have been 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 edge (x, y_2), y_3 remains inner node, and y_4 remains leaf, although it is now connected through x.

Lemma 6. *Let $G = (V, E)$ be a graph and let $k > 2$. If G does not contain a k-leaf tree,* MaxLeaf$(G, T_v, \emptyset, N(v), k)$ *returns "no" for each $v \in V$. If G contains a k-leaf tree rooted in r, Algorithm 1 returns "yes" if called as* MaxLeaf$(G, T_r, \emptyset, N(r), k)$.

Proof. We first show that all subsequent calls to MaxLeaf are always given an inner-maximal leaf-labeled tree: The star T_v is inner-maximal, and hence $(T_v, \emptyset, N(v))$ is an inner-maximal leaf-labeled tree. Let (T, R, B) be the inner-maximal tree given as argument to MaxLeaf. The algorithm chooses $x \in B$ and either fixes it as a leaf or as an inner node. If x becomes a leaf, then $(T, R \cup \{x\}, B \setminus \{x\}) \succ (T, R, B)$ is inner-maximal. If otherwise x becomes inner node, a tree T' is obtained from T by adding the nodes in $N_{\overline{T}}(x)$ as children of x, so that they are leaves. 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 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 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 tree $(T^{(l+1)}, R^{(l+1)}, B^{(l+1)})$.

Whenever MaxLeaf(G, T, R, B, k) returns "yes", T is a tree in G with $|\text{leaves}(T)| = |R \cup B| = |R| + |B| \geq k$. Therefore, G does contain a k-leaf tree and the algorithm never answers "yes" on no-instances. If otherwise G contains a k-leaf tree rooted in r, we use induction over $\succ$ as follows: Under the hypothesis that (T, R, B) is an inner-maximal leaf-labeled tree, such that there is a k-leaf tree $T' \succeq (T, R, B)$, we prove: Either $T = T'$, or there are (T''', R''', B''') and (T'', R'', B''), such that T''' is a k-leaf tree, $(T''', R''', B''') \succeq (T'', R'', B'') \succ (T, R, B)$ and MaxLeaf is called with (T'', R'', B''). Since G is finite, eventually MaxLeaf is called with a k-leaf leaf-labeled tree and returns "yes".

Let r be the root of some k-leaf tree T in G. Since $k > 2$, we may assume $r \in \text{inner}(T)$ even when G is undirected. Consider $T' = (\{r\}, \emptyset)$. Then $(T', \emptyset, \{r\})$ is a leaf-labeled tree, and trivially $T \succ (T', \emptyset, \{r\})$. By Lemma 4, then there is also a k-leaf tree $T'' \succ (T_r, \emptyset, N(r))$.

We hence may now consider an arbitrary inner-maximal leaf-labeled tree (T, R, B) that is given a argument to MaxLeaf, such that there is a k-leaf tree $T' \succeq (T, R, B)$. If $|\text{leaves}(T)| = |R \cup B| \geq k$, then (T, R, B) already is a k-leaf tree in G and the algorithm correctly returns "yes".

Otherwise, $B \neq \emptyset$ by Lemma 1, since (T, R, B) is inner-maximal. Fix an arbitrary $u \in B$. By Lemma 4, there is a k-leaf tree $T''' \succeq (T, R \cup \{u\}, B \setminus \{u\})$, or there is a k-leaf tree $T''' \succeq (T + \{ (u, y) \mid y \in N_{\overline{T}}(u) \}, R, N_{\overline{T}}(u) \cup B \setminus \{u\})$.

We first assume the first case is true. Then $T''' \succeq (T, R \cup \{u\}, B \setminus \{u\}) \succ (T, R, B)$ and the call to MaxLeaf$(G, T, R \cup \{u\}, B \setminus \{u\}, k)$ does satisfy the induction hypothesis for the next induction step. If however the first case is false, we know by Lemma 4, that since there is at least one k-leaf tree that extends (T, R, B) (namely $T' \succeq (T, R, B)$), there is also a k-leaf tree $T''' \succeq (T + \{ (u, y) \mid y \in N_{\overline{T}}(u) \}, R, N_{\overline{T}}(u) \cup B \setminus \{u\})$. Furthermore, by Lemma 5 there is a unique sequence of vertices $v_0, v_1, \ldots, v_l$ and leaf-labeled trees $(T_0, R_0, B_0), \ldots, (T_l, R_l, B_l)$, such that $v_0 = u$, $(T_0, R_0, B_0) = (T, R, B)$, and

1. $(T_{i+1}, R_{i+1}, B_{i+1}) = (T_i + (v_i, v_{i+1}), R_i, B_i \cup N_{\overline{T_i}}(v_i) \setminus \{v_i\})$,
2. $N_{\overline{T_i}}(v_i) = \{v_{i+1}\}$ for $0 \leq i < l$,
3. $|N_{\overline{T_l}}(v_l)| \neq 1$, and
4. for each $0 \leq i \leq l$ there is a k-leaf tree $T_i' \succeq (T_i, R_i, B_i)$.

By Corollary 1, we have $N_{\overline{T_l}}(v_l) \neq \emptyset$, i.e., the algorithm does not return "no". Hence the algorithm recursively calls itself as MaxLeaf(G, T_l, R_l, B_l, k), where (T_l, R_l, B_l) satisfies the induction hypothesis. □

Lemma 7. *Let $G = (V, E)$ be a graph and $v \in V$. The number of recursive calls of Algorithm 1 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 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 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 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 $2^{\Phi(\emptyset, N(v))} = O(2^{2k-|N(v)|}) = O(4^k)$. □

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 [12], 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 $k > 2$. We do not know, which node $v \in V$ suffices as a root. It is however 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 node $u \in \text{inner}(T)$. If furthermore v has minimum degree, the cost to test all $u \in N[v]$ disappears in the run time estimation.

Therefore, let $v \in V$ be a node of minimum degree. We need to call MaxLeaf with arguments $(G, T_u, R, N(u), k)$ for all $u \in N[v]$: 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 and an additional per-vertex membership flag is used, so that a membership test and insert and delete set 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 can be done in constant time as well, since $|N_{\overline{T}}(u)| = 1$. 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 k-leaf tree is bounded by $O(poly(n) + 4^k \cdot k^2)$. By Lemma 2, 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 1 can easily be modified to return a k-leaf spanning tree in G within the same run time bound. In this case, an additional $O(n + m)$ postprocessing is required to extend the k-leaf tree to a k-leaf spanning tree.

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

Proof. Let $G = (V, E)$ be a directed graph. We first consider DMLOT: If G contains a k-leaf out-tree rooted in 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

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

What remains to show is that only $O(n+m) = O(m)$ operations are performed *on average* on each call of MaxLeaf. Consider one complete path in the recursion tree: It is easy to see, that each vertex $v \in V$ occurs at most once as the respective u in either lines 6 or 10. In particular each edge (v, w) is visited at most once per path when computing $N_{\overline{T}}(u)$. Therefore, the overall run time to solve DMLOT is bounded by $O(4^k \cdot nm)$.

To prove the run time bound for DMLST, 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 spanning out-tree. By Lemma 3, each k-leaf out-tree that shares the same root with some k-leaf spanning out-tree *can* be extended to a k-leaf spanning out-tree in time $O(n+m) = O(m)$. Thus the run time remains bounded by $O(4^k \cdot nm)$. □

Conclusion

We solve open problems [7, 16] on whether there exist $c^k poly(n)$-time algorithms for the k-leaf out-tree and k-leaf spanning out-tree problems on directed graphs. Our algorithms for DMLOT and DMLST have a run time of $O(4^k|V||E|)$, which is a significant improvement over the currently best bound of $2^{O(k \log k)} poly(|V|)$.

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 $O(poly(|V|) + 4^k \cdot k^2)$, where $poly(|V|)$ is the time to compute the problem kernel of size $3.75k$. This improves over the currently best algorithm with a run time of $O(poly(|V|) + 6.75^k poly(k))$.

The question by Michael Fellows et al. from the year 2000 [14] whether there will ever be a parameterized algorithm for MLST with running time $f(k)poly(n)$, where $f(50) < 10^{20}$ unfortunately remains open, but the gap is not so big anymore.

References

1. N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Better algorithms and bounds for directed maximum leaf problems. In *Proc. of 27th FSTTCS*, number 4855 in LNCS, pages 316–327. Springer, Nov. 2007.
2. N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized algorithms for directed maximum leaf problems. In *Proc. of 34th ICALP*, number 4596 in LNCS, pages 352–362. Springer, 2007.
3. H. L. Bodlaender. On linear time minor tests with depth-first search. *J. Algorithms*, 14(1):1–23, 1993.

4. P. Bonsma. *Sparse cuts, matching-cuts and leafy trees in graphs.* PhD thesis, University of Twente, the Netherlands, 2006.
5. P. S. Bonsma, T. Brüggemann, and G. J. Woeginger. A faster fpt algorithm for finding spanning trees with many leaves. In *Proc. of 28th MFCS*, volume 2747 of *LNCS*, pages 259–268. Springer, 2003.
6. P. S. Bonsma and F. Dorn. An FPT algorithm for directed spanning k-leaf, 2007. `http://arxiv.org/abs/0711.4052`.
7. P. S. Bonsma and F. Dorn. Tight Bounds and Faster Algorithms for Directed Max-Leaf Problems. In *Proc. of 16th ESA*, LNCS. Springer, 2008. To appear.
8. P. S. Bonsma and F. Zickfeld. A 3/2-Approximation Algorithm for Finding Spanning Trees with Many Leaves in Cubic Graphs. In *Proc. of 34th WG*, LNCS. Springer, 2008. To appear.
9. P. S. Bonsma and F. Zickfeld. Spanning trees with many leaves in graphs without diamonds and blossoms. In *Proc. of 8th LATIN*, number 4957 in LNCS, pages 531–543. Springer, 2008.
10. F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 15(10):908–920, 2004.
11. R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.
12. V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. FPT is P-time extremal structure I. In *Proc. of 1st ACiD*, pages 1–41, 2005.
13. M. R. Fellows and M. A. Langston. On well-partial-ordering theory and its applications to combinatorial problems in VLSI design. *SIAM J. Discrete Math.*, 5:117–126, 1992.
14. 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 *Proc. of 20th FSTTCS*, pages 240–251. Springer-Verlag, 2000.
15. G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Inf. Process. Lett.*, 52(1):45–49, 1994.
16. G. Gutin, I. Razgon, and E. J. Kim. Minimum Leaf Out-Branching Problems. In *Proc. of. AAIM 2008*, volume 5034 of *LNCS*, pages 235–246, 2008.
17. D. J. Kleitman and D. B. West. Spanning trees with many leaves. *SIAM J. Discret. Math.*, 4(1):99–106, 1991.
18. W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proc. of 3rd MobiHoc*, pages 112–122, New York, NY, USA, 2002. ACM.
19. N. Linial and D. Sturtevant, 1987. Unpublished result.
20. H. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *J. Algorithms*, 29(1):132–141, 1998.
21. M. A. Park, J. Willson, C. Wang, M. Thai, W. Wu, and A. Farago. A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In *Proc. of 8th MobiHoc*, pages 22–31, New York, NY, USA, 2007. ACM.
22. N. Robertson and P. D. Seymour. Graph minors—a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge University Press, 1985.
23. R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In *Proc. of 6th ESA*, number 1461 in LNCS, pages 441–452. Springer, 1998.
24. M. Thai, F. Wang, D. Liu, S. Zhu, and D. Du. Connected dominating sets in wireless networks with different transmission ranges. *IEEE Trans. Mob. Comput.*, 6(7):721–730, 2007.