

# FPT Algorithms and Kernels for the Directed $k$ -Leaf Problem

Jean Daligault\*, Gregory Gutin†, Eun Jung Kim‡ and Anders Yeos§

May 13, 2009

## Abstract

A subgraph  $T$  of a digraph  $D$  is an *out-branching* if  $T$  is an oriented spanning tree with only one vertex of in-degree zero (called the *root*). The vertices of  $T$  of out-degree zero are *leaves*. In the DIRECTED  $k$ -LEAF Problem, we are given a digraph  $D$  and an integral parameter  $k$ , and we are to decide whether  $D$  has an out-branching with at least  $k$  leaves. Recently, Kneis et al. (2008) obtained an algorithm for the problem of running time  $4^k \cdot n^{O(1)}$ . We describe a new algorithm for the problem of running time  $3.72^k \cdot n^{O(1)}$ . In ROOTED DIRECTED  $k$ -LEAF Problem, apart from  $D$  and  $k$ , we are given a vertex  $r$  of  $D$  and we are to decide whether  $D$  has an out-branching rooted at  $r$  with at least  $k$  leaves. Very recently, Fernau et al. (2008) found an  $O(k^3)$ -size kernel for ROOTED DIRECTED  $k$ -LEAF. In this paper, we obtain an  $O(k)$  kernel for ROOTED DIRECTED  $k$ -LEAF restricted to acyclic digraphs.

## 1 Introduction

The MAXIMUM LEAF problem is to find a spanning tree with the maximum number of leaves in a given undirected graph  $G$ . The problem is well studied from both algorithmic [17, 18, 23, 25] and graph-theoretical [10, 19, 20, 22] points of view. The problem has been studied from the parameterized complexity perspective as well and several authors [7, 13, 14] have designed fixed parameter tractable (FPT) algorithms for solving the parameterized version of MAXIMUM LEAF (the  $k$ -LEAF problem): given a graph  $G$  and an integral parameter  $k$ , decide whether  $G$  has a spanning tree with at least  $k$  leaves.

MAXIMUM LEAF can be extended to digraphs. A subgraph  $T$  of a digraph  $D$  is an *out-tree* if  $T$  is an oriented tree with only one vertex of in-degree zero (called the *root*).

---

\*Université Montpellier II, LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5 - France, daligault@lirmm.fr

†Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, gutin@cs.rhul.ac.uk

‡Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, eunjung@cs.rhul.ac.uk

§Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, anders@cs.rhul.ac.uk

The vertices of  $T$  of out-degree zero are *leaves*. If  $V(T) = V(D)$ , then  $T$  is an *out-branching* of  $D$ . The DIRECTED MAXIMUM LEAF problem is to find an out-branching with the maximum number of leaves in an input digraph. The parameterized version of the DIRECTED MAXIMUM LEAF problem is DIRECTED  $k$ -LEAF: given a digraph  $D$  and an integral parameter  $k$ , decide whether  $D$  has an out-branching with at least  $k$  leaves. If we add a condition that every out-branching in DIRECTED  $k$ -LEAF must be rooted at a given vertex  $r$ , we obtain a variation of DIRECTED  $k$ -LEAF called the ROOTED DIRECTED  $k$ -LEAF problem.

The study of DIRECTED  $k$ -LEAF has only begun recently. Alon et al. [1, 2] proved that the problem is FPT for a wide family of digraphs including classes of strongly connected and acyclic digraphs. Bonsma and Dorn extended this result to all digraphs in [8], and improved the running time of the algorithm in [2] to  $2^{k \log k} n^{O(1)}$  in [9]. Recently, Kneis et al. [21] obtained an algorithm for solving the problem in time  $4^k n^{O(1)}$ . Notice that the algorithm of Kneis et al. [21] applied to undirected graphs is of smaller running time (as a function of  $k$ ) than all previously known algorithms for  $k$ -LEAF. Yet, the algorithm of Kneis et al. [21] is not fast enough to answer in affirmative the question of Fellows et al. [14] of whether there exists a parameterized algorithm for MAX LEAF of running time  $f(k)n^{O(1)}$ , where  $f(50) < 10^{20}$ . Very recently, Fernau et al. [15] proved that no polynomial kernel for DIRECTED  $k$ -LEAF is possible unless polynomial hierarchy collapses to the third level (they applied a recent breakthrough result of Bodlaender et al. [6]). Interestingly, ROOTED DIRECTED  $k$ -LEAF admits a polynomial size kernel and Fernau et al. [15] obtained one of size  $O(k^3)$ .

The only known approximation algorithm for DIRECTED MAX LEAF is due to Drescher and Vetta [12] and its approximation ratio is  $O(\sqrt{\ell_{\max}(D)})$ , where  $\ell_{\max}(D)$  is the maximum number of leaves in an out-branching of a digraph  $D$ .

In this paper, we obtain an algorithm faster than the one of Kneis et al. [21] for DIRECTED  $k$ -LEAF. Our algorithm runs in time  $3.72^k n^{O(1)}$ . Unfortunately, our algorithm cannot solve the above-mentioned question of Fellows et al. [14], but it shows that the remaining gap is not wide anymore. We also obtain a linear size kernel for DIRECTED  $k$ -LEAF restricted to acyclic digraphs. Notice that (i) DIRECTED MAX LEAF restricted to acyclic digraphs is still NP-hard [3], and (ii) for acyclic digraphs DIRECTED  $k$ -LEAF and ROOTED DIRECTED  $k$ -LEAF are equivalent since all out-branchings must be rooted at the unique vertex of in-degree zero.

We recall some basic notions of parameterized complexity here, for a more in-depth treatment of the topic we refer the reader to [11, 16, 24].

A parameterized problem  $\Pi$  can be considered as a set of pairs  $(I, k)$  where  $I$  is the *problem instance* and  $k$  (usually an integer) is the *parameter*.  $\Pi$  is called *fixed-parameter tractable (FPT)* if membership of  $(I, k)$  in  $\Pi$  can be decided in time  $O(f(k)|I|^c)$ , where  $|I|$  is the size of  $I$ ,  $f(k)$  is a computable function, and  $c$  is a constant independent from  $k$  and  $I$ . Let  $\Pi$  be a parameterized problem. A *reduction  $R$  to a problem kernel* (or *kernelization*) is a many-to-one transformation from  $(I, k) \in \Pi$  to  $(I', k') \in \Pi'$ , such that (i)  $(I, k) \in \Pi$  if and only if  $(I', k') \in \Pi$ , (ii)  $k' \leq k$  and  $|I'| \leq g(k)$  for some function  $g$  and (iii)  $R$  is computable in time polynomial in  $|I|$  and  $k$ . In kernelization, an instance  $(I, k)$  is reduced to another instance  $(I', k')$ , which is called the *problem kernel*;  $|I'|$  is the *size* of the kernel.

The set of vertices (arcs) of a digraph  $D$  will be denoted by  $V(D)$  ( $A(D)$ ). The

number of vertices (arcs) of the digraph under consideration will be denoted  $n$  ( $m$ ). For a vertex  $x$  of a subgraph  $H$  of a digraph  $D$ ,  $N_H^+(x)$  and  $N_H^-(x)$  denote the sets of out-neighbors and in-neighbors of  $x$ , respectively. Also, let  $A_H^+(x) = \{xy : y \in N_H^+(x)\}$ ,  $d_H^+(x) = |N_H^+(x)|$ , and  $d_H^-(x) = |N_H^-(x)|$ . When  $H = D$  we will omit the subscripts in the notation above.

Let  $D$  be a digraph,  $T$  an out-tree and  $L \subseteq V(D)$ . A  $(T, L)$ -out-tree of  $D$  is an out-tree  $T'$  of  $D$  such that (1)  $A(T) \subseteq A(T')$ , (2)  $L$  are leaves in  $T'$ , (3)  $T$  and  $T'$  have the same root. A  $(T, L)$ -out-branching is a  $(T, L)$ -out-tree which is spanning. Let  $\ell_{\max}(D, T, L)$  be the maximum number of leaves over all  $(T, L)$ -out-branchings of  $D$ . We set this number to 0 if there is no  $(T, L)$ -out-branching. For an out-tree  $T$  in a digraph  $D$ ,  $\text{Leaf}(T)$  denotes the set of leaves in  $T$  and  $\text{Int}(T) = V(T) - \text{Leaf}(T)$ , the set of *internal vertices* of  $T$ . For any vertex  $x$  in a tree  $T$  let  $T_x$  denote the maximal subtree of  $T$  which has  $x$  as its root.

Throughout this paper we use a triple  $(D, T, L)$  to denote a given digraph  $D$ , an out-tree  $T$  of  $D$  and a set of vertices  $L \subseteq V(D) - \text{Int}(T)$ . We denote by  $\hat{D}(T, L)$  the subgraph of  $D$  obtained after deleting all arcs out of vertices in  $L$  and all arcs not in  $A(T)$  which go into a vertex in  $V(T)$ . When  $T$  and  $L$  are clear from the context we will omit them and denote  $\hat{D}(T, L)$  by  $\hat{D}$ . For further terminology and notation on directed graphs, one may consult [5]. The following simple lemma will be used in the rest of the paper.

**Lemma 1.1.** [5] *A digraph  $D$  has an out-branching if and only if  $D$  has a single strong component without incoming arcs. One can decide whether a digraph has an out-branching in time  $O(n + m)$ .*

## 2 Another $4^k n^{O(1)}$ Time Algorithm

The algorithm of this section is similar to the algorithm in [21], but it differs from the algorithm in [21] as follows. We decide in an earlier stage which one of the current leaves of  $T$  cannot be a leaf in a final  $(T, L)$ -out-branching and make them to be internal vertices based on Lemma 2.3, see step 2 in Algorithm  $\mathcal{A}(D, T, L)$ . This decision works as a preprocessing of the given instance and gives us a better chance to come up with a  $(T, L)$ -out-tree with at least  $k$  leaves more quickly. A more important reason for this step is the fact that our algorithm is easier than the main algorithm in [21] to transform into a faster algorithm.

The following simple result was used in [1, 2] and its proof can be found in [21].

**Lemma 2.1.** *If there is an out-branching rooted at vertex  $r$ , whenever we have an out-tree rooted at  $r$  with at least  $k$  leaves we can extend it to an out-branching rooted at  $r$  with at least  $k$  leaves in time  $O(m + n)$ .*

**Lemma 2.2.** *Given a triple  $(D, T, L)$ , we have  $\ell_{\max}(D, T, L) = \ell_{\max}(\hat{D}, T, L)$ .*

*Proof.* If there is no  $(T, L)$ -out-branching in  $D$ , the subgraph  $\hat{D}$  does not have a  $(T, L)$ -out-branching either and the equality holds trivially. Hence suppose that  $T^*$  is a  $(T, L)$ -out-branching in  $D$  with  $\ell_{\max}(D, T, L)$  leaves. Obviously we have  $\ell_{\max}(D, T, L) \geq \ell_{\max}(\hat{D}, T, L)$ . Since the vertices of  $L$  are leaves in  $T^*$ , all arcs out of vertices in  $L$  do not appear in  $T^*$ , i.e.  $A(T^*) \subseteq A(D) \setminus \{A_D^+(x) : x \in L\}$ . Moreover  $A(T) \subseteq A(T^*)$

and thus all arcs not in  $A(T)$  which go into a vertex in  $V(T)$  do not appear in  $T^*$  since otherwise we have a vertex in  $V(T)$  with more than one arc of  $T^*$  going into it (or, the root has an arc going into it). Hence we have  $A(T^*) \subseteq A(\hat{D})$  and the above equality holds.  $\square$

**Lemma 2.3.** *Given a triple  $(D, T, L)$ , the following equality holds for each leaf  $x$  of  $T$ .*

$$\ell_{\max}(D, T, L) = \max\{\ell_{\max}(D, T, L \cup \{x\}), \ell_{\max}(D, T \cup A_D^+(x), L)\}$$

*Proof.* If  $\ell_{\max}(D, T, L) = 0$  then the equality trivially holds, so we assume that  $\ell_{\max}(D, T, L) \geq 1$ . Since any  $(T, L \cup \{x\})$ -out-branching or  $(T \cup A_D^+(x), L)$ -out-branching is a  $(T, L)$ -out-branching as well, the inequality  $\geq$  obviously holds. To show the opposite direction, suppose  $T'$  is an optimal  $(T, L)$ -out-branching. If  $x$  is a leaf in  $T'$ , then  $T'$  is a  $(T, L \cup \{x\})$ -out-branching and  $\ell_{\max}(D, T, L) \leq \ell_{\max}(D, T, L \cup \{x\})$ .

Suppose  $x$  is not a leaf in  $T'$ . Delete all arcs entering  $N_{\hat{D}}^+(x)$  in  $T'$ , add  $A_D^+(x)$  and let  $T''$  denote the resulting subgraph. Note that  $d_{T''}^-(y) = 1$  for each vertex  $y$  in  $\hat{T}''$  which is not the root and  $A(T'') \subseteq A(\hat{D})$ . In order to show that  $T''$  is an out-branching it suffices to see that there is no cycle in  $T''$  containing  $x$ . If there is a cycle  $C$  containing  $x$  in  $T''$  and  $xy \in A(C)$ , then  $C - \{xy\}$  forms a directed  $(y, x)$ -path in  $\hat{D}$ . However this is a contradiction as  $x \in V(T)$  and  $y \notin V(T)$  and there is no path from  $V(D) - V(T)$  to  $V(T)$  in  $\hat{D}$ . Hence  $T''$  is an out-branching.

As no vertex in  $L$  has any arcs out of it in  $\hat{D}$  we note that  $L \subseteq \text{Leaf}(T'')$ . Furthermore we note that  $A(T) \subseteq A(T'')$  as  $A(T) \subseteq A(T')$  and all arcs we deleted from  $A(T')$  go to a vertex not in  $V(T)$ . Therefore  $T''$  is a  $(T, L)$ -out-branching which has as many leaves as  $T'$ . This shows  $\ell_{\max}(D, T, L) \leq \ell_{\max}(D, T \cup A_D^+(x), L)$ .  $\square$

**Definition 2.4.** *Given a triple  $(D, T, L)$  and a vertex  $x \in \text{Leaf}(T) - L$ , define  $T_{D,L}^{\text{root}}(x)$  as follows.*

- (1)  $x' := x$ .
- (2) While  $d_D^+(x') = 1$  add  $A_D^+(x) = \{x'y\}$  to  $T$  and let  $x' := y$ .
- (3) Add  $A_D^+(x')$  to  $T$ .

Now let  $T_{D,L}^{\text{root}}(x) = T_x$ . That is,  $T_{D,L}^{\text{root}}(x)$  contains exactly the arcs added by the above process.

**Lemma 2.5.** *Suppose we are given a triple  $(D, T, L)$  and a leaf  $x \in \text{Leaf}(T) - L$ . If  $\ell_{\max}(D, T, L \cup \{x\}) \geq 1$  then the following holds.*

- (i) If  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$  then  $\ell_{\max}(D, T, L) = \max\{\ell_{\max}(D, T, L \cup \{x\}), \ell_{\max}(D, T \cup T_{D,L}^{\text{root}}(x), L)\}$ .
- (ii) If  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| = 1$  then  $\ell_{\max}(D, T, L) = \ell_{\max}(D, T, L \cup \{x\})$ .

*Proof.* Assume that  $T'$  is an optimal  $(T, L)$ -out-branching and that  $|\text{Leaf}(T'_x)| = 1$ . We will now show that  $\ell_{\max}(D, T, L \cup \{x\}) = |\text{Leaf}(T')| = \ell_{\max}(D, T, L)$ . If  $x$  is a leaf of  $T'$  then this is clearly the case, so assume that  $x$  is not a leaf of  $T'$ . Let  $y$  be the unique out-neighbor of  $x$  in  $T'$ . As  $\ell_{\max}(D, T, L \cup \{x\}) \geq 1$  we note that there exists

a path  $P = p_0 p_1 p_2 \dots p_r (= y)$  from the root of  $T$  to  $y$  in  $\hat{D}(T, L \cup \{x\})$ . Assume that  $q$  is chosen such that  $p_q \notin T'_x$  and  $\{p_{q+1}, p_{q+2}, \dots, p_r\} \subseteq V(T'_x)$ . Consider the digraph  $D^* = D[V(T'_x) \cup \{p_q\} - \{x\}]$  and note that  $p_q$  can reach all vertices in  $D^*$ . Therefore there exists an out-branching in  $D^*$ , say  $T^*$ , with  $p_q$  as the root. Let  $T''$  be the out-branching obtained from  $T'$  by deleting all arcs in  $T'_x$  and adding all arcs in  $T^*$ . Note that  $|\text{Leaf}(T'')| \geq |\text{Leaf}(T')|$  as  $\text{Leaf}(T^*) \cup \{x\}$  are leaves in  $T''$  and  $\text{Leaf}(T'_x) \cup \{p_q\}$  are the only leaves in  $T'$  which may not be leaves in  $T''$  (and  $|\text{Leaf}(T'_x) \cup \{p_q\}| = 2$ ). Therefore  $\ell_{\max}(D, T, L \cup \{x\}) \geq |\text{Leaf}(T'')| = \ell_{\max}(D, T, L)$ . As we always have  $\ell_{\max}(D, T, L) \geq \ell_{\max}(D, T, L \cup \{x\})$  we get the desired equality.

This proves part (ii) of the lemma, as if  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| = 1$  then any optimal  $(T, L)$ -out-branching,  $T'$ , must have  $|\text{Leaf}(T'_x)| = 1$ .

We therefore consider part (i), where  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$ . Let  $Q$  denote the set of leaves of  $T_{D,L}^{\text{root}}(x)$  and let  $R = V(T_{D,L}^{\text{root}}(x)) - Q$ . Note that by the construction of  $T_{D,L}^{\text{root}}(x)$  the vertices of  $R$  can be ordered  $(x =) r_1, r_2, \dots, r_i$  such that  $r_1 r_2 \dots, r_i$  is a path in  $T_{D,L}^{\text{root}}(x)$ . As before let  $T'$  be an optimal  $(T, L)$ -out-branching and note that if any  $r_j$  ( $1 \leq j \leq i$ ) is a leaf of  $T'$  then  $|\text{Leaf}(T'_x)| = 1$  and the above gives us  $\ell_{\max}(D, T, L \cup \{x\}) = \ell_{\max}(D, T, L)$ . This proves part (i) in this case, as we always have  $\ell_{\max}(D, T, L) \geq \ell_{\max}(D, T \cup T_{D,L}^{\text{root}}(x), L)$ . Therefore no vertex in  $\{r_1, r_2, \dots, r_i\}$  is a leaf of  $T'$  and all arcs  $(x =) r_1 r_2, r_2 r_3, \dots, r_{i-1} r_i$  belong to  $T'$ . By Lemma 2.3 we may furthermore assume that  $T'$  contains all the arcs from  $r_i$  to vertices in  $Q$ . Therefore  $T_{D,L}^{\text{root}}(x)$  is a subtree of  $T'$  and  $\ell_{\max}(D, T, L) = \ell_{\max}(D, T \cup T_{D,L}^{\text{root}}(x), L)$ . This completes the proof of part (i).  $\square$

The following is an  $O(4^k n^{O(1)})$  algorithm. Its complexity can be obtained similarly to [21]. We restrict ourselves only to proving its correctness.

For every vertex  $x \in V(D)$ , do  $\mathcal{A}(D, \{x\}, \emptyset)$ .

If one of the returns of  $\mathcal{A}(D, \{x\}, \emptyset)$  is “YES” then output “YES”.

Otherwise, output “NO”.

$\mathcal{A}(D, T, L)$ :

- (1) If  $\ell_{\max}(D, T, L) = 0$ , return “NO”. Stop.
- (2) While there is a vertex  $x \in \text{Leaf}(T) - L$  such that  $\ell_{\max}(D, T, L \cup \{x\}) = 0$ , add the arcs  $A_D^+(x)$  to  $T$ .
- (3) If  $|L| \geq k$ , return “YES”. Stop.  
If the number of leaves in  $T$  is at least  $k$ , return “YES”. Stop.  
If all leaves in  $T$  belong to  $L$ , return “NO”. Stop.
- (4) Choose a vertex  $x \in \text{Leaf}(T) - L$ .  
 $B_1 := \mathcal{A}(D, T, L \cup \{x\})$  and  $B_2 := \text{“NO”}$ .  
If  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$  then let  $B_2 := \mathcal{A}(D, T \cup T_{D,L}^{\text{root}}(x), L)$ .  
Return “YES” if either  $B_1$  or  $B_2$  is “YES”. Otherwise return “NO”.

**Remark 2.6.** While the first line in step 3 is unnecessary, we keep it since it is needed in the next algorithm where  $L \subseteq \text{Leaf}(T)$  is not necessarily true, see (4.2) in the next algorithm, where  $p_0 \notin V(T)$ .

**Theorem 2.7.** *Algorithm  $\mathcal{A}(D, T, L)$  works correctly. In other words,  $D$  has a  $(T, L)$ -out-branching with at least  $k$  leaves if and only if Algorithm  $\mathcal{A}(D, T, L)$  returns “YES”.*

*Proof.* We begin by showing that a call to  $\mathcal{A}(D, T, L)$  is always made with a proper argument  $(D, T, L)$ , that is,  $T$  is an out-tree of  $D$  and  $L \cap \text{Int}(T) = \emptyset$ . Obviously the initial argument  $(D, \{x\}, \emptyset)$  is proper. Suppose  $(D, T, L)$  is a proper argument. It is easy to see that  $(D, T, L \cup \{x\})$  is a proper argument. Let us consider  $(D, T \cup T_{D,L}^{\text{root}}(x), L)$ . By Definition 2.4 we note that  $T \cup T_{D,L}^{\text{root}}(x)$  is an out-tree in  $D$  and since we consider the digraph  $\hat{D}$  at each step in Definition 2.4 we note that no vertex in  $L$  is an internal vertex of  $T \cup T_{D,L}^{\text{root}}(x)$ . Hence  $(D, T \cup T_{D,L}^{\text{root}}(x), L)$  is a proper argument.

Consider the search tree  $ST$  that we obtain by running the algorithm  $\mathcal{A}(D, T, L)$ . First consider the case when  $ST$  consists of a single node. If  $\mathcal{A}(D, T, L)$  returns “NO” in step 1, then clearly we do not have a  $(T, L)$ -out-branching. Step 2 is valid by Lemma 2.3, i.e. it does not change the return of  $\mathcal{A}(D, T, L)$ . So now consider Step 3. As  $\ell_{\max}(D, T, L) \geq 1$  after step 1, and by Lemma 2.3 the value of  $\ell_{\max}(D, T, L)$  does not change by step 2 we note that  $\ell_{\max}(D, T, L) \geq 1$  before we perform step 3. Therefore there exists a  $(T, L)$ -out-branching in  $D$ . If  $|L| \geq k$  or  $|\text{Leaf}(T)| \geq k$  then, by Lemma 2.1, any  $(T, L)$ -out-branching in  $D$  has at least  $k$  leaves and the algorithm returns “YES”. If  $\text{Leaf}(T) \subseteq L$  then the only  $(T, L)$ -out-branching in  $D$  is  $T$  itself and as  $|\text{Leaf}(T)| < k$  the algorithm returns “NO” as it must do. Thus, the theorem holds when  $ST$  is just a node.

Now suppose that  $ST$  has at least two nodes and the theorem holds for all successors of the root  $R$  of  $ST$ . By the assumption that  $R$  makes further recursive calls, we have  $\ell_{\max}(D, T, L) \geq 1$  and there exists a vertex  $x \in \text{Leaf}(T) - L$ . If there is a  $(T, L)$ -out-branching with at least  $k$  leaves, then by Lemma 2.5 there is a  $(T, L \cup \{x\})$ -out-branching with at least  $k$  leaves or  $(T \cup T_{D,L}^{\text{root}}(x), L)$ -out-branching with at least  $k$  leaves. By induction hypothesis, one of  $B_1$  or  $B_2$  is “YES” and thus  $\mathcal{A}(D, T, L)$  correctly returns “YES”. Else if  $\ell_{\max}(D, T, L) < k$ , then again by Lemma 2.5 and induction hypothesis both  $B_1$  and  $B_2$  are “NO”. Therefore the theorem holds for the root  $R$  of  $ST$ , which completes the proof.  $\square$

### 3 Faster Algorithm

We now show how the algorithm from the previous section can be made faster by adding an extra vertex to the set  $L$  in certain circumstances. In order for this improvement to work, we need step (2) in the algorithm, which is new compared to the algorithm in [21]. We will also allow  $L$  to contain vertices which are not leaves of the current out-tree  $T$ . The improved algorithm is now described.

For every vertex  $x \in V(D)$ , do  $\mathcal{B}(D, \{x\}, \emptyset)$ .

If one of the returns of  $\mathcal{B}(D, \{x\}, \emptyset)$  is “YES” then output “YES”.

Otherwise, output “NO”.

$\mathcal{B}(D, T, L)$  :

(1) If  $\ell_{\max}(D, T, L) = 0$ , return “NO”. Stop.

- (2) While there is a  $x \in \text{Leaf}(T) - L$  such that  $\ell_{\max}(D, T, L \cup \{x\}) = 0$ , then add the arcs  $A_D^+(x)$  to  $T$ .
- (3) If  $|L| \geq k$ , return “YES”. Stop.  
If the number of leaves in  $T$  is at least  $k$ , return “YES”. Stop.  
If all leaves in  $T$  belong to  $L$ , return “NO”. Stop.
- (4) Choose a vertex  $x \in \text{Leaf}(T) - L$ , color  $x$  red and let  $H_x := \hat{D}$ .
  - (4.1) Let  $z$  be the nearest ancestor of  $x$  in  $T$  colored red, if it exists.
  - (4.2) Let  $L' := L \cup \{x\}$ .  
If  $z$  exists and  $T_z$  has exactly two leaves  $x$  and  $x'$  and  $x' \in L$  then:  
Let  $P = p_0 p_1 \dots p_r$  be a path in  $H_z - A_D^+(z)$  such that  $V(P) - V(T_z) = \{p_0\}$  and  $p_r \in N_D^+(z)$ , and let  $L' := L \cup \{p_0, x\}$ .
  - (4.3)  $B_1 := \mathcal{A}(D, T, L')$  and  $B_2 := \text{“NO”}$ .
  - (4.4) If  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \geq 2$  then let  $B_2 := \mathcal{A}(D, T \cup T_{D,L}^{\text{root}}(x), L)$ .
  - (4.5) Return “YES” if either  $B_1$  or  $B_2$  is “YES”. Otherwise return “NO”.

For the sake of simplifying the proofs below we furthermore assume that the above algorithm picks the vertex  $x$  in Step 4 in a depth first manner. That is, the vertex  $x$  is chosen to be the last vertex added to  $T$  such that  $x \in \text{Leaf}(T) - L$ .

**Theorem 3.1.** *Algorithm  $\mathcal{B}(D, T, L)$  works correctly. In other words,  $D$  has a  $(T, L)$ -out-branching with at least  $k$  leaves if and only if Algorithm  $\mathcal{B}(D, T, L)$  returns “YES”.*

*Proof.* The only difference between  $\mathcal{B}(D, T, L)$  and  $\mathcal{A}(D, T, L)$  is that in step (4.2) we may add an extra vertex  $p_0$  to  $L$  which was not done in  $\mathcal{A}(D, T, L)$ . We will now prove that this addition does not change the correctness of the algorithm.

So assume that we let  $L' = L \cup \{p_0, x\}$  in step (4.2) and that there is an optimal  $(T, L)$ -out-branching  $T'$  with  $x \in \text{Leaf}(T')$  but  $p_0 \notin \text{Leaf}(T')$ . We will show that this implies that an optimal solution is found in the branch of the search tree where we put  $z$  into  $L$ . This will complete the proof as if an optimal  $(T, L)$ -out-branching  $T'$  does not contain  $x$  as a leaf, by Lemma 2.5 it is found in  $\mathcal{A}(D, T \cup T_{D,L}^{\text{root}}(x), L)$  and if it includes both  $x$  and  $p_0$  as leaves then it is found in  $\mathcal{A}(D, T, L')$  (in step (4.3)).

Note that  $\text{Leaf}(T'_z) = \{x, x'\}$  as we put  $L' := L \cup \{p_0, x\}$  in step (4.2) and thus, we have that  $T_z$  had exactly two leaves  $x$  and  $x'$  and  $x' \in L$  and we have just assumed that  $x$  is a leaf of  $T'$ . Let  $D^* = D[V(T'_z) \cup \{p_0\} - \{z\}]$  and consider the following two cases.

If  $p_0$  can reach all vertices of  $D^*$  in  $D^*$  then proceed as follows. Let  $T^*$  be an out-branching in  $D^*$  with  $p_0$  as the root. Let  $T''$  be the out-branching obtained from  $T'$  by deleting all arcs in  $T'_z$  and adding all arcs in  $T^*$ . Note that  $|\text{Leaf}(T'')| \geq |\text{Leaf}(T')|$  as  $\text{Leaf}(T^*) \cup \{z\}$  are leaves in  $T''$  and  $\text{Leaf}(T'_z)$  are the only two leaves in  $T'$  which may not be leaves in  $T''$ . Therefore an optimal solution is found when we add  $z$  to  $L$ .

So now consider the case when  $p_0$  cannot reach all vertices of  $D^*$  in  $D^*$ . This means that there is a vertex  $u \in N_T^+(z)$  which cannot be reached by  $p_0$  in  $D^*$ . Let  $W = w_0 w_1 w_2 \dots w_l u$  be a path from the root of  $T$  to  $u$ , which does not use any arcs

out of  $z$  (which exists as  $z$  was colored red in step (4.1), so adding  $z$  to  $L$  at this stage would not destroy all out-branchings). Assume that  $a$  is chosen such that  $w_a \notin T'_z$  and  $\{w_{a+1}, w_{a+2}, \dots, w_l, u\} \subseteq V(T'_z)$ .

Consider the digraph  $D'' = D[V(T'_z) \cup \{p_0, w_a\} - \{z\}]$  and note that every vertex in  $D''$  can be reached by either  $p_0$  or  $w_a$  in  $D''$ . Therefore, there exists two vertex disjoint out-trees  $T_{p_0}$  and  $T_{w_a}$  rooted at  $p_0$  and  $w_a$ , respectively, such that  $V(T_{p_0}) \cup V(T_{w_a}) = V(D'')$  (to see that this claim holds add a new vertex  $y$  and two arcs  $yp_0$  and  $yw_a$ ). Furthermore since  $p_0$  cannot reach  $u$  in  $D^*$  we note that both  $T_{p_0}$  and  $T_{w_a}$  must contain at least two vertices. Let  $T'''$  be the out-branching obtained from  $T'$  by deleting all arcs in  $T'_z$  and adding all arcs in  $T_{p_0}$  and in  $T_{w_a}$ . Note that  $|\text{Leaf}(T''')| \geq |\text{Leaf}(T')|$  as  $\text{Leaf}(T_{p_0}) \cup \text{Leaf}(T_{w_a}) \cup \{z\}$  are leaves in  $T'''$  and  $\text{Leaf}(T'_z) \cup \{w_a\}$  are the only three vertices which may be leaves in  $T'$  but not in  $T'''$ . Therefore again an optimal solution is found when we add  $z$  to  $L$ .  $\square$

**Theorem 3.2.** *Algorithm  $\mathcal{B}(D, T, L)$  runs in time  $O(3.72^k n^{O(1)})$ .*

*Proof.* Consider the search tree  $ST$  that we obtain by running the algorithm  $\mathcal{B}(D, \{x\}, \emptyset)$ . That is, the root of  $ST$  is the triple  $(D, \{x\}, \emptyset)$ . The children of this root is  $(D, \{x\}, L')$  when we make a recursive call in step (4.3) and  $(D, T_{D,L}^{\text{root}}(x), \emptyset)$  if we make a recursive call in step (4.4). The children of these nodes are again triples corresponding to the recursive calls.

Let  $g(T, L)$  be the number of leaves in a subtree  $R$  of  $ST$  with triple  $(D, T, L)$ . Clearly,  $g(T, L) = 1$  when  $(D, T, L)$  is a leaf of  $ST$ . For a non-trivial subtree  $R$  of  $ST$ , we will prove, by induction, that  $g(T, L) \leq c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|}$ , where  $\alpha = 1.96$ ,  $\beta = 1.896$  and  $c \geq \alpha^2\beta^2$ . Assume that this holds for all smaller non-trivial subtrees. (Note that the value of  $c$  is chosen in such a way that in the inequalities in the rest of the proof, we have upper bounds for  $g(T^*, L^*)$  being at least 1 when  $(D, T^*, L^*)$  is a leaf of  $ST$ .)

Recall that  $x \in \text{Leaf}(T) - L$  was picked in step (4). Now consider the following possibilities.

If  $|L'| = |L| + 2$ , then the number of leaves of  $R$  is at most the following as if a call is made to  $\mathcal{B}(D, T \cup T_{D,L}^{\text{root}}(x), L)$  in (4.4) the the number of leaves of  $T$  increases by at least one:

$$\begin{aligned} g(T, L') + g(T \cup T_{D,L}^{\text{root}}(x), L) &\leq c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|-2} + c\alpha^{k-|\text{Leaf}(T)|-1}\beta^{k-|L|} \\ &= c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|} \left( \frac{1}{\beta^2} + \frac{1}{\alpha} \right) \\ &\leq c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|}. \end{aligned}$$

So we may assume that  $|L'| = |L| + 1$  in (4.3). Now assume that  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| \neq 2$  in (4.4). In this case either no recursive call is made in (4.4) or we increase the number of leaves in  $T$  by at least two. Therefore the number of leaves of  $R$  is at most

$$\begin{aligned} c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|-1} + c\alpha^{k-|\text{Leaf}(T)|-2}\beta^{k-|L|} &= c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|} \left( \frac{1}{\beta} + \frac{1}{\alpha^2} \right) \\ &\leq c\alpha^{k-|\text{Leaf}(T)|}\beta^{k-|L|}. \end{aligned}$$

So we may assume that  $|L'| = |L| + 1$  in (4.3) and  $|\text{Leaf}(T_{D,L}^{\text{root}}(x))| = 2$  in (4.4). Let  $T' = T \cup T_{D,L}^{\text{root}}(x)$  and consider the recursive call to  $\mathcal{B}(D, T', L)$ . If we increase the

number of leaves in  $T'$  in step (2) of this recursive call, then the number of leaves of the subtree of  $ST$  rooted at  $(D, T', L)$  is at most

$$c\alpha^{k-|\text{Leaf}(T')-1}\beta^{k-|L|-1} + c\alpha^{k-|\text{Leaf}(T')-2}\beta^{k-|L|} = c\alpha^{k-|\text{Leaf}(T')}\beta^{k-|L|}\left(\frac{1}{\alpha\beta} + \frac{1}{\alpha^2}\right).$$

Therefore, the number of leaves in  $R$  is at most

$$\begin{aligned} g(T, L') + g(T', L) &\leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|-1} + c\alpha^{k-|\text{Leaf}(T)-1}\beta^{k-|L|}\left(\frac{1}{\alpha\beta} + \frac{1}{\alpha^2}\right) \\ &= c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}\left(\frac{1}{\beta} + \frac{1}{\alpha^2\beta} + \frac{1}{\alpha^3}\right) \\ &\leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in step (2) when we consider  $(D, T', L)$ . Let  $y$  and  $y'$  denote the two leaves of  $T'_x$  (after possibly adding some arcs in step (2)). Consider the recursive call to  $\mathcal{B}(D, T', L \cup \{y\})$ . If we increase the number of leaves of  $T'$  in step (2) in this call then the number of leaves in  $R$  is at most

$$\begin{aligned} g(T, L \cup \{x\}) + g(T', L \cup \{y\}) + g(T' \cup (T')_{D,L}^{\text{root}}(y), L) \\ \leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}\left(\frac{1}{\beta} + \left(\frac{1}{\alpha\beta^2} + \frac{1}{\alpha^3\beta}\right) + \frac{1}{\alpha^2}\right) \\ \leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in step (2) when we consider  $(D, T', L \cup \{y\})$ . However in this case we note that  $|L'| = |L| + 2$  in this recursive call as when we consider  $y'$  the conditions of (4.3) are satisfied. So in this last case the number of leaves in  $R$  is at most

$$\begin{aligned} g(T, L \cup \{x\}) + g(T', L \cup \{y\}) + g(T' \cup (T')_{D,L}^{\text{root}}(y), L) \\ \leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}\left(\frac{1}{\beta} + \left(\frac{1}{\alpha\beta^3} + \frac{1}{\alpha^2\beta}\right) + \frac{1}{\alpha^2}\right) \\ \leq c\alpha^{k-|\text{Leaf}(T)}\beta^{k-|L|}. \end{aligned}$$

The depth of  $ST$  is at most  $2k$ , as we increase either  $|L|$  or  $|\text{Leaf}(T)|$  whenever we consider a child in the search tree and no non-leaf has  $|L| \geq k$  or  $|\text{Leaf}(T)| \geq k$ . Therefore, the number of nodes in  $ST$  is at most  $O(k\alpha^k\beta^k) = O(k3.72^k)$ . As the amount of work we do in each recursive call is polynomial we get the desired time bound.  $\square$

## 4 Linear Kernel for DIRECTED $k$ -LEAF restricted to Acyclic Digraphs

Lemma 1.1 implies that an acyclic digraph  $D$  has an out-branching if and only if  $D$  has a single vertex of in-degree zero. Since it is easy to check that  $D$  has a single vertex of in-degree zero, in what follows, we assume that the acyclic digraph  $D$  under consideration has a single vertex  $s$  of in-degree zero.

We start from the following simple lemma.

**Lemma 4.1.** *In an acyclic digraph  $H$  with a single source  $s$ , every spanning subgraph of  $H$ , in which each vertex apart from  $s$  has in-degree 1, is an out-branching.*

Let  $B$  be an undirected bipartite graph with vertex bipartition  $(V', V'')$ . A subset  $S$  of  $V'$  is called a *bidomination set* if for each  $y \in V''$  there is an  $x \in S$  such that

$xy \in E(B)$ . The so-called *greedy covering algorithm* [4] proceeds as follows: Start from the empty bidominating set  $C$ . While  $V'' \neq \emptyset$  do the following: choose a vertex  $v$  of  $V'$  of maximum degree, add  $v$  to  $C$ , and delete  $v$  from  $V'$  and the neighbors of  $v$  from  $V''$ .

The following lemma have been obtained independently by several authors, see Proposition 10.1.1 in [4].

**Lemma 4.2.** *If the minimum degree of a vertex in  $V''$  is  $d$ , then the greedy covering algorithm finds a bidominating set of size at most  $1 + \frac{|V_1|}{d} \left(1 + \ln \frac{d|V_2|}{|V_1|}\right)$ .*

Let  $D$  be an acyclic digraph with a single source. We use the following reduction rules to get rid of some vertices of in-degree 1.

- (A) If  $D$  has an arc  $a = xy$  with  $d^+(x) = d^-(y) = 1$ , then contract  $a$ .
- (B) If  $D$  has an arc  $a = xy$  with  $d^+(x) \geq 2$ ,  $d^-(y) = 1$  and  $x \neq s$ , then delete  $x$  and add arc  $uv$  for each  $u \in N^-(x)$  and  $v \in N^+(x)$ .

The reduction rules are of interest due to the following:

**Lemma 4.3.** *Let  $D^*$  be the digraph obtained from an acyclic digraph  $D$  with a single source using Reduction Rules A and B as long as possible. Then  $D^*$  has a  $k$ -out-branching if and only if  $D$  has one.*

*Proof.* Let  $D$  have an arc  $a = xy$  with  $d^+(x) = d^-(y) = 1$  and let  $D'$  be the digraph obtained from  $D$  by contracting  $a$ . Let  $T$  be a  $k$ -out-branching of  $D$ . Clearly,  $T$  contains  $a$  and let  $T'$  be an out-branching obtained from  $T$  by contracting  $a$ . Observe that  $T'$  is also a  $k$ -out-branching whether  $y$  is a leaf of  $D$  or not. Similarly, if  $D'$  has a  $k$ -out-branching, then  $D$  has one, too.

Let  $D$  have an arc  $a = xy$  with  $d^+(x) \geq 2$ ,  $d^-(y) = 1$  and  $x \neq s$  and let  $D'$  be obtained from  $D$  by applying Rule B. We will prove that  $D'$  has a  $k$ -out-branching if and only if  $D$  has one. Let  $T$  be a  $k$ -out-branching in  $D$ . Clearly,  $T$  contains arc  $xy$  and  $x$  is not a leaf of  $T$ . Let  $U$  be the subset of  $N^+(x)$  such that  $xu \in A(T)$  for each  $u \in U$  and let  $v$  be the vertex such that  $vx \in A(T)$ . Then the out-branching  $T'$  of  $D'$  obtained from  $T$  by deleting  $x$  and adding arcs  $vu$  for every  $u \in U$  has at least  $k$  leaves ( $T'$  is an out-branching of  $D'$  by Lemma 4.1). Similarly, if  $D'$  has a  $k$ -out-branching, then  $D$  has one, too.  $\square$

Now consider  $D^*$ . Let  $B$  be an undirected bipartite graph, with vertex bipartition  $(V', V'')$ , where  $V'$  is a copy of  $V(D^*)$  and  $V''$  is a copy of  $V(D^*) - \{s\}$ . We have  $E(B) = \{u'v'' : u' \in V', v'' \in V'', uv \in A(D^*)\}$ .

**Lemma 4.4.** *Let  $R$  be a bidominating set of  $B$ . Then  $D^*$  has an out-branching  $T$  such that the copies of the leaves of  $T$  in  $V'$  form a superset of  $V' - R$ .*

*Proof.* Consider a subgraph  $Q$  of  $B$  obtained from  $B$  by deleting all edges apart from one edge between every vertex in  $V''$  and its neighbor in  $R$ . By Lemma 4.1,  $Q$  corresponds to an out-branching  $T$  of  $D^*$  such that the copies of the leaves of  $T$  in  $V'$  form a superset of  $V' - R$ .  $\square$

**Theorem 4.5.** *If  $D^*$  has no  $k$ -out-branching, then the number  $n^*$  of vertices in  $D^*$  is less than  $6.6(k + 2)$ .*

*Proof.* Suppose that  $n^* \geq 6.6(k + 2)$ ; we will prove that  $D^*$  has a  $k$ -out-branching. Observe that by Rules A and B, all vertices of  $D^*$  are of in-degree at least 2 apart from  $s$  and some of its out-neighbors. Let  $X$  denote the set of out-neighbors of  $s$  of in-degree 1 and let  $X''$  be the set of copies of  $X$  in  $V''$ . Observe that the vertices of  $V'' - X''$  of  $B - X''$  are all of degree at least 2. Thus, by Lemma 4.2,  $B - X''$  has a bidominating set  $S$  of size at most  $\frac{n^*}{2}(1 + \ln 2) + 1$ . Hence,  $S \cup \{s\}$  is a bidominating set of  $B$  and, by Lemma 4.4,  $D^*$  has a  $b$ -out-branching with  $b \geq n^* - \frac{n^*}{2}(1 + \ln 2) - 2$ . It is not difficult to see that  $b \geq \frac{n^*}{2}(1 - \ln 2) - 2 \geq 0.153n^* - 2 \geq k$ .  $\square$

## 5 Open Problems

It would be interesting to see whether DIRECTED  $k$ -LEAF admits an algorithm of significantly smaller running time, say  $O(3^k n^{O(1)})$ . Another interesting and natural question is to check whether a linear-size kernel exists for ROOTED DIRECTED  $k$ -LEAF (for all digraphs).

**Acknowledgements** Research of Gutin, Kim and Yeo was supported in part by an EPSRC grant.

## References

- [1] N. Alon, F.V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized Algorithms for Directed Maximum Leaf Problems. *Proc. 34th ICALP*, Lect. Notes Comput. Sci. 4596 (2007), 352–362.
- [2] N. Alon, F.V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Better Algorithms and Bounds for Directed Maximum Leaf Problems. *Proc. 27th Conf. Foundations Software Technology and Theoretical Computer Science*, Lect. Notes Comput. Sci. 4855 (2007), 316–327.
- [3] N. Alon, F.V. Fomin, G. Gutin, M. Krivelevich and S. Saurabh, Spanning directed trees with many leaves. To appear in *SIAM J. Discrete Math.*
- [4] A.S. Asratian, T.M.J. Denley, and R. Häggkvist, *Bipartite Graphs and Their Applications*, Univ. Press, Cambridge, 1998.
- [5] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London, 2000.
- [6] H.L. Bodlaender, R.G. Downey, M.R. Fellows and D. Hermelin. On Problems without Polynomial Kernels. *Lect. Notes Comput. Sci.* 5125 (2008), 563–574.
- [7] P.S. Bonsma, T. Brueggermann and G.J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. *Lect. Notes Comput. Sci.* 2747 (2003), 259–268.
- [8] P.S. Bonsma and F. Dorn. An FPT algorithm for directed spanning  $k$ -leaf, 2007. <http://arxiv.org/abs/0711.4052>
- [9] P.S. Bonsma and F. Dorn. Tight bounds and faster algorithms for Directed Max-Leaf. *Proc. 16th ESA*, Lect. Notes Comput. Sci. 5193 (2008), 222–233.

- [10] G. Ding, Th. Johnson, and P. Seymour. Spanning trees with many leaves. *J. Graph Theory* 37 (2001), 189–197.
- [11] R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer, 1999.
- [12] M. Drescher and A. Vetta. An approximation algorithm for the maximum leaf spanning arborescence problem. To appear in *ACM Transactions on Algorithms*.
- [13] V. Estivill-Castro, M.R. Fellows, M.A. Langston, and F.A. Rosamond, FPT is P-Time Extremal Structure I. Proc. ACiD'05, College Publications, London (2005), 1–41.
- [14] M.R. Fellows, C. McCartin, F.A. Rosamond, and U. Stege. Coordinated kernels and catalytic reductions: An improved FPT algorithm for max leaf spanning tree and other problems. *Lect. Notes Comput. Sci.* 1974 (2000), 240–251.
- [15] H. Fernau, F.V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger, Kernel(s) for problems with no kernel: on out-trees with many leaves. Submitted.
- [16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer, 2006.
- [17] F.V. Fomin, F. Grandoni and D. Kratsch. Solving Connected Dominating Set Faster Than  $2^n$ . *Algorithmica* 52 (2008), 153–166.
- [18] G. Galbiati, A. Morzenti, and F. Maffioli. On the approximability of some maximum spanning tree problems. *Theor. Computer Sci.* 181 (1997), 107–118.
- [19] J. R. Griggs and M. Wu. Spanning trees in graphs of minimum degree four or five. *Discrete Math.* 104 (1992), 167–183.
- [20] D.J. Kleitman and D.B. West. Spanning trees with many leaves. *SIAM J. Discrete Math.* 4 (1991), 99–106.
- [21] J. Kneis, A. Langer and P. Rossmanith. A new algorithm for finding trees with many leaves. To appear in Proc. ISAAC 2008.
- [22] N. Linial and D. Sturtevant. Unpublished result (1987).
- [23] H.I. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *J. Algorithms* 29 (1998), 132–141.
- [24] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, 2006.
- [25] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. *Lect. Notes Comput. Sci.* 1461 (1998), 441–452.