Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 105 (2008) 164–169
Information Processing Letters
www.elsevier.com/locate/ipl

# On finding spanning trees with few leaves

Gábor Salamon [1], Gábor Wiener [*,2]

*Department of Computer Science and Information Theory, Budapest University of Technology and Economics, H-1521, Hungary*

Received 9 August 2006; accepted 6 August 2007

Available online 14 September 2007

Communicated by K. Iwama

**Abstract**

The problem of finding a spanning tree with few leaves is motivated by the design of communication networks, where the cost of the devices depends on their routing functionality (ending, forwarding, or routing a connection). Besides this application, the problem has its own theoretical importance as a generalization of the Hamiltonian path problem. Lu and Ravi showed that there is no constant factor approximation for minimizing the number of leaves of a spanning tree, unless $P = NP$. Thus instead of minimizing the number of leaves, we are going to deal with maximizing the number of non-leaves: we give a linear-time 2-approximation for arbitrary graphs, a $\frac{3}{2}$-approximation for claw-free graphs, and a $\frac{6}{5}$-approximation for cubic graphs.


*Keywords:* Approximation algorithms; Graph algorithms; Spanning trees

## 1. Introduction and notations

Spanning tree optimization problems naturally arise in many applications, such as network design and connection routing. Several of these problems have an objective function based on the degrees of nodes of the spanning tree (see for example [1,4,6]). This model is extremely useful when designing networks where the cost of devices to install depends fundamentally on the required routing functionality (ending, forwarding, or routing a connection).

In this paper we deal with a problem of this kind. The MINLST (Minimum Leaf Spanning Tree) problem consists of finding a spanning tree of a given graph having a minimum number of leaves. Since Hamiltonian paths (if exist) are the only spanning trees with exactly 2 leaves, MINLST is a generalization of the Hamiltonian path problem and is therefore *NP*-hard. Moreover, it is even hard to approximate: using a result of Karger et al. [2] concerning the problem of finding the longest simple path of a graph, Lu and Ravi [3] showed that no constant-factor approximation exists for the MINLST problem, unless $P = NP$.

From an optimization point of view, MINLST is equivalent to the problem of finding a spanning tree with a maximum number of internal nodes (non-leaves). However, we show that this latter problem (called MAXIST—Maximum Internal Spanning Tree) has

* Corresponding author.
*E-mail addresses:* gsala@cs.bme.hu (G. Salamon), wiener@cs.bme.hu (G. Wiener).

[1] Research is supported by Grant Nos. 042559, and 044733 of the Hungarian National Science Foundation (OTKA), and by the Grant No. 2003-5044438 of the European MCRTN Adonet Contract, while visiting the Graph Theory and Combinatorial Optimization group in Grenoble.

[2] Research is supported by Grant No. 042559 of the Hungarian National Science Foundation (OTKA).

0020-0190/$ – see front matter 
doi:10.1016/j.ipl.2007.08.030

much better approximability properties. In Section 2 we give a linear time 2-approximation algorithm for the MAXIST problem based on depth first search. In Section 3 we show that a refined version of the depth first search algorithm provides a $\frac{3}{2}$-approximation on claw-free graphs (graphs not containing $K_{1,3}$ as an induced subgraph) and a $\frac{6}{5}$-approximation on cubic graphs (3-regular graphs). It is worth mentioning that Lu and Ravi [3,4] gave the first constant factor approximation for the problem of finding a spanning tree with a maximum number of leaves. Currently, the best known approximation has a factor 2 and is due to Solis-Oba [6].

By a graph $G = (V, E)$ we mean an undirected simple connected graph of $n = |V|$ nodes and $m = |E|$ edges. The degree of a node $v$ in a graph $G$ is denoted by $d_G(v)$. For any set of nodes $X$, $d_G(X) = \sum_{v \in X} d_G(v)$. Let $X$ and $Y$ be disjoint vertex sets of $G$, then $e_G(X, Y)$ denotes the number of edges between $X$ and $Y$ in $G$. $V_i(G)$ ($V_{\geqslant i}(G)$) denotes the set of nodes having degree exactly $i$ (at least $i$) in a graph $G$. For a graph $G = (V, E)$ and $X \subseteq V$ the graph $G[X] = (X, E')$ is the induced subgraph of $G$ on $X$ and $e_G(X) = |E'|$. $\mathrm{comp}_G(X)$ denotes the number of connected components of $G[X]$.

For a vertex set $V'$ and a vertex $v$, $V' - v$ stands for $V' \setminus \{v\}$ and for a subgraph $H$ the subgraph $G \setminus H$ has node set $V(G)$ and edge set $E(G) \setminus E(H)$. Finally, given two nodes $u$ and $v$ of a tree $T$ we denote by $P_T(u, v)$ the unique path in $T$ connecting $u$ and $v$.

## 2. Maximizing the number of internal nodes

In this section, we first give a linear-time algorithm (ILST) that finds either a Hamiltonian path of a given graph $G$ or a spanning tree of $G$ with independent leaves. Then we prove that such a tree has at least half the optimal number of internal nodes.

Our algorithm is basically a depth-first search. However, it can happen that the leaves of a DFS-tree $T$ are not independent. Thus, a single additional local replacement step might be required.

For a detailed discussion, let us recall that depth first search (DFS) (see for example [7, p. 538]) is a traversal, that is, it visits the nodes of the graph one by one thus producing a spanning tree (the so-called DFS-tree) $T$ of $G$ *rooted* at some node $r$. We assign a unique *DFS number* to each node $v$ which is the rank of $v$ in the order of visiting. Each non-root node $v$ has a unique *parent* $u$, namely the node succeeding $v$ on the path $P_T(v, r)$. The node $v$ is called a *child* of $u$, and the nodes of the path $P_T(u, r)$ are the *ancestors* of $v$. A node having no child is called a *d-leaf*. Note that all d-leaves of $T$ are also leaves of $T$, and only the root $r$ can be a leaf of $T$ without being a d-leaf. We recall a well-known property of DFS-trees.

**Claim 1.** *Let $T$ be a DFS-tree of the undirected graph $G$. Then each edge of $G$ connects two nodes of which one is an ancestor of the other in $T$. This implies that the d-leaves of $T$ form an independent set of $G$.*

Though the d-leaves of a DFS-tree $T$ are independent in $G$, it may happen that the root of $T$ is a leaf and is adjacent to some d-leaves of $T$. In this case, an additional replacement step is executed that decreases the number of leaves by one and also makes the leaves independent.

Algorithm ILST produces a spanning tree, as the replacement step first creates a unique cycle by adding an edge to the DFS-tree and then removes an edge of this cycle. If the replacement step is applied then $l$ and $r$ become internal nodes and $y$ becomes a leaf. Since $y$ is not an ancestor of the other leaves, the obtained spanning tree has independent leaves. The DFS-tree can be found in linear time. If we check $(r, l) \in E(G)$ for each d-leaf $l$ during the traversal then the evaluation of the "if" condition needs only constant extra time. Once $l$ is found, finding $x$ and $y$ and executing the replacement step needs linear time. Thus we have proved

**Input**: An undirected graph $G = (V, E)$
**Output**: A spanning tree $T$ of $G$ with independent leaves
$T \leftarrow$ `DFS(G)` ; `// an arbitrary DFS tree of` $G$
$r \leftarrow$ the root of $T$;
**if** *$T$ is not a Hamiltonian path and $d_T(r) = 1$ and $l$ is a d-leaf such that $(r, l) \in E(G)$* **then**
  `// r is a leaf and is adjacent to an other leaf` $l$
  $x \leftarrow$ the branching node being closest to $l$ in $T$;
  $y \leftarrow$ the neighbor of $x$ on the path $(l, x)$;
  Add edge $(l, r)$ to $T$;
  Delete edge $(x, y)$ from $T$;
**return** $T$;

Algorithm 1. Independent Leaves Spanning Tree (ILST).

**Claim 2.** *Algorithm ILST gives either a Hamiltonian path or a spanning tree whose leaves form an independent set of $G$ in $\mathcal{O}(m)$ time.*

In order to show that ILST is a 2-approximation, first we define the *cut-asymmetry* of a graph $G = (V, E)$ to be $\mathrm{ca}(G) = \max_{X \subsetneq V, X \neq \emptyset}(\mathrm{comp}_G(X) - \mathrm{comp}_G(V \setminus X))$. Lemma 3 shows a connection between cut-asymmetry and the number of leaves of a tree.

**Lemma 3.** *Let $T$ be an arbitrary tree on at least 3 vertices. Then $\mathrm{ca}(T) = |V_1(T)| - 1$.*

**Proof.** First observe that $\mathrm{comp}_T(V_1(T)) - \mathrm{comp}_T(V \setminus V_1(T)) = |V_1(T)| - 1$ since $V_1(T)$ is an independent set and $V \setminus V_1(T)$ spans a subtree. This implies $\mathrm{ca}(T) \geqslant |V_1(T)| - 1$.

To show that $\mathrm{ca}(T) \leqslant |V_1(T)| - 1$ let $X \subset V$ be a set of vertices for which $\mathrm{ca}(T) = \mathrm{comp}_T(X) - \mathrm{comp}_T(V \setminus X)$. For the sake of convenience, let $x = \mathrm{comp}_T(X)$ and $\bar{x} = \mathrm{comp}_T(V \setminus X)$. Then $e_T(X) = |X| - x$ and $e_T(V \setminus X) = n - |X| - \bar{x}$, thus

$$\begin{aligned} d_T(X) &= 2e_T(X) + e_T(X, V \setminus X) \\ &= 2e_T(X) + n - 1 - \big(e_T(X) + e_T(V \setminus X)\big) \\ &= 2\big(|X| - x\big) + x + \bar{x} - 1 \\ &= 2|X| - x + \bar{x} - 1. \end{aligned} \tag{1}$$

Observe that the contribution of each internal node to $d_T(X)$ is at least 2, yielding

$$\big|V_1(T) \cap X\big| \geqslant 2|X| - d_T(X). \tag{2}$$

Therefore, by (1) and (2), we have

$$\begin{aligned} \big|V_1(T)\big| &\geqslant \big|V_1(T) \cap X\big| \geqslant 2|X| - d_T(X) \\ &\geqslant x - \bar{x} + 1 = \mathrm{ca}(T) + 1 \end{aligned}$$

for the number of leaves of $T$, completing the proof of the lemma. □

Now we apply the above lemma to prove the approximation ratio.

**Theorem 4.** *Algorithm ILST is a 2-approximation for the* MAXIST *problem.*

**Proof.** We have seen that the algorithm is polynomial (actually, linear), so we only have to prove the approximation factor. Let $T^*$ be an optimal spanning tree, namely one having a maximum number of internal nodes and let $T$ be a spanning tree given by the algorithm. If $T$ is a Hamiltonian path then we are done. Otherwise we apply Lemma 3:

$$\begin{aligned} \big|V_1(T^*)\big| &= \mathrm{ca}(T^*) + 1 \\ &\geqslant \mathrm{comp}_{T^*}\big(V_1(T)\big) - \mathrm{comp}_{T^*}\big(V \setminus V_1(T)\big) + 1 \\ &\geqslant \big|V_1(T)\big| - \big|V \setminus V_1(T)\big| + 1 \\ &= 2\big|V_1(T)\big| - n + 1, \end{aligned}$$

since $V_1(T)$ is an independent set of $G$ (and thus also of $T^*$) by Claim 2. Thus

$$\begin{aligned} \big|V_{\geqslant 2}(T^*)\big| &= n - \big|V_1(T^*)\big| \leqslant 2\big(n - \big|V_1(T)\big|\big) \\ &= 2\big|V_{\geqslant 2}(T)\big|, \end{aligned}$$

proving the theorem. □

Notice that in DFS—and thus in ILST as well—the way of selecting the next node to visit is not fully specified. The only criterion is to choose an unvisited neighbor of the currently visited node. In Section 3 we present a refined version of DFS (RDFS) which applies a node selection rule to (partially) resolve the non-deterministic behavior of the original algorithm. We can profit of this refinement by getting a better approximation ratio on claw-free and on cubic graphs. Unfortunately, the approximation factor of 2 cannot be improved by the same refinement technique in general graphs. Fig. 1 shows a graph on $3k + 1$ vertices for which *every* run of ILST produces $k + 2$ internal nodes, while an optimal spanning tree has $2k + 1$ internal nodes.

## 3. Claw-free and cubic graphs

In this section we deal with claw-free graphs (graphs not containing $K_{1,3}$ as an induced subgraph) and cubic


Fig. 1. (a) A graph on $3k + 1$ vertices for which *every* run of ILST produces $k + 2$ internal nodes; (b) an optimal solution with $2k + 1$ internal nodes.

graphs (3-regular graphs). First we present a refined version of the original DFS algorithm, called RDFS. Then we prove that RDFS approximates the MAXIST problem within a factor of $\frac{3}{2}$ for claw-free graphs and within a factor of $\frac{6}{5}$ for cubic graphs.

RDFS is a depth first search in which we specify how to choose the next node of the traversal in the cases when DFS itself would choose arbitrarily from several candidates. The main idea is to select the vertex that has the minimum number of non-visited neighbors. For this purpose, we use the array "actdeg" to maintain the number of non-visited neighbors of each node.

RDFS differs from DFS only at line $(*)$, where the latter one would choose a non-visited neighbor of $v$ arbitrarily, but RDFS uses its own selection rule. Recall that DFS runs in linear time. At line $(*)$ we make at most $\Delta$ steps to find the minimum (where $\Delta$ is the maximum degree of $G$), and this line is executed at most once for each edge of $T$. Thus the total running time is $\mathcal{O}(\Delta n)$.

A tree produced by RDFS is called an *RDFS-tree*. We use the following notation. Let $T$ be an RDFS-tree, and let $l$ be a d-leaf of $T$ such that $d_G(l) \geqslant 2$. $c(l)$ stands for the neighbor of $l$ having the greatest DFS number such that $(l, c(l)) \in E(G) \setminus E(T)$ ($c(l)$ exists, because $d_G(l) \geqslant 2$). Let $g(l)$ denote the neighbor of $c(l)$ along the path $P_T(c(l), l)$. We also use the shorthand notations $v_1, v_2, v_3$ for the numbers $|V_1(T)|$, $|V_2(T)|$, and $|V_3(T)|$, respectively.

Now we prove a useful lemma concerning the degree of the node $g(l)$ in $T$.

**Lemma 5.** *Let $T$ be an RDFS-tree and let $l$ be a d-leaf of $T$. Then $d_T(g(l)) = 2$.*

**Proof.** Let us denote by $Y_v$ the set of vertices having DFS number greater than or equal to the DFS number of a given vertex $v$. It is obvious that $l \in Y_{c(l)}$.

Consider now that step of RDFS when we choose $g(l)$ to be the next visited vertex. By the RDFS rule, $d_{G[(Y_{g(l)}]}(g(l)) \leqslant d_{G[Y_{g(l)}]}(l)$. By the definition of $c(l)$ and $g(l)$, $d_{G[(Y_{g(l)}]}(l) = 1$, thus $d_{G[(Y_{g(l)}]}(g(l)) = 1$ (since $d_{G[(Y_{g(l)}]}(g(l)) \geqslant 1$ is obvious). Therefore $g(l)$ has only one child (and one parent, namely $c(l)$), so $d_T(g(l)) = 2$, indeed. □

Now we prove the approximation ratio for claw-free graphs.

**Theorem 6.** *RDFS is a $\frac{3}{2}$-approximation for the* MAXIST *problem for claw-free graphs.*

```
Input: An undirected graph G = (V, E)
Output: An RDFS tree T of G
begin
    T ← (V, ∅);
    foreach v ∈ V(G) do
        dfs[v] ← 0 ;                                   // the DFS number of v
        actdeg[v] ← d_G(v) ;          // the number of non-visited neighbors of v
    k ← 0 ;                             // the number of already visited vertices
    r ← a minimum degree vertex of G;
    RDFSNode(r);
    return T;
end

// Traversing from a node v
function RDFSNode(v)
begin
    k ← k + 1;
    dfs[v] ← k;
    foreach neighbor w of v do actdeg[w] ← actdeg[w] − 1;
    while actdeg[v] > 0 do
        // We refine the original DFS by specifying how to choose
        // the next node to visit.  This is called the RDFS rule.
*       w ← a neighbor of v that has not been visited yet and that minimizes actdeg[.];
        Add edge (v, w) to T;
        RDFSNode (w);
end
```

Algorithm 2. Refined DFS (RDFS).

**Proof.** We have seen that the algorithm is polynomial, so we have to check the approximation ratio. Let $G$ be an arbitrary connected claw-free graph on $n$ vertices and let $T$ be an RDFS-tree of $G$. First notice that $d_T(v) \leqslant 3$ for any $v \in V(T) = V(G)$, otherwise the node $v$ and three of its children would induce a subgraph $K_{1,3}$ in $G$ because of Claim 1. Thus our aim is to show that $v_2 + v_3 \geqslant \frac{2}{3} i_{\text{opt}}$, where $i_{\text{opt}}$ is the number of internal nodes of an optimal spanning tree. Since $T$ is a tree and $d_T(v) \leqslant 3$ for any $v \in V(T)$, we have $v_1 = v_3 + 2$.

Now we would like to find many nodes of degree 2 in $T$ in order to show that the number of internal nodes is large. For this, we use Lemma 5. The problem is that the nodes $g(l)$ (having degree 2 in $T$) are not necessarily distinct. However, we show that for claw-free graphs this is not the case.

**Lemma 7.** *Let $T$ be an RDFS-tree of $G$ and let $l$ and $l'$ be d-leaves of $T$ such that $d_G(l) \geqslant 2$ and $d_G(l') \geqslant 2$. Then $g(l) \neq g(l')$.*

**Proof.** Suppose to the contrary that $g(l) = g(l')$. It is obvious that $c(l)$ and $c(l')$ are ancestors of $l$ and $l'$, respectively, thus from $d_T(g(l)) = 2$ (Lemma 5) $c(l) = c(l')$ follows. Now consider the induced subgraph $S = G[\{c(l), l, l', g(l)\}]$. The vertices $l, l'$, and $g(l)$ are all adjacent to $c(l)$ in $G$. On the other hand, $l$ and $l'$ are d-leaves of $T$, thus they cannot be adjacent in $G$. Moreover, $g(l)$ cannot be adjacent either to $l$ or to $l'$ in $G \setminus T$, because $g(l)$ clearly has greater DFS number than $c(l) = c(l')$. Since $(l, g(l))$ and $(l', g(l))$ are not edges of $T$ either (otherwise $g(l)$ could not be a common ancestor of $l$ and $l'$), the induced subgraph $S$ is isomorphic to $K_{1,3}$, a contradiction. □

So we have found as many nodes of degree 2 in $T$ as the number of those d-leaves that have degree at least 2 in $G$. Let us denote by $w$ the number of vertices having degree 1 in $G$. These vertices are clearly leaves of any spanning tree of $G$, so the optimal spanning tree has at most $\min(n - w, n - 2)$ internal nodes.

We consider two cases.

Case 1. $w = 0$. Now every d-leaf has degree at least 2 in $G$, thus $v_2 \geqslant v_1 - 1$. Since $v_1 = v_3 + 2$ and $n = v_1 + v_2 + v_3$, we have $v_2 + v_3 \geqslant \frac{2}{3}(n - 2) \geqslant \frac{2}{3} i_{\text{opt}}$.

Case 2. $w \geqslant 1$. It suffices to show that $v_2 + v_3 \geqslant \frac{2}{3}(v_1 + v_2 + v_3 - w)$. Since the degree of the root of the RDFS-tree $T$ is minimum in $G$, now the root has degree 1 in $G$. Thus we have $v_2 \geqslant v_1 - w$, so it suffices to show that $v_2 + v_3 \geqslant \frac{2}{3}(2v_2 + v_3)$, that is, $v_3 \geqslant v_2$, which is equivalent to $v_1 - 2 \geqslant v_2$. If this latter inequality holds then we are done. Otherwise $v_2 \geqslant v_1 - 1$ holds, from which $v_2 + v_3 \geqslant \frac{2}{3}(n - 2) \geqslant \frac{2}{3} i_{\text{opt}}$ follows just like in Case 1. □

Now we turn to cubic graphs.

**Theorem 8.** *RDFS is a $\frac{6}{5}$-approximation algorithm for the* MAXIST *problem for cubic graphs.*

**Proof.** We have seen that the running time is $\mathcal{O}(n\Delta)$, which is linear for cubic graphs. Now we check the approximation ratio. Let $G$ be an arbitrary connected cubic graph on $n$ vertices and let $T$ be an RDFS-tree of $G$. Obviously, $d_T(v) \leqslant 3$ for any $v \in V(T) = V(G)$, thus $v_1 = v_3 + 2$. We show that $v_2 \geqslant 4v_1 - 6$, then some elementary computation gives $|V_{\geqslant 2}(T)| = v_2 + v_3 \geqslant \frac{5}{6} n - \frac{4}{3} \geqslant \frac{5}{6}(n - 2)$, from which the theorem follows.

Let $l$ be a d-leaf of $T$. Now $l$ has 2 neighbors in $G \setminus T$, one of them is $c(l)$, call the other one $c'(l)$. It is obvious that $d_T(c(l)) = 2$ and also $d_T(c'(l)) = 2$ if $c'(l)$ is not the root. Furthermore, $d_T(g(l)) = 2$ by Lemma 5. Now let $h(l)$ be the only neighbor of $g(l)$ in $G \setminus T$. As a consequence of the RDFS rule and of Claim 1 we obtain that $h(l)$ is an ancestor of $g(l)$. Then either $h(l)$ is the root or $d_T(h(l)) = 2$. It is easy to check that if $l$ and $l'$ are two distinct d-leaves then the sets $\{c(l), c'(l), g(l), h(l)\}$ and $\{c(l'), c'(l'), g(l'), h(l')\}$ can only have the root as a common element.

This way we associate 4 nodes (namely $c(l)$, $c'(l)$, $g(l)$, and $h(l)$) to every d-leaf $l$. Among these nodes only the root can occur more than once and all the other nodes have degree 2. Since the root can occur at most twice (because of 3-regularity) and the number of d-leaves is at least $v_1 - 1$, we have found $4(v_1 - 1) - 2$ distinct nodes of degree 2, that is, $v_2 \geqslant 4v_1 - 6$. □

## 4. Conclusions

We have shown that though the problem of finding a spanning tree having a minimum number of leaves has no constant-factor approximation (unless $P = NP$) [3], the problem of finding a spanning tree with a maximum number of non-leaves (MAXIST) has a linear time 2-approximation based on the well-known depth first search algorithm. By refining the algorithm we have also given a $\frac{3}{2}$-approximation for claw-free graphs and a $\frac{6}{5}$-approximation for cubic graphs. It is an open question whether these approximation factors can be improved, but the authors conjecture that algorithms using local improvement steps can provide an approximation ratio better than 2 for general graphs.

## Note added in proof

Recently Salamon has proved [5] that there exists a 7/4-approximation algorithm for the MAXIST problem for graphs without degree one nodes.

## Acknowledgement

The authors would like to thank to an anonymous referee whose comments helped a lot to improve the presentation of this paper.

## References

[1] L. Gargano, M. Hammar, P. Hell, L. Stacho, U. Vaccaro, Spanning spiders and light-splitting switches, Discrete Math. 285 (2004) 83–95.

[2] D. Karger, R. Motwani, G.D.S. Ramkumar, On approximating the longest path in a graph, in: Proc. of WADS'93, 1993, pp. 421–432.

[3] H.-I. Lu, R. Ravi, The power of local optimization: Approximation algorithms for maximum-leaf spanning tree (DRAFT), Technical Report CS-96-05, Department of Computer Science, Brown University, Providence, Rhode Island, 1996.

[4] H.-I. Lu, R. Ravi, Approximation for maximum leaf spanning trees in almost linear time, J. Algorithms 29 (1) (1998) 132–141.

[5] G. Salamon, Approximation algorithms for the maximum internal spanning tree problem, in: MFCS 2007, in: Lecture Notes in Computer Science, vol. 4708, 2007, pp. 90–102.

[6] R. Solis-Oba, 2-approximation algorithm for finding a spanning tree with maximum number of leaves, in: Proc. of 6th ESA Symposium, in: Lecture Notes in Computer Science, vol. 1461, Springer, 1998, pp. 441–452.

[7] J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity, Elsevier, 1990.