

On the Parameterized Max-Leaf Problems: Digraphs and Undirected Graphs*

JIANER CHEN and YANG LIU
Department of Computer Science
Texas A&M University
College Station, TX 77843, USA
{chen,yangliu}@cs.tamu.edu

Abstract

The parameterized MAX-LEAF problem on undirected graphs (which is also named the MAX-LEAF SPANNING-TREE problem) is formulated as follows: given an undirected graph G and a parameter k , either construct a spanning tree with at least k leaves for G or report ‘No’ if such a tree does not exist. The problem also has a version for directed graphs that is named the MAX-LEAF OUT-BRANCHING problem. In this paper, we present a simple branch-and-search algorithm of running time $O^*(4^k)$ that solves the MAX-LEAF OUT-BRANCHING problem. This significantly improves the previous best algorithm for the problem that runs in time $O^*(2^{O(k \log k)})$. Our main contributions consist of new observations on the combinatorial structures of the problem and the introduction of a new algorithmic technique that provides new perspectives for design and analysis of parameterized algorithms. Our algorithm of running time $O^*(4^k)$ is also applicable to the simpler MAX-LEAF SPANNING-TREE problem, improving the previous best algorithm of running time $O^*(6.75^k)$ for the problem.

1 Introduction

The MAX-LEAF SPANNING-TREE problem is to find a spanning tree with the maximum number of leaves in an undirected graph. The problem also has a version for directed graphs (i.e., digraphs), which is named the MAX-LEAF OUT-BRANCHING problem. The problems are of importance in many theoretical and practical applications [16, 20, 21]. The problems are NP-hard [15]. In terms of polynomial time approximability, the MAX-LEAF SPANNING-TREE problem is APX-hard [14], and can be approximated with a ratio 2 in polynomial time [17, 19], while very recent research shows that the MAX-LEAF OUT-BRANCHING problem can be approximated with a ratio $O(\sqrt{n})$ in polynomial time [10].

In this paper, we are concentrated on the study of parameterized complexity of the problems, for both undirected graphs and digraphs. Formally, the (parameterized) MAX-LEAF SPANNING-TREE problem is for a given undirected graph G and a parameter k , to either construct a spanning tree with at least k leaves for G or report ‘No’ if such a tree does not exist. For the version for digraphs, we define an *out-tree* to be an acyclic digraph T with a specific vertex r (the *root*) such that for every vertex v in T , there is a unique (directed) path in T from r to v . The vertices of out-degree 0 in the out-tree T are called the *leaves*. An *out-branching* B (rooted at a vertex r) of a digraph G is a subgraph of G such that B is an out-tree (rooted at r) and that B contains all vertices of G . The (parameterized) MAX-LEAF OUT-BRANCHING problem is for a given digraph G and a parameter k , to either construct an out-branching with at least k leaves for G or report ‘No’ if such an out-branching does not exist.

The MAX-LEAF SPANNING-TREE and MAX-LEAF OUT-BRANCHING problems have been extensively studied. In particular, there has been an impressive list of improved algorithms for the MAX-LEAF SPANNING-TREE problem. The first algorithm for the problem was proposed by Bodlaender and runs in time $O^*((17k^4)!)^3$.¹ Downey and Fellows developed an improved algorithm of running time $O^*((2k)^{4k})$ [12], which was later improved by Fellows *et al.*, who presented an algorithm of running time $O^*(14.23^k)$ [13]. Bonsma, Brueggemann, and Woeginger developed a further improved algorithm of running time $O^*(9.49^k)$ [4]. Currently, the

*Supported in part by the US National Science Foundation under the Grant CCF-0430683.

¹Following the recent convention, we will use $O^*(t)$ to denote the bound $O(t \cdot n^{O(1)})$.

best algorithm for the MAX-LEAF SPANNING-TREE problem is due to Bonsma and Zickfeld, whose running time is bounded by $O^*(6.75^k)$ [7].

The more general MAX-LEAF OUT-BRANCHING problem seems much more difficult. The problem has become significantly interesting very recently because of its rich combinatorial structures and challenging algorithmic issues. Some recent results in this research direction have not been officially published but are only available on internet. Alon *et al.* considered the problem, and developed an algorithm of running time $O^*(2^{O(k^2 \log k)})$ for the MAX-LEAF OUT-BRANCHING problem on strongly connected digraphs and on acyclic digraphs [1]. The same research group subsequently improved the results with an algorithm of running time $O^*(2^{O(k \log^2 k)})$ for strongly connected digraphs and an algorithm of running time $O^*(2^{O(k \log k)})$ for acyclic digraphs. The parameterized complexity of the MAX-LEAF OUT-BRANCHING problem on general digraphs (that is, whether the problem is fixed-parameter tractable, i.e., solvable in time $f(k)n^{O(1)}$ for a fixed function f) was proposed as an open problem in [17, 1, 2]. The open problem was resolved by Bonsma and Dorn [5] who presented an algorithm of running time $O^*(2^{O(k^3 \log k)})$ for the problem [5], which was further improved by the same authors with an algorithm of running time $O^*(2^{O(k \log k)})$ for the problem [6].

In this paper, we present a simple branch-and-search algorithm of running time $O^*(4^k)$ that solves the MAX-LEAF OUT-BRANCHING problem. This significantly improves the previous best algorithm for the problem that runs in time $O^*(2^{O(k \log k)})$ [6]. Our improved algorithm is based on a careful study of the combinatorial structures of digraphs and on effective applications of a new technique for the design and analysis of parameterized algorithms. Our combinatorial analysis reveals a close relationship between out-trees and out-branchings of a digraph, and our new algorithmic technique identifies two indirected branching measures that limit the number of branchings that are unable to effectively reduce the parameter value k in a parameterized branch-and-search process.

Before closing this introductory section, we remark that the MAX-LEAF SPANNING-TREE problem can be trivially reduced to the MAX-LEAF OUT-BRANCHING problem if we convert the input undirected graph G into a digraph by replacing each undirected edge $[u, v]$ in G by two directed edges $[u, v]$ and $[v, u]$. Therefore, our algorithm of running time $O^*(4^k)$ for the MAX-LEAF OUT-BRANCHING problem can be directly used to solve the MAX-LEAF SPANNING-TREE problem, improving the previous best algorithm of running time $O^*(6.75^k)$ for the MAX-LEAF SPANNING-TREE problem [7].

2 Preliminaries

All graphs in our discussion are simple graphs, i.e., there are no multiple edges from a vertex to another vertex, and no self-loops on any vertex. For an edge $[x, y]$ in a digraph G , the vertex x is called an *in-neighbor* of the vertex y , and the vertex y is called an *out-neighbor* of the vertex x . The *in-degree* of a vertex x in G is the number of in-neighbors of x , and the *out-degree* of a vertex x is the number of out-neighbors of x .

A *path* P in a digraph G from a vertex x_1 to another vertex x_q is a sequence of vertices $P = \{x_1, \dots, x_q\}$ such that $[x_i, x_{i+1}]$ is an edge in G for all $1 \leq i \leq q - 1$. The path P is *simple* if no vertex is repeated in P . A vertex x can *reach* another vertex y (or equivalently, the vertex y is reachable from the vertex x) in a digraph G if there is a path from x to y in G .

An acyclic digraph T is an *out-tree* (rooted at a vertex r) if the vertex r has in-degree 0 and for every vertex x in T there is a unique path from r to x . The vertex r is the *root* of the out-tree T , and each vertex of out-degree 0 is a *leaf* of T . A vertex x in T is an *internal vertex* if it is not a leaf. A *k-out-tree* is an out-tree that has at least k leaves. An out-tree *in a digraph* G is a subgraph of G that is an out-tree. An *out-branching* of a digraph G is an *out-tree* of G that contains all vertices of G . A *k-out-branching* is an out-branching that has at least k leaves.

Let T be an out-tree in a digraph G , and let l be a leaf of T . An *out-chain* $\pi(l)$ of T is a simple path $\pi(l) = \{x_0, x_1, \dots, x_q\}$ in G , where $x_0 = l$, such that for all $0 \leq i \leq q - 1$, x_{i+1} is the only out-neighbor of x_i in $G - \{T \cup \{x_1, \dots, x_i\}\}$. A *y-truncated* out-chain $\pi_y(l)$ of an out-chain $\pi(l)$, where $y \neq l$, is defined as follows: (1) if $y \in \pi(l)$, then $\pi_y(l)$ is the path from l to z in $\pi(l)$, where z is the in-neighbor of y in $\pi(l)$; and (2) if $y \notin \pi(l)$, then $\pi_y(l) = \pi(l)$. Inductively, $\pi_{y_1, \dots, y_q}(l)$ is the y_q -truncated out-chain of $\pi_{y_1, \dots, y_{q-1}}(l)$. For a given set Q of out-chains of the out-tree T , a leaf l of T is *determined* if an out-chain $\pi(l)$ is included in Q .

Let T be an out-tree in a digraph G with a given set $\{\pi(l_1), \dots, \pi(l_p)\}$ of out-chains, and let k be a positive integer. We say that $(T; \pi(l_1), \dots, \pi(l_p); k)$ is *extendable* if there exists a k -out-branching T_s such that (1) T_s and T have the same root, (2) T is a subgraph of T_s , and (3) all vertices reachable in T_s from l_i are in $\pi(l_i)$. Such an out-branching T_s is called a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -*extended out-branching*.

We formulate the following problem.

EXTENDING($T; \pi(l_1), \dots, \pi(l_p); k$): given $(T; \pi(l_1), \dots, \pi(l_p); k)$, either construct a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching or report ‘No’ if no such an out-branching exists.

The following reduction will play an important role in our discussion.

Definition EXTENDING($T; \pi(l_1), \dots, \pi(l_p); k$) is *reducible to* EXTENDING($T'; \pi'(l_1), \dots, \pi'(l_q); k'$) if

- (1) any $(T'; \pi'(l_1), \dots, \pi'(l_q); k')$ -extended out-branching is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching;
- (2) $(T'; \pi'(l_1), \dots, \pi'(l_q); k')$ is extendable when $(T; \pi(l_1), \dots, \pi(l_p); k)$ is extendable.

It is easy to verify that the above reduction is a transitive relation.

In the rest of this section, we give two lemmas on out-trees and out-branchings, which will be used in our later discussion.

Lemma 2.1 *Let T and T' be two out-trees in a digraph with the same root such that T' is a subgraph of T , and let l be a leaf of T' and $y \neq l$ be a vertex in T' . Then l cannot reach y in T .*

PROOF. Since y is a vertex of T' and T' is an out-tree, there is a unique path P_1 in T' from the root r of T' to y , which obviously does not pass through the leaf l because l has out-degree 0 in T' . Since T' is a subgraph of T , P_1 is also a path from r to y in T . Now if l can reach y in T , then the unique path in T from r to l followed by the path from l to y in T forms a second path P_2 from r to y in T that passes through the vertex l . This contradicts the assumption that T is an out-tree since r is also the root of T . \square

Lemma 2.2 *Let T_s be an out-branching and T be a k -out-tree in a digraph such that T_s and T have the same root r and T is a subgraph of T_s . Then T_s is a k -out-branching.*

PROOF. Let $\{l_1, \dots, l_p\}$ be the set of all leaves of T . For each $1 \leq i \leq p$, let L_i be the set of leaves of T_s that can be reached from l_i in T_s . Note that each set L_i contains at least one vertex in T_s .

We prove that no two sets L_i and L_j share a common vertex. Suppose that $w \in L_i \cap L_j$. Then the unique path P'_i in T (which is also the unique path in T_s) from the root r to l_i followed by a path P''_i from l_i to w in T_s forms a path P_i in T_s from the root r to w . Similarly, there is a path P_j in T_s from the root r to w that passes through the vertex l_j . Since T_s is an out-tree, we must have $P_i = P_j$. In consequence, the path P_i in T_s contains the vertex l_j . Since both l_i and l_j are leaves in T , the unique path P'_i in T from r to l_i does not contain the vertex l_j . Therefore, the vertex l_j must be contained in the path P''_i from l_i to w , so l_i can reach l_j in T_s . However, this is impossible according to Lemma 2.1.

Therefore, T_s has at least $|L_1| + \dots + |L_p| \geq p$ leaves. Since T is a k -out-tree, $p \geq k$. Thus, T_s is a k -out-branching. \square

3 Extending an out-tree

Throughout the discussion of this section, we let T be an out-tree with the root r in a digraph G with a given set $\{\pi(l_1), \dots, \pi(l_p)\}$ of out-chains. Let x be a vertex in T that is not a determined leaf of T and let y be an out-neighbor of x in $G - T$. Define $T_{xy} = T \cup [x, y]$. In subsection 3.1, we discuss some properties that are useful for reducing the problem EXTENDING($T; \pi(l_1), \dots, \pi(l_p); k$) to the problem EXTENDING($T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k$). In subsection 3.2, we show that the results in subsection 3.1 can be generalized to the case where many out-neighbors of x in $G - T$ are considered. Furthermore, if T is already a k -out-tree, then we can solve the problem EXTENDING($T; \pi(l_1), \dots, \pi(l_p); k$) in polynomial time, which gives a boundary condition for our algorithm.

3.1 Properties for extending an out-tree

We first verify that T_{xy} is an out-tree in G and that the y -truncated out-chains $\pi_y(l_1), \dots, \pi_y(l_p)$ are out-chains of T_{xy} . This is necessary before we can discuss the extendability of $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$.

Lemma 3.1 *T_{xy} is an out-tree with root r in the digraph G .*

PROOF. Since the vertex y has out-degree 0 in T_{xy} , it cannot help creating any new path in T_{xy} from the root r to any vertex $w \neq y$ in T_{xy} . Thus, there is a unique path from the root r to w in T_{xy} . Moreover,

since x is the only in-neighbor of y in T_{xy} , the only path from the root r to y in T_{xy} must be the unique path from r to x followed by the edge $[x, y]$. This proves that T_{xy} is an out-tree. \square

Lemma 3.2 *The truncated out-chains $\pi_y(l_1), \dots, \pi_y(l_p)$ are out-chains of T_{xy} .*

PROOF. Pick any out-chain of T : $\pi(l_i) = \{u_0, u_1, \dots, u_q\}$, where $u_0 = l_i$. Then (1) $T \cap \pi(l_i) = \{u_0\}$, and (2) u_j has a unique out-neighbor u_{j+1} in $G - (T \cup \{u_0, \dots, u_j\})$ for all $0 \leq j \leq q - 1$. By (1), $y \notin T$, and the definition of y -truncated out-chain, $T \cap \pi_y(l_i) = \{u_0\}$. By the definition of $\pi_y(l_i)$, $y \notin \pi_y(l_i)$, so $T_{xy} \cap \pi_y(l_i) = \{u_0\}$. Then by (2) and because T is a subgraph of T_{xy} , u_j has a unique out-neighbor in $G - (T_{xy} \cup \{u_0, \dots, u_j\})$ for any $u_j \in \pi_y(l_i)$. So $\pi_y(l_i)$ is an out-chain of T_{xy} for all $1 \leq i \leq p$. \square

Lemma 3.3 *Any $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching.*

PROOF. Let T_s be a $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching. By the definition we have: (1) T_s is a k -out-branching; (2) T_{xy} and T_s has the same root r ; (3) T_{xy} is a subgraph of T_s ; and (4) all vertices reachable in T_s from l_i are in $\pi_y(l_i)$. We show T_s is also a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching.

By Lemma 3.1, T and T_{xy} have the same root. Thus, by (2), T and T_s have the same root. Since T is a subgraph of T_{xy} , by (3) T is also a subgraph of T_s . Finally, since $\pi_y(l_i) \subseteq \pi(l_i)$, by (4), all vertices reachable in T_s from l_i are in $\pi(l_i)$. In conclusion, T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. \square

In the following, we assume that $(T; \pi(l_1), \dots, \pi(l_p); k)$ is extendable and T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. Let z be the (unique) in-neighbor of y in T_s . Let T'_s be the graph obtained from T_s by first deleting the edge $[z, y]$ then adding the edge $[x, y]$. We will prove that T'_s is a $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching. We first prove the following properties.

Lemma 3.4 *T_{xy} is a subgraph of T'_s .*

PROOF. Since T is a subgraph of T_s , and $y \notin T$ so the edge $[z, y]$ is not in T , all edges in T are in $T'_s = (T_s - [z, y]) \cup [x, y]$. Moreover, by the definition of T'_s , the edge $[x, y]$ is contained in T'_s . Therefore, all edges in $T_{xy} = T \cup [x, y]$ are in T'_s , and the out-tree T_{xy} is a subgraph of T'_s . \square

Lemma 3.5 *T'_s is an out-branching of G with root r . Moreover, if x is not a leaf of T , then T'_s is an k -out-branching of G with root r .*

PROOF. Since T_s is an out-branching of the digraph G , by the definition, T'_s is also rooted at r and contains all vertices in G . For any vertex w in G , if the unique path P_w in T_s from the root r to w does not pass through the vertex y , then P_w is also a path in T'_s . On the other hand, if the path P_w from r to w is a concatenation of a path P'_w from r to y and a path P''_w from y to w , then the path in T from r to x (note that $[z, y] \notin T$) followed by the edge $[x, y]$ then by the path P''_w makes a path in T'_s from r to w . The uniqueness of the path in T'_s from r to w can be easily verified because each vertex in T'_s has in-degree bounded by 1.

If x is not a leaf of T_s , then deleting the edge $[z, y]$ then adding the edge $[x, y]$ cannot decrease the number of vertices of degree 0 in the out-branching. Thus, T'_s has at least as many leaves as that of T_s . Since T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching, T_s has at least k leaves. In consequence, T'_s is a k -out-branching. \square

Lemma 3.6 *The vertices reachable in T'_s from a determined leaf l_i are all in $\pi_y(l_i)$.*

PROOF. We first prove that the vertices reachable in T'_s from the determined leaf l_i are all in $\pi(l_i)$. Suppose that l_i can reach a vertex $w \notin \pi(l_i)$ in T'_s via a path P . Then the edge $[x, y]$ must be in P : otherwise, the path P is also in T_s and l_i can reach $w \notin \pi(l_i)$ in T_s , contradicting the assumption that T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. Thus, l_i can reach x in T_s , which is impossible by Lemma 2.1 because both l_i and x are in T , and l_i is a determined leaf of T but x is not a determined leaf. This contradiction proves that all vertices reachable in T'_s from l_i must be in $\pi(l_i)$.

Now we prove that the vertices reachable in T'_s from l_i are all in $\pi_y(l_i)$. If $y \notin \pi(l_i)$, then $\pi_y(l_i) = \pi(l_i)$, and the lemma is proved by the above analysis. Suppose $y \in \pi(l_i)$ and l_i can reach a vertex $w \in \pi(l_i) - \pi_y(l_i)$ via a path P' . Since l_i is a leaf of T , by Lemma 2.1 any vertex $u \neq l_i$ of T cannot be in P' . Thus, $\pi_y(l_i) \cup \{y\}$ must be in P' . However, since (1) both l_i and y are in T_{xy} , (2) l_i is a leaf of T_{xy} , (3) T'_s have the same root as T by Lemmas 3.1 and 3.5, and (4) T_{xy} is a subgraph of T'_s by Lemma 3.4, therefore, l_i can not reach y in T'_s by Lemma 2.1. Thus, all vertices reachable from l_i in T'_s are in $\pi_y(l_i)$ when $y \in \pi(l_i)$. \square

3.2 Extending an out-tree

Now, we are ready to show that we can extend T_{xy} instead of T . Note that T_{xy} can be found in polynomial time, and this can occur at most n times because T_{xy} has one more vertex than T . Thus, this is an efficient operation without branching.

Lemma 3.7 *Suppose x is an internal vertex of T . Then $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$.*

PROOF. If $(T; \pi(l_1), \dots, \pi(l_p); k)$ is extendable, then by Lemma 3.5, T'_s is a k -out-branching with root r . Thus T'_s and T_{xy} have the same root according to Lemma 3.1, and T_{xy} is a subgraph of T'_s by Lemma 3.4. Finally, all vertices reachable in T'_s from a determined leaf l_i are in $\pi_y(l_i)$ by Lemma 3.6. In consequence, T'_s is a $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching, i.e., $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ is extendable.

By Lemma 3.3, any $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. This completes the proof of the lemma. \square

Lemma 3.7 can be generalized to the case where we add many out-neighbors of the vertex x in $G - T$ to the out-tree T . For this, let y_1, \dots, y_q be out-neighbors of x in $G - T$, and let $T(y_1, \dots, y_q)$ the tree T plus the edges $[x, y_1], \dots, [x, y_q]$.

Lemma 3.8 *Suppose that x is an internal vertex of T . Then $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T(y_1, \dots, y_q); \pi_{y_1, \dots, y_q}(l_1), \dots, \pi_{y_1, \dots, y_q}(l_p); k)$.*

PROOF. We prove the lemma by induction on q . The initial case $q = 1$ is proved by Lemma 3.7.

Now suppose that when $q = i$, the problem $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to the problem $\text{EXTENDING}(T(y_1, \dots, y_i); \pi_{y_1, \dots, y_i}(l_1), \dots, \pi_{y_1, \dots, y_i}(l_p); k)$. For $q = i + 1$, note that $T(y_1, \dots, y_i, y_{i+1})$ is the out-tree $T(y_1, \dots, y_i)$ plus the edge $[x, y_{i+1}]$ where $y_{i+1} \notin T(y_1, \dots, y_i)$, and the truncated out-chain $\pi_{y_1, \dots, y_i, y_{i+1}}(l_j)$ is a y_{i+1} -truncated out-chain of the out-chain $\pi_{y_1, \dots, y_i}(l_j)$ for the tree $T(y_1, \dots, y_i)$ for all j . Moreover, the vertex x is obviously an internal vertex of the tree $T(y_1, \dots, y_i)$. Therefore, by Lemma 3.7 again, the problem $\text{EXTENDING}(T(y_1, \dots, y_i); \pi_{y_1, \dots, y_i}(l_1), \dots, \pi_{y_1, \dots, y_i}(l_p); k)$ is reducible to the problem $\text{EXTENDING}(T(y_1, \dots, y_i, y_{i+1}); \pi_{y_1, \dots, y_i, y_{i+1}}(l_1), \dots, \pi_{y_1, \dots, y_i, y_{i+1}}(l_p); k)$. Now the transitivity of the reduction proves that the problem $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to the problem $\text{EXTENDING}(T(y_1, \dots, y_i, y_{i+1}); \pi_{y_1, \dots, y_i, y_{i+1}}(l_1), \dots, \pi_{y_1, \dots, y_i, y_{i+1}}(l_p); k)$, and the induction goes through. \square

The conditions in the previous lemmas can be further relaxed without requiring that the vertex x be an internal vertex of T , if the out-tree T is a k -out-tree, as shown in the following lemmas.

Lemma 3.9 *Suppose that x is not a determined leaf of T and T is a k -out-tree. Then T_{xy} is a k -out-tree and $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$.*

PROOF. By Lemma 3.1, T_{xy} is an out-tree with root r . The number of leaves of T_{xy} cannot be less than that of T because adding the edge $[x, y]$ to T adds a vertex y of out-degree 0 and may change the out-degree of at most one vertex in T (i.e., x). Since T is a k -out-tree, T_{xy} is also a k -out-tree.

Next we show that $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$. By Lemma 3.3, any $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. Now suppose that $(T; \pi(l_1), \dots, \pi(l_p); k)$ is extendable. Let T'_s be a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. By Lemma 3.5, the graph T'_s constructed from T'_s in the lemma is an out-branching. Since T is a subgraph of T'_s by Lemma 3.4 and T is a k -out-tree, T'_s is a k -out-branching by Lemma 2.2. Also T'_s and T_{xy} have the same root r according to Lemma 3.1 and Lemma 3.5. Moreover, T_{xy} is a subgraph of T'_s by Lemma 3.4. Finally, all vertices reachable in T'_s by a determined leaf l_i are in $\pi_y(l_i)$ by Lemma 3.6. So T'_s is a $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ -extended out-branching, i.e., $(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$ is extendable.

Therefore, T_{xy} is a k -out-tree with root r and the problem $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to the problem $\text{EXTENDING}(T_{xy}; \pi_y(l_1), \dots, \pi_y(l_p); k)$. \square

Similarly, we can generalize Lemma 3.9 to many out-neighbors of the vertex x . For this, again let y_1, \dots, y_q be out-neighbors of x in $G - T$, and let $T(y_1, \dots, y_q)$ the tree T plus the edges $[x, y_1], \dots, [x, y_q]$.

Lemma 3.10 *Suppose that x is not a determined leaf of T and T is a k -out-tree. Then the problem $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T(y_1, \dots, y_q); \pi_{y_1, \dots, y_q}(l_1), \dots, \pi_{y_1, \dots, y_q}(l_p); k)$.*

PROOF. The lemma is proved by induction on q . The initial case $q = 1$ is proved by Lemma 3.9, and the inductive step is proved by Lemma 3.9 and the transitivity of the reduction. \square

Lemma 3.10 is very useful since it implies that by continuously adding out-neighbors of a undetermined leaf, a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching can be found if T is already a k -out-tree. This will be discussed in the following algorithm and its proof.

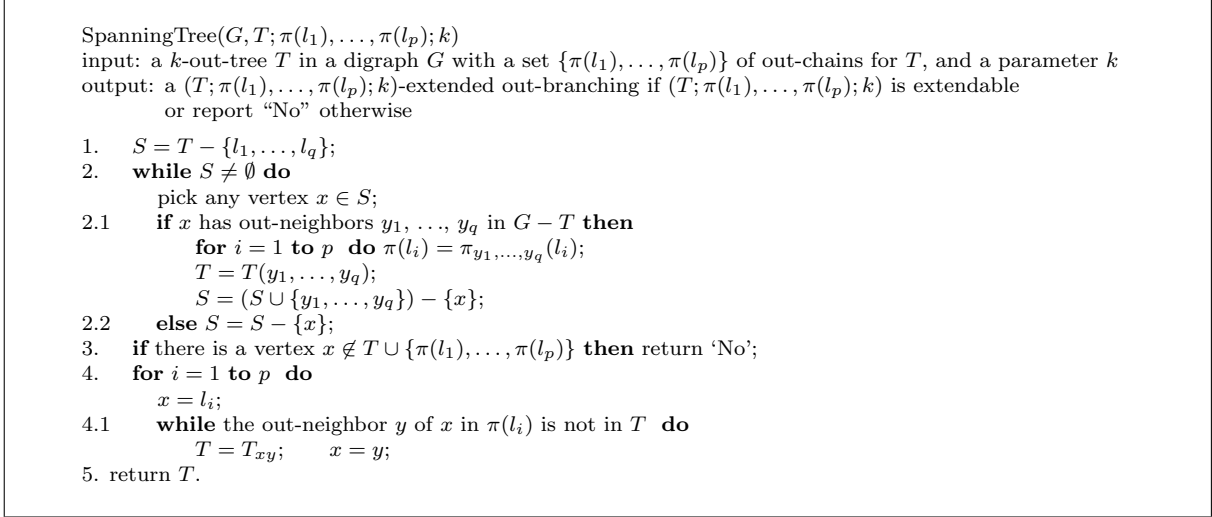


Figure 1: Algorithm 1. SpanningTree

Theorem 3.11 For a given k -out-tree T in a digraph G of n vertices, with a set $\{\pi(l_1), \dots, \pi(l_p)\}$ of out-chains for T , the algorithm $\text{SpanningTree}(G, T; \pi(l_1), \dots, \pi(l_p); k)$ runs in time $O(n^2)$ and
(1) either constructs a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching T_s if such one exists, or
(2) reports 'No' if $(T; \pi(l_1), \dots, \pi(l_p); k)$ is not extendable.

PROOF. We first prove the correctness of the algorithm.

To distinguish the given input and the dynamically changing values during the execution of the algorithm, let T_0 be the out-tree and $\pi_0(l_1), \dots, \pi_0(l_p)$ be the out-chains given to the algorithm before step 1.

Claim 1. After steps 1, 2.1, 2.2, and 2, T is a k -out-tree and $\text{EXTENDING}(T_0; \pi_0(l_1), \dots, \pi_0(l_p); k)$ is reducible to $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$.

Since step 1 does not change T and any out-chain, our claim is true after step 1. Since T is a k -out-tree before step 2.1, our claim still holds true after step 2.1 by Lemma 3.10 and the transitivity of the reduction. Finally, since steps 2.2 does not change T and any out-chain, the claim is true after step 2.2. By the transitivity of the reduction, our claim holds true after step 2. This proves Claim 1.

Claim 2. Let $x \notin T_0$. The vertex x is in T after step 2 if and only if there is a path P from the root r of T to x such that P contains no determined leaves.

Suppose that there is a path P from r to x that contains no determined leaves. If $x \notin T$, let $y \notin T$ be the vertex in P such that all vertices before y in P are in T , and let z be the in-neighbor of y in P . By the choice of y , $z \in T$. Since the path P contains no determined leaves, the vertex z is not a determined leaf. Thus, either z is in S in step 1 or is added to S in step 2.1. However, in either case the vertex y should have been added to T during step 2.1 when z is picked from S . This contradicts the choice of y . In consequence, the vertex x must be in T after step 2.

Now suppose that x is in T after step 2. During step 2.1, let x be the parent of y if y is added to S and T because of x . Initially, all vertices in S at step 1 are reachable from r without passing any determined leaf. At step 2.1, any vertex y added to S is reachable from r through the path from r to its parent x and edge $[x, y]$, i.e., without passing any determined leaf. By induction, for any vertex x in T after step 2, there is a path from r to x which contains no determined leaves. This proves Claim 2.

If there is a vertex $x \notin T \cup \{\pi(l_1), \dots, \pi(l_p)\}$ after step 2, then r cannot reach x without passing a determined leaf. This implies that some determined leaf l_i should reach the vertex $x \notin \pi(l_i)$ in any out-branching. This derives that $(T; \pi(l_1), \dots, \pi(l_p); k)$ is not extendable, i.e., $(T_0; \pi_0(l_1), \dots, \pi_0(l_p); k)$ is not extendable by Claim 1. Thus step 3 reports 'No' correctly.

After step 3, a vertex is in either T or $\pi(l_i)$ for some determined leaf l_i . Given an out-chain $\pi(l_i)$, let y be the vertex in $\pi(l_i)$ such that all the vertices but l_i before y in $\pi(l_i)$ are not in T . By Claim 2, any vertex x after y in $\pi(l_i)$ are in T since r can reach y and then x without passing any determined leaf. To prove that step 4 is correct, we prove the following claim.

Claim 3. Given an out-tree T with a set $\{\pi(l_1), \dots, \pi(l_p)\}$ of out-chains such that (1) a vertex $x \in G$ is either in T or in some out-chain, and (2) if a vertex $y \in \pi(l_i)$ is in T , then any vertex after y in $\pi(l_i)$ is also in T . Let T_s be the out-tree T after step 4. Then there is a distinct leaf in T_s corresponding to a leaf of T .

We prove the claim by induction on p . When $p = 1$, all vertices of G are in T after step 4.1. The last vertex visited at step 4.1 is not in T and is a leaf in T_s corresponding to l_1 and all the other leaves of T are still leaves of T_s . So our claim is true.

Suppose when $p = i$, our claim is true. For $p = i + 1$, let T' be the out-tree after execution of step 4.1 for l_1 . Similarly, the last vertex y visited at step 4.1 is not in T and is a leaf of T' . Also the other leaves of T are leaves of T' . If y is in some $\pi(l_j)$ where $j > 1$, then all vertices after y in $\pi(l_j)$ are in T after step 4.1. Also a vertex $x \in G$ is still either in T' or in some out-chain $\pi(l_j)$ for $2 \leq j \leq i + 1$. So T' has the same number of leaves as T . By the inductive hypothesis, after step 4, there is an out-branching T_s such that there is a distinct leaf in T_s . Then there is a distinct leaf in T_s corresponding to a leaf of T . This proves Claim 3.

Since T is a k -out-tree after step 3, T is a k -out-branching after step 4 by Claim 3. Then step 5 returns a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching, i.e., a $(T_0; \pi_0(l_1), \dots, \pi_0(l_p); k)$ -extended out-branching.

Summarizing the above discussion, we conclude that the algorithm `SpanningTree` either constructs a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching if such one exists, and correctly reports 'No' otherwise.

Now we analyze the complexity of the algorithm.

In step 1, S are assigned all vertices but determined leaves of T . This takes time $O(n)$. After step 1, all vertices of S are in T . Any vertex added to S at step 2.1 is also in T . So at step 2, all vertices of S are in T . Since only vertices not in T could be added to T at step 2.1, a vertex x cannot be added to S again once it is removed from S , i.e., any vertex can only be added to and removed from S once. Since steps 2.1 and 2.2 remove a vertex from S , step 2.1 and 2.2 can be executed at most n times and step 2.1 takes time $O(n)$ to find all its neighbors. Thus step 2 takes time $O(n^2)$. Step 3 takes time $O(n)$ to check every vertex in G . Step 4 takes time $O(p \cdot n) = O(n^2)$: check p out-chains once, and each out-chain takes time $O(n)$. In summary, the total time of the algorithm `SpanningTree` is bounded by $O(n^2)$. \square

4 The main algorithm and complexity analysis

Before we solve the MAX-LEAF OUT-BRANCHING problem, we give an algorithm to solve the problem `EXTENDING` $(T; \pi(l_1), \dots, \pi(l_p); k)$, which is used as a subroutine for the algorithm solving the MAX-LEAF OUT-BRANCHING problem. Consider the algorithm `ExtendTree` given in Figure 2.

Theorem 4.1 *For a given out-tree T in a digraph G of n vertices, with a set $\{\pi(l_1), \dots, \pi(l_p)\}$ of out-chains for T , the algorithm `ExtendTree` runs in time $O(n^3 4^k)$ and either constructs a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching if such one exists, or reports 'No' otherwise.*

PROOF. We first verify the correctness of the algorithm.

If the input T is already an out-branching with less than k leaves, then there is no $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. So step 1 reports 'No' correctly. If T is a k -out-tree, then by Theorem 3.11, the algorithm `SpanningTree` either constructs a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching if such one exists, or reports 'No' otherwise. Therefore, step 2 is correct. In step 3, x is a internal vertex of T . By Lemma 3.8, `EXTENDING` $(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to `EXTENDING` $(T(y_1, \dots, y_q); \pi_{y_1, \dots, y_q}(l_1), \dots, \pi_{y_1, \dots, y_q}(l_p); k)$. Thus, step 3 is correct.

After step 3, only leaves of T may have out-neighbors in $G - T$ and T has fewer than k leaves. Suppose that $(T; \pi(l_1), \dots, \pi(l_p); k)$ is extendable, and T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. Then any determined leaf l_i of T can only reach vertices in $\pi(l_i)$ in T_s . By the definition of out-chains, l_i can only reach vertices in $\pi(l_i)$ one by one in T_s . So l_i can only reach one leaf in T_s . If all leaves of T are determined leaves, then T_s has fewer than k leaves since T has fewer than k leaves. This contradicts that T_s is a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. Therefore, step 4 returns 'No' correctly.

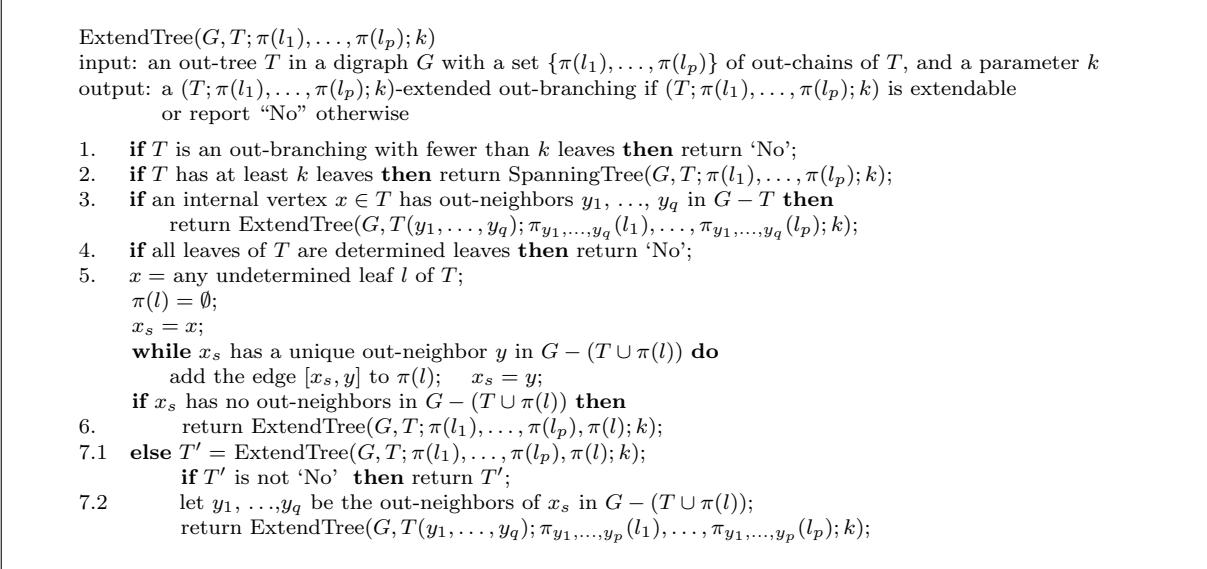


Figure 2: Algorithm 2. ExtendTree

After step 4, T has fewer than k leaves and some leaves of T are not determined leaves. Step 5 constructs an out-chain $\pi(l)$ for an undetermined leaf l such that the end vertex x_s of $\pi(l)$ either has at least two out-neighbors or has no out-neighbor in $G - (T \cup \pi(l))$. Note that $\pi(l)$ can be $\{l\}$.

If the end vertex x_s has no out-neighbor in $G - (T \cup \pi(l))$, then all vertices reachable from l are in $\pi(l)$ for all possible $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. This means that $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p), \pi(l); k)$. So step 6 is correct.

Now suppose x_s has more than one out-neighbors in $G - (T \cup \pi(l))$. If there is a $(T; \pi(l_1), \dots, \pi(l_p), \pi(l); k)$ -extended out-branching T_s , then T_s is also a $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching. So step 7.1 is correct. If step 7.1 can not find a $(T; \pi(l_1), \dots, \pi(l_p), \pi(l); k)$ -extended out-branching, then for all possible $(T; \pi(l_1), \dots, \pi(l_p); k)$ -extended out-branching T_s , the vertices reachable in T_s from l must contain some vertex x_s not in $\pi(l)$. This can only happen when x_s is reachable from l and x_s is not a leaf in T_s , i.e., x_s is an internal vertex of T_s . Then all vertices in $\pi(l)$ should be reachable from l in T_s . By Lemma 3.8, $(T; \pi(l_1), \dots, \pi(l_p); k)$ is reducible to $(T(y_1, \dots, y_q); \pi_{y_1, \dots, y_q}(l_1), \dots, \pi_{y_1, \dots, y_q}(l_p); k)$. Thus step 7.2 is correct.

Summarizing the above discussion, we conclude that the algorithm ExtendTree correctly solves the problem $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$.

Now we consider the complexity of the algorithm.

Let u the number of leaves of T , p be the number of determined leaves of T , then $a = k - u$ and $b = k - p$. Let $f(a, b) = a + b = 2k - u - p$ be the complexity of the algorithm ExtendTree. We use the search-tree method to analyze the time complexity.

Step 1 takes time $O(n)$. Step 2 takes time $O(n^2)$ by Theorem 3.11. Step 3 takes time $O(n^2)$: finding out-neighbors takes time $O(n)$, finding truncated out-chains takes time $O(p \cdot n) = O(n^2)$. Step 4 takes time $O(n)$. Step 5 can be implemented by a BFS search, thus, it takes time $O(n^2)$.

At step 6, the leaf l becomes a determined leaf from a undetermined leaf. So $u' = u$ and $p' = p + 1$. Then the complexity changes from $f(a, b) = 2k - u - p$ to $f'(a', b') = 2k - u - p - 1 = f(a, b) - 1$. At step 7.1, the complexity is the same as step 6: $f(a_1, b_1) = 2k - u - p - 1 = f(a, b) - 1$. Since x_s has at least two out-neighbors in step 7.2, $u' = u + 1$ and $p' = p$. So the complexity of step 7.2 is $f(a_2, b_2) = 2k - u - p - 1 = f(a, b) - 1$. Since steps 7.1 and 7.2 are two branches, we have the recurrence equation: $f(a, b) = 2f(a, b) - 1$.

Initially, $a + b = 2k - u - p \leq 2k$. When either $a = k - u = 0$ or $b = k - p = 0$, T is a k -out-tree. By Theorem 3.11, $\text{EXTENDING}(T; \pi(l_1), \dots, \pi(l_p); k)$ can be solved in time of $O(n^2)$, i.e., no branches are needed when $a = 0$ or $b = 0$. Then when $a + b = 0$, no branches are needed for ExtendTree, i.e., $f(0, 0) = 1$. By $f(a, b) = 2f(a, b) - 1$, we have $f(a, b) \leq 2^{a+b} = O(4^k)$, which is an upper bound of the total number of leaves in the search-tree for the algorithm ExtendTree. Since steps 1, 2, and 4 are executed only once, steps 3 and 5 are executed at most $O(n)$ times from the root to a leaf in the search-tree, the depth of the search-tree is $O(n)$, and the running time is $O(n^2)$ for each step, the total running time of SpanningTree is $O(n^3 4^k)$. \square

Now we are ready to present our algorithm for the MAX-LEAF OUT-BRANCHING problem.

Theorem 4.2 *The MAX-LEAF OUT-BRANCHING problem can be solved in time $O(n^4 4^k)$.*

PROOF. The algorithm for solving the MAX-LEAF OUT-BRANCHING problem is very simple: for each vertex $x \in G$, we call the algorithm $\text{ExtendTree}(G, T(y_1, \dots, y_q); \emptyset; k)$ where (1) T is the single vertex x ; (2) y_1, \dots, y_q are all out-neighbors of x in G ; and (3) \emptyset means that all leaves of $T(y_1, \dots, y_q)$ are not determined. If a $(T(y_1, \dots, y_q); \emptyset; k)$ -extended out-branching T_s is found for some x , then T_s is a k -out-branching of G ; otherwise reports 'No'. To prove that the algorithm is correct, we prove the following claim.

Claim. If there is a k -out-branching T_s of G whose root is r , then $(T(y_1, \dots, y_q); \emptyset; k)$ is extendable, where y_1, \dots, y_q are the out-neighbors of r in G .

First r must have some out-neighbor z in T_s , and z must be in $\{y_1, \dots, y_q\}$. Suppose that $z = y_1$. Then T_s is a $(T_{ry_1}; \emptyset; k)$ -extended out-branching since (1) T_s is a k -out-branching; (2) T_s and T_{ry_1} have the same root r ; (3) T_{ry_1} is a subgraph of T_s ; and (4) all vertices reachable in T_s from a determined leaf l_i of T_{ry_1} are in $\pi(l_i)$ since there is no determined leaf in T_{ry_1} . Therefore, $(T_{ry_1}; \emptyset; k)$ is extendable.

Now r is an internal vertex of T_{ry_1} , so $\text{EXTENDING}(T_{ry_1}; \emptyset; k)$ is reducible to $\text{EXTENDING}(T(y_1, \dots, y_q); \emptyset; k)$ by Lemma 3.8. Since $(T_{ry_1}; \emptyset; k)$ is extendable, $(T(y_1, \dots, y_q); \emptyset; k)$ is extendable. This proves the Claim.

Therefore, if there is a k -out-branching T_s rooted at r for G , the algorithm described above for the MAX-LEAF OUT-BRANCHING problem must call $\text{ExtendTree}(G, T(y_1, \dots, y_q); \emptyset; k)$, where y_1, \dots, y_q are the out-neighbors of the vertex r , since ExtendTree is executed on every vertex in G . By our claim, $(T(y_1, \dots, y_q); \emptyset; k)$ is extendable. By Theorem 4.1, the algorithm $\text{ExtendTree}(G, T(y_1, \dots, y_q); \emptyset; k)$ will construct a $(T(y_1, \dots, y_q); \emptyset; k)$ -extended out-branching, which is a k -out-branching by definition.

If no $(T(y_1, \dots, y_q); \emptyset; k)$ -extended out-branching is found for any $x \in G$, then there is no k -out-branching of G . So the algorithm for the MAX-LEAF problem is correct: it either constructs a k -out-branching if such one exists, or reports 'No' otherwise.

The running time of this algorithm is $O(n)$ times the time of ExtendTree , which is $O(n^4 4^k)$. \square

5 Conclusion

In this paper, we presented an $O^*(4^k)$ time algorithm for the MAX-LEAF OUT-BRANCHING problem on digraphs, which significantly improves the previous best algorithm of running time $O^*(2^{O(k \log k)})$ for the problem. The algorithm can be applied directly to solve the simpler MAX-LEAF SPANNING-TREE problem on undirected graphs, which also gives an improvement over the previous best algorithm of running time $O^*(6.75^k)$ for the problem. The improvements are based on a deeper study of the combinatorial structures of digraphs that reveals further relationship between out-trees and out-branchings in digraphs, and on applications of a new algorithmic technique for the design and analysis of parameterized algorithms. In particular, the new algorithmic technique identifies indirected branching measures that bound the number of computational branches in a branch-and-search process that do not reduce the parameter value effectively. This new technique eases the task of designing branching rules, reduces the analysis of complicated cases, and helps to design simpler but faster algorithms.

Studies of effective parameterized algorithms for NP-hard problems have attracted considerable attention recently. A number of new algorithmic techniques have been proposed and investigated that have turned out to be very effective in development of parameterized algorithms. Identifying indirected branching measures and effectively use them seem among the most promising ones. The idea has been used, implicitly or explicitly, in tackling a number of parameterized problems, including the well-known FEEDBACK VERTEX SET problems on digraphs and on undirected graphs [8, 9]. The technique seems relatively new, not well studied in the traditional algorithm design and analysis, and deserves further investigation.

References

- [1] N. ALON, F. FOMIN, G. GUTIN, M. KRIVELEVICH, AND S. SAURABH, Parameterized algorithms for directed maximum leaf problems, *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, *Lecture Notes in Computer Science 4596*, pp. 352-362, (2007).
- [2] N. ALON, F. FOMIN, G. GUTIN, M. KRIVELEVICH, AND S. SAURABH, Better algorithms and bounds for directed maximum leaf problems, *Proc. 27th International Conference on Foundations of Software*

- Technology and Theoretical Computer Science (FSTTCS 2007), Lecture Notes in Computer Science 4855*, pp. 316-327, (2007).
- [3] H. BODLADENDER, On linear time minor tests and depth-first search, *Proc. 1st Workshop on Algorithms and Data Structures (WADS 1989), Lecture Notes in Computer Science 382*, pp. 577-590, (1989).
- [4] P. BONSMMA, T. BRUEGGEMANN, AND G. WOEGINGER, A faster FPT algorithm for finding spanning trees with many leaves, *Proc. 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003), Lecture Notes in Computer Science 2747*, pp. 259-268, (2003).
- [5] P. BONSMMA AND F. DORN, An FPT algorithm for directed spanning k -Leaf, *CoRR abs/0711.4052*, (2007).
- [6] P. BONSMMA AND F. DORN, Tight bounds and faster algorithms for directed max-leaf problems, *Proc. 16th Annual European Symposium on Algorithms, (ESA 2008)*, to appear (2008).
- [7] P. BONSMMA AND F. ZICKFELD, Spanning trees with many leaves in graphs without diamonds and blossoms, *Proc. 8th Latin American Theoretical Informatics (LATIN 2008), Lecture Notes in Computer Science 4957*, pp. 531-543, (2008).
- [8] J. CHEN, F. FOMIN, Y. LIU, S. LU, AND Y. VILLANGER, Improved algorithms for the feedback vertex set problems, *Proc. 10th Workshop on Algorithms and Data Structures (WADS 2007), Lecture Notes in Computer science 4619*, pp. 422-433, (2007).
- [9] J. CHEN, Y. LIU, S. LU, B. O'SULLIVAN, AND I. RAZGON, A fixed-parameter algorithm for the directed feedback vertex set problem, *Proc. 40th ACM Symposium on Theory of Computing (STOC 2008)*, pp. 177-186, (2008).
- [10] M. DRESCHER AND A. VETTA, An approximation algorithm for the maximum leaf spanning arborescence problem, *Manuscript*, (2007).
- [11] R. DOWNEY AND M. FELLOWS, Fixed parameter tractability and completeness. in *Complexity Theory: Current Research*, K. AMBOS-SPIES, S. HOMER, AND U. SCHONING (eds.), Cambridge University Press, pp. 191-225, (1993).
- [12] R. DOWNEY AND M. FELLOWS, Parameterized computational feasibility, in *Feasible Mathematics II*, P. CLOTE, J. REMMEL (eds.), Birkhauser, pp. 219-244, (1995).
- [13] M. FELLOWS, C. MCCARTIN, F. ROSAMOND, AND U. STEGE, Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems, *Proc. 20th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2000), Lecture Notes in Computer Science 1974*, pp. 240-251, (2000).
- [14] G. GALBIATI, F. MAFFIOLI, AND A. MORZENTI, A short note on the approximability of the maximum leaves spanning tree problem, *Information Processing Letters 52*, pp. 45-49, (1994).
- [15] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.
- [16] S. GUHA AND S. KHULLER, Approximation algorithms for connected dominating sets, *Algorithmica 20*, pp. 374-387, (1998).
- [17] G. GUTIN AND A. YEO, Some parameterized problems on digraphs, *The Computer Journal*, (2008).
- [18] H.-I. LU AND R. RAVI, Approximation maximum leaf spanning trees in almost linear time, *Journal of Algorithms 29*, pp. 132-141, (1998).
- [19] R. SOLIS-OBA, 2-approximation algorithm for finding a spanning tree with maximum number of leaves, *Proc. 6th Annual European Symposium on Algorithms, (ESA 1998), Lecture Notes in Computer Science 1461*, pp. 441-452, (1998).
- [20] M. THAI, F. WANG, D. LIU, S. ZHU, AND D. DU, Connected dominating sets in wireless networks with different transmission range, *IEEE Transaction on Mobile Computing 6*, pp. 721-730, (2007).
- [21] B. WU AND K. CHAO, *Spanning Trees and Optimization Problems*, CRC Press, 2003.