RWTH Aachen

Department of Computer Science
Technical Report

A New Algorithm for Finding Trees with Many Leaves

Joachim Kneis, Alexander Langer, and Peter Rossmanith

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2008-15

RWTH Aachen · Department of Computer Science · Juni 2001

A New Algorithm for Finding Trees with Many Leaves*

Joachim Kneis, Alexander Langer, Peter Rossmanith

Dept. of Computer Science, RWTH Aachen University, Germany
Email: {kneis, langer, rossmani}@cs.rwth-aachen.de

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. Currently, the fastest algorithms for these problems have run times of $O(poly(n) + 6.75^k poly(k))$ and $2^{O(k \log k)} poly(n)$, respectively.

1 Introduction

In this paper we consider the graph theoretical problem 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 common. The 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 case is defined as follows:

Maximum Leaf Spanning Tree (MLST)

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

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

It is long known that MLST $\in$ FPT because a graph G contains a k-leaf spanning tree iff G has $K_{1,k}$ (a k-star) as a minor [13]. However, the proof 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 $K_{1,k}$ 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. 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 $f(k)$ 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 $|G'| \le f(k)$ and $k' \le g(k)$. 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 instance to a graph 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 two 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: An directed graph $G = (V, E)$, a positive integer k
Parameter: k
Question: Does G have a spanning 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]).

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 an k-leaf out-tree

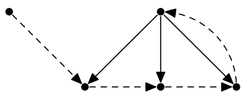

Fig. 1. A graph containing a 3-leaf out-tree, but no 3-leaf spanning tree.

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 [1] to $2^{O(k \log^2 k)} poly(|V|)$ and, if G is acyclic, to $2^{O(k \log k)} poly(|V|)$.

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.

The current state of affairs can be summarized as follows: While algorithms for MLST can already be considered efficient for sufficiently small values of k, today's algorithm for directed graphs are still far from being practical.

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). Therefore previous algorithms for DMLOT cannot solve DMLST even with small modifications. In this paper, we show 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 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 $r = \mathrm{root}(T)$, 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 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 an *reverse edge*. Such a representation is commonly used by algorithmic graph libraries like LEDA. Edges are therefore denoted by (u, v). Without loss of generality ($k > 2$ or $n \neq 2$), trees in undirected graphs are assumed to be rooted, and we use terms *tree* and *spanning tree* for *out-tree* and *spanning out-tree.*

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 $\mathrm{root}(T)$, $\mathrm{leaves}(T)$ and $\mathrm{inner}(T) := V(T) \setminus \mathrm{leaves}(T)$, 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 $N(\mathrm{inner}(T)) \subseteq V(T)$, 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* $\mathrm{leaves}(T)$. *(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 $\mathrm{root}(T') = \mathrm{root}(T)$ 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 $R \subseteq \mathrm{leaves}(T')$ (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 $x_1 := \mathrm{root}(T) = \mathrm{root}(T')$ and consider the path $x_1, \ldots, x_l$ with $x = x_l$ 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 $x_{i+1} \notin V(T)$. Since $T \prec T'$, x_i is a leaf in T and hence $x_i \in \mathrm{leaves}(T) = R \cup B$. On the other hand, $x_i \in \mathrm{inner}(T')$, and with $R \subseteq \mathrm{leaves}(T')$, 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 expanded 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 expanded to a spanning tree using an efficient postprocessing operation.

Lemma 2 is, however, not applicable for directed graphs, as seen in 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 r, then any k-leaf out-tree rooted in r can be expanded to a k-leaf spanning out-tree of G in time $O(n + m)$.*

Proof. Let T be an out-tree that has at least k leaves, let $x_1 := r$ be its root, 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 tree rooted in $r = x_1$, such a path must exist in G. Furthermore, $x \notin V(T)$ and hence there is $1 \le i \le s$ such that $x_i \in V(T)$ and $x_j \notin U$ 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.

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 they are preserved 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 $|N_{\overline{T}}(u)| \le 1$, we can skip the latter of the two branches by Lemma 5 and

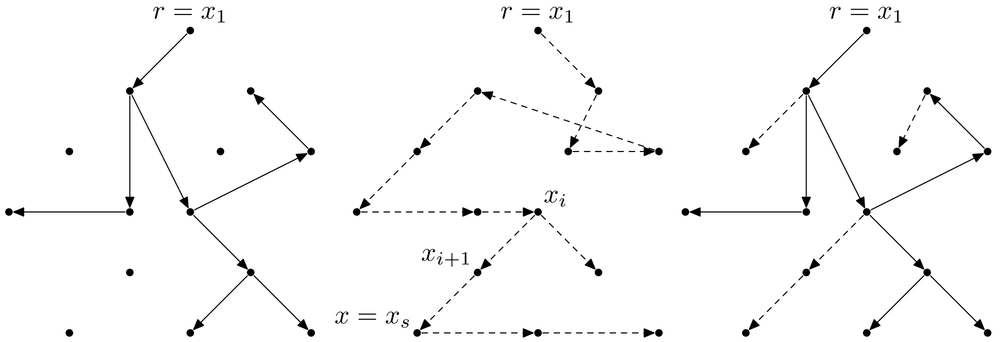

Fig. 2. How to extend a k-leaf out-tree into a k-leaf spanning out-tree: For the ease of illustration, we do not show all the edges in G. A 4-leaf out-tree is depicted in the first figure. The second figure shows an arbitrary spanning out-tree, we chose one with two leaves. We can enrich the first out-tree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

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 $x \in \operatorname{inner}(T')$.*
2. *If there is a k-leaf tree T', such that $T' \succeq (T, R, B)$ and $x \in \operatorname{inner}(T')$, then there is also a k-leaf tree $T'' \succeq (T + \{\, (x, u) \mid u \in N_{\overline{T}}(x) \,\}, R, N_{\overline{T}}(x) \cup B \setminus \{x\})$.*

Proof. 1. Let T' be a k-leaf tree, such that $T' \succeq (T, R, B)$, i.e., T' is a leaf-preserving extension of T. Then either $x \in \operatorname{inner}(T')$ or $x \in \operatorname{leaves}(T')$. If $x \in \operatorname{leaves}(T')$, then T' is also a leaf-preserving extension of $(T, R \cup \{x\}, B \setminus \{x\})$.

2. Let T' be a k-leaf tree, such that $T' \succeq (T, R, B)$ and $x \in \operatorname{inner}(T')$. First note that $N_{\overline{T}}(x) \neq \emptyset$, because $x \in \operatorname{inner}(T')$ 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 $x_1 := \operatorname{root}(T')$ 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 $y \in \operatorname{leaves}(T')$ implies $y \in \operatorname{leaves}(T'')$. Furthermore, the connectivity of T' remains intact. See Figure 3 for an example. Doing so iteratively for all neighbors y of x yields a k-leaf tree T'' with $\{\, (x, u) \mid u \in N_{\overline{T}}(x) \,\} \subseteq E(T'')$. Therefore we obtain $T'' \succeq (T + \{\, (x, u) \mid u \in N_{\overline{T}}(x) \,\}, R, N_{\overline{T}}(x) \cup B \setminus \{x\})$.

The algorithm furthermore does only fix some node x as an inner node, if this will result in a tree that has at least *two new* leaves. Hence, paths are followed until at least two new nodes have been found.

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. Assume T' is 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 $x \in \operatorname{leaves}(T'')$, i.e., $T'' \succ (T, R \cup \{x\}, B \setminus \{x\})$.

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

if $|R| + |B| \geq k$ **then return** "yes"
if $B = \emptyset$ **then return** "no"
Choose $u \in B$.
// Try branch where u is a leaf
if $\textsc{MaxLeaf}(G, T, R \cup \{u\}, B \setminus \{u\}, k)$ **then return** "yes"
// If u is not a leaf, it must be inner node in all extending solutions
$B := B \setminus \{u\}$
$N := N_{\overline{T}}(u)$
$T := T \cup \{ (u, u') \mid u' \in N \}$
// follow paths, see Lemma 5
while $|N| = 1$ **do**
  $u := v \in N$
  $N := N_{\overline{T}}(u)$
  $T := T \cup \{ (u, u') \mid u' \in N \}$
done
// Do not branch if no neighbors left, see Corollary 1
if $N = \emptyset$ **then return** "no".
return $\textsc{MaxLeaf}(G, T, R, B \cup N, k)$

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 $x \in \text{leaves}(T)$. 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 $(z, x) \in E(T)$, and $E(T) \subseteq E(T')$. In particular, $(x, y) \notin E(T')$, since T' is a tree. Hence $x \in \text{leaves}(T')$ and $T' \succ (T, R \cup \{x\}, B \setminus \{x\})$.

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

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 $(T^{(l+1)}, R^{(l+1)}, B^{(l+1)})$.

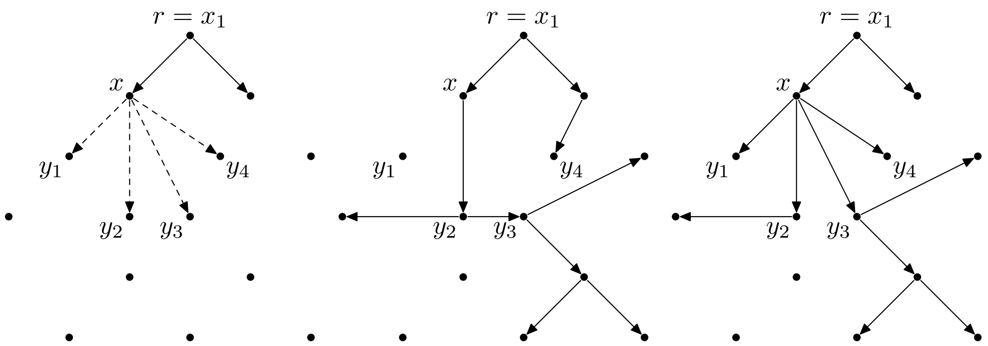

Fig. 3. The exchange argument (Lemma 4): The first figure shows a leaf-labeled tree (T, R, B) with $x \in B$. The neighborhood of x, $N_{\overline{T}}(x)$, is shown with dashed edges. The second figure shows a 5-leaf tree $T' \succ (T, R, B)$, but different choices for edges originating in x have been 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.

Whenever $\textsc{MaxLeaf}(G, T, R, B, k)$ 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 $\textsc{MaxLeaf}$ is called with (T'', R'', B''). Since G is finite, eventually $\textsc{MaxLeaf}$ 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 $k > 2$, $r \in \text{inner}(T)$. 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 $\textsc{MaxLeaf}$, 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 $x \in B$. By Lemma 4,

1. there is a k-leaf tree $T'' \succeq (T, R \cup \{x\}, B \setminus \{x\})$, or
2. there is a k-leaf tree $T'' \succeq (T + \{\, (x, u) \mid u \in N_{\overline{T}}(x) \,\}, R, N_{\overline{T}}(x) \cup B \setminus \{x\})$.

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 $\textsc{MaxLeaf}(G, T, R \cup \{u\}, B \setminus \{u\}, k)$ 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 $T''' \succeq (R, B \setminus \{x\} \cup N_{\overline{R,B,I}}(x), I \cup \{x\})$. 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 $v_0 = x$, $(T_0, R_0, B_0) = (T, R, B)$, and

1. $(T_{i+1}, R_{i+1}, B_{i+1}) = (T_i + (v_i, v_{i+1}), R_i, B_i \cup N_{\overline{T_i}}(v_i))$,
2. $N_{\overline{T_i}}(v_i) = \{v_{i+1}\}$ for $0 \le i < l$,
3. $|N_{\overline{T_l}}(v_l)| \ne 1$, and
4. for each $0 \le i \le l$ there is a k-leaf tree $T'_i \succeq (T_i, R_i, B_i, I_i)$.

By Corollary 1, we have $N_{\overline{T_l}}(v_l) \ne \emptyset$, i.e., the algorithm does not return "no". Hence the algorithm recursively calls itself as $\textsc{MaxLeaf}(G, T_l, R_l, B_l, k)$, 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 $\textsc{MaxLeaf}(G, T_v, \emptyset, N(v), k)$ 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 $\textsc{MaxLeaf}$ is called with a leaf-labeled tree (T, R, B), the algorithm recursively calls itself at most two times: In the first branch some vertex $u \in B$ is fixed as a leaf and the algorithm calls itself as $\textsc{MaxLeaf}(G, T, R \cup \{u\}, B \setminus \{u\}, k)$. 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 $|N| \ge 2$. Here, the recursive call is $\textsc{MaxLeaf}(G, T', R, B \setminus \{u\} \cup N, k)$ for some tree T', and hence the potential decreases by $\Phi(k, R, B) - \Phi(k, R, B \setminus \{u\} \cup N) \ge 1$.

Note that $\Phi(k, R, B) \le 0$ implies $|R + B| \ge k$. Since the potential decreases by at least 1 in each recursive call, the height of the search tree is therefore at most $\Phi(k, R, B) \le 2k$. 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 $G = (V, E)$. 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, $n = |V| = O(k)$.

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, so we need to iterate over possible roots. Since $k > 2$, it is easy to see that either some $v \in V$ or one of its neighbors is root of some k-leaf spanning tree, if any k-leaf spanning tree T exists at all: If $v \in \text{leaves}(T)$, the unique predecessor u of v in T is an inner node $u \in \text{inner}(T)$. By choosing a node of minimum degree, we obtain the best run time bounds.

Let $v \in V$ be a node of minimum degree. We need to call $\textsc{MaxLeaf}$ with parameters $(G, T_u, R, N(u), k)$ for all $u \in N[v]$. By Lemma 6, these calls suffice to solve MLST: If G contains a k-leaf tree, at least one of those u is a root of some k-leaf tree, and hence the respective call to $\textsc{MaxLeaf}$ returns "yes". Otherwise each call returns "no".

By Lemma 7, the total number of recursive calls is bounded by

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

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 doubled 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 $O(\mathit{poly}(n) + 4^k \cdot k^2)$. By Lemma 2, each k-leaf tree can be extended to a spanning tree with at least k leaves, so 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 expand 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, $\textsc{MaxLeaf}(G, T_r, \emptyset, N(r), k)$ returns "yes" by Lemma 6. Otherwise, $\textsc{MaxLeaf}(G, T_v, \emptyset, N(v), k)$ 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 $\textsc{MaxLeaf}$. 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 $c^k \mathit{poly}(n)$-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 *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 4855 in Lecture Notes in Computer Science, pages 316–327, New Delhi, India, Nov. 2007. Springer-Verlag.
2. N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized algorithms for directed maximum leaf problems. In *Proceedings of the 34th International Colloquium on Automata, Languages, and Programming (ICALP)*, number 4596 in Lecture Notes in Computer Science, pages 352–362. Springer-Verlag, 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 *Proceedings of the 28th Conference on Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 259–268. Springer-Verlag, 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 *Proceedings of the 16th European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science. Springer-Verlag, 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 *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, Lecture Notes in Computer Science. Springer-Verlag, 2008. To appear.
9. P. S. Bonsma and F. Zickfeld. Spanning trees with many leaves in graphs without diamonds and blossoms. In *Proceedings of the 8th Symposium on Latin American Theoretical Informatics*, number 4957 in Lecture Notes in Computer Science, pages 531–543, Búzios, Brazil, 2008. Springer-Verlag.
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 *Proceedings of the 1st ACiD Workshop*, 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 *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 240–251, London, UK, 2000. Springer-Verlag.

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 *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 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 *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 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 *Proceedings of the 6th European Symposium on Algorithms (ESA)*, number 1461 in Lecture Notes in Computer Science, pages 441–452, London, UK, 1998. Springer-Verlag.
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.

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from http://aib.informatik.rwth-aachen.de/. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2005-04 Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem

2005-05 Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots

2005-06 Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information

2005-07 Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks

2005-08 Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut

2005-09 Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures

2005-10 Benedikt Bollig: Automata and Logics for Message Sequence Charts

2005-11 Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12 Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems

2005-13 Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments

2005-14 Felix C. Freiling, Sukumar Ghosh: Code Stabilization

2005-15 Uwe Naumann: The Complexity of Derivative Computation

2005-16 Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)

2005-17 Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)

2005-18 Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"

2005-19 Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-02 Michael Weber: Parallel Algorithms for Verification of Large Systems

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07	Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications
2007-08	Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches
2007-09	Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption
2007-10	Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes
2007-11	Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke
2007-12	Uwe Naumann: An L-Attributed Grammar for Adjoint Code
2007-13	Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs
2007-14	Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes
2007-15	Volker Stolz: Temporal assertions for sequential and concurrent programs
2007-16	Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks
2007-17	René Thiemann: The DP Framework for Proving Termination of Term Rewriting
2007-18	Uwe Naumann: Call Tree Reversal is NP-Complete
2007-19	Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control
2007-20	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems
2007-21	Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains
2007-22	Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets
2008-01 *	Fachgruppe Informatik: Jahresbericht 2007
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René :Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs

2008-10 George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time

2008-11 George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows

2008-12 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages

2008-13 Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutierrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs

2008-14 Bastian Schlich: Model Checking Software for Microcontrollers

* These reports are only available as a printed version.

Please contact biblio@informatik.rwth-aachen.de to obtain copies.