Approximation Algorithms for the Maximum Leaf Spanning Tree Problem on Acyclic Digraphs

Nadine Schwartges, Joachim Spoerhase, and Alexander Wolff

Chair of Computer Science I, University of Würzburg
http://www1.informatik.uni-wuerzburg.de/en/staff

Abstract. We consider the problem Maximum Leaf Spanning Tree (MLST) on digraphs, which is defined as follows. Given a digraph G, find a directed spanning tree of G that maximizes the number of leaves. MLST is NP-hard. Existing approximation algorithms for MLST have ratios of $O(\sqrt{\mathrm{OPT}})$ and 92.

We focus on the special case of acyclic digraphs and propose two linear-time approximation algorithms; one with ratio 4 that uses a result of Daligault and Thomassé and one with ratio 2 based on a 3-approximation algorithm of Lu and Ravi for the undirected version of the problem. We complement these positive results by observing that MLST is MaxSNP-hard on acyclic digraphs. Hence, this special case does not admit a PTAS (unless $\mathcal{P} = \mathcal{NP}$).

1 Introduction

Network design deals with the problem of optimally connecting a given set of network nodes by links. Network design problems arise for example in the planning of telecommunications networks, logistical networks or in circuit layout. Often, network design problems are modeled as graph optimization problems. Specifically, the goal is to find a subgraph G' of a given graph G so that G' meets certain connectivity requirements and optimizes a quality measure tailored to the respective application.

An important class of network design problems are *spanning tree* problems. Here, a solution of the problem has to satisfy only a minimum connectivity requirement. Specifically, there must be a node r—the root—such that every node is reachable from r by some path. Spanning trees use the minimum number of edges (links) among all subgraphs of G with this property.

A prominent spanning tree problem is Minimum Spanning Tree (MST), where every edge of the input graph G has an associated cost. The goal is to find a spanning tree whose total edge cost is minimum. A natural extension of this problem is Steiner Tree, where only a given subset T of so-called *terminal nodes* needs to be connected.

In contrast to MST, the quality measure of the spanning tree problem we investigate here is associated with the *nodes*, not the edges. This assumption is driven by applications in which the network nodes perform a certain function. We assume that nodes of higher degree have more sophisticated and thus also costlier functionality. Specifically, if we distinguish only between pure receivers (leaves of the tree) and routers (internal nodes), which are more expensive, we arrive at the MAXIMUM LEAF SPANNING TREE problem (MLST): given a graph G with root r, the task is to find an r-rooted spanning tree that maximizes the number of leaves. MLST is one of the classical NP-hard problems listed by Garey and Johnson [11].

We consider *digraphs*, that is, edges can only be traversed in one direction. Directed MLST is NP-hard, too, since it is a generalization of the undirected version. This motivates our interest in approximations. Although approximation algorithms are known for digraphs, their performance guarantees are not satisfactory. Therefore, we focus on the special case of *acyclic* digraphs, which is still NP-hard [2]. It turns out that we can exploit the special structure of acyclic digraphs to obtain guarantees that are significantly better than those known for general digraphs.

Previous results and related work. On undirected graphs, MLST is well-investigated. It is known that undirected MLST is NP-hard [11]. Galbiati et al. [10] showed that undirected MLST is even MaxSNP-complete, that is, there is no polynomial-time approximation scheme (PTAS) for this problem (if $\mathcal{P} \neq \mathcal{NP}$).

These negative results have stimulated the development of a series of approximation algorithms for MLST. Improving on their own earlier results, Lu and Ravi [13] developed a nearly-linear-time 3-approximation algorithm based on an *expansion* strategy. Basically, this strategy consists of growing a subforest F of the input graph by iteratively connecting nodes to a maximal set of edges so that F remains a forest. The expansion idea originally goes back to Kleitman and West [12] who considered graphs with bounded minimum degree and derived lower bounds for the maximum number of leaves of spanning trees in such graphs.

Solis-Oba [14] later proposed a linear-time algorithm based on the ideas of Kleitman and West [12] and Lu and Ravi [13]. By means of a clever analysis he showed that his algorithm is not only faster than the algorithm of Lu and Ravi but also gives a 2-approximation. So far, better results have been obtained only for special graph classes such as cubic graphs, the currently best being a 3/2-approximation algorithm [4].

Recently, there has been a lot of interest in the *directed* version of MLST. As often in network design, the directed case seems to be much harder than the undirected one. Drescher and Vetta [9] pointed out that the techniques that are successful for undirected graphs—namely, edge-swapping and expansion—fail for digraphs. They end up giving an algorithm for directed MLST with a ratio of $O(\sqrt{\mathrm{OPT}})$, which is considerably worse than the ratio 2 known for undirected graphs [14]. Daligault and Thomassé [8] improved upon this result by providing a 92-approximation algorithm. The techniques employed in both of the above algorithms differ completely from the approaches for the undirected case.

A large portion of the research on directed MLST has focused on the development of fixed-parameter tractable algorithms. The parameterized version of MLST includes an additional parameter k. The goal is to decide whether a given graph has a spanning tree with at least k leaves. The currently fastest fixed-parameter tractable algorithm is due to Gutin et al. [7] and has a running time of $3.72^k \cdot n^{O(1)}$ where n is the number of nodes in the input graph. There are also specialized fixed-parameter results for *acyclic* digraphs [1,2], that is, for the graph class considered in this work.

The currently fastest (unparameterized) exact algorithm was given by Binkele-Raible and Fernau [3]. It runs in $O^*(1.9043^n)$ time, where the O^*-notation neglects polynomial factors.

Our contribution. In this paper, we give two linear-time approximation algorithms for MLST on acyclic digraphs.

Our first result is a 4-approximation algorithm that makes use of a result of Daligault and Thomassé [8] who gave a lower bound on the number of leaves in a special class of digraphs.

In our second and main result we investigate the expansion approach, which has already led to several positive results for undirected MLST [12,13,14]. Applying the expansion idea to acyclic digraphs we obtain a 2-approximation algorithm. So we improve significantly upon the 92-approximation algorithm known for general digraphs and close up to the undirected case.

Our positive results are complemented by the observation that MLST in acyclic digraphs is MaxSNP-hard, that is, there is no PTAS for this problem (unless $\mathcal{P} = \mathcal{NP}$). That justifies the development of constant-factor approximation algorithms for MLST in acyclic digraphs.

To stress the relevance of our main result, let us compare MLST to Steiner Tree (ST), which can be considered paradigmatic among the tree-based network design problems. The best known algorithm for undirected ST has a ratio of roughly 1.39 [5]. This has to be compared to the best known algorithm for the directed case, which has a performance guarantee of $O(n^{\varepsilon})$ [6], for any $\varepsilon > 0$. There is a specialized approximation algorithm solving ST in acyclic digraphs but it yields the same result [15]. It can even be shown that for the acyclic case the approximation ratio is lower-bounded by $\Omega(\log n)$ [15]. To sum up, even the acyclic directed case of ST is significantly harder than the undirected one.

In terms of general graphs, ST and MLST behave similarly. In both cases, the results for undirected graphs are much better than the results for digraphs w.r.t. approximation. For *acyclic* digraphs, however, the problems exhibit significant difference. For ST, the acyclic case is provably harder than the undirected one and no improvement upon the general case has been obtained so far. In this paper, we provide an example of a tree-based network design problem (namely MLST) for which acyclicity can be exploited very well. It turns out that both algorithm *and* proof are a lot simpler in the acyclic than in the undirected case.

Since both our algorithms have the same (linear) asymptotic running time, the expansion algorithm supersedes the 4-approximation algorithm. Nevertheless, we think it is worth describing both algorithms since they are based on two conceptually different existing approaches that yield strong results. Finally, the analysis of the 4-approximation algorithm is considerably simpler than the analysis of the expansion algorithm.

We use n and m as shorthand for the numbers of nodes and edges of the given acyclic digraph G with root r. We denote an optimum spanning tree of G by T^* and the number of its leaves by OPT. Given an arbitrary spanning tree T of G, we denote the set of leaves of T by $L(T)$.

2 Indegree-Based Algorithm

In this section, we develop a 4-approximation algorithm based on (an extension of) a lemma by Daligault and Thomassé [8]. Let $V_{=1}$ be the set of nodes of indegree 1 in the given digraph G, and let $V_{\geq 2}$ be the set of nodes of indegree at least 2 in G.

Lemma 1 ([8]). *Any rooted acyclic digraph G has a spanning tree with at least $|V_{\geq 2}|/3$ leaves. Such a spanning tree can be computed in $O(m)$ time.*

Proof (Sketch). Daligault and Thomassé [8] prove the existence of a spanning tree with at least $(|V_{\geq 2}| + \deg(r) + 2)/3 \geq |V_{\geq 2}|/3$ leaves.

The proof of Daligault and Thomassé is constructive, and it is not hard to verify that the construction can be carried out in linear time. □

Our approximation algorithm is based on the following observation. Lemma 1 gives us already a good approximation in the case that $|V_{\geq 2}|$ is large enough in comparison to OPT. On the other hand, if $|V_{\geq 2}|$ is small then $|V_{=1}|$ is large. Since each of the nodes in $V_{=1}$ has exactly one incoming edge, *every* spanning tree (including the optimum one) must use these incoming edges. In other words, a large fraction of the edges are fixed, which leaves less freedom for the choice of the remaining edges. Intuitively we expect that even an arbitrary spanning tree gives us a good approximation.

Theorem 1. *The algorithm of Lemma 1 is a 4-approximation algorithm for MLST on acyclic digraphs.*

Proof. Let $\alpha := |V_{\geq 2}|/\text{OPT}$ and let T be the spanning tree output by the algorithm of Lemma 1. We now prove the following two bounds

$$|L(T)| \geq \frac{\alpha}{3}\text{OPT} \tag{1}$$

$$|L(T)| \geq (1-\alpha)\text{OPT}\,. \tag{2}$$

Bound (1) is an immediate consequence of the definition of α and Lemma 1.

For proving bound (2), we consider the graph $F = (V, E')$ with $E' = \{\, (u, v) \mid v \in V_{=1} \,\}$. The subgraph F of G is a forest containing only edges that are part of every spanning tree of G. Let L' be the set of leaves and isolated nodes of F, that is, the set of nodes with outdegree 0.

Consider an optimum spanning tree T^* of G. As argued above, F is a subforest of T^* and F has the same node set as T^*. Every leaf of T^* has outdegree 0 in T^*

and therefore also in F. Hence, L' contains the set of leaves of T^* as a subset. This yields $|L'| \geq \mathrm{OPT}$.

As F contains all edges ending in a node of $V_{=1}$, there are exactly $|V_{=1}|$ edges in F. The output tree T contains F as a subforest. Let us reconstruct T from F. To this end, we need $n - 1 - |V_{=1}| = |V_{\geq 2}|$ additional edges that are part of T but do not lie in F. By adding these edges to F, at most $|V_{\geq 2}|$ nodes in L' get connected with an outgoing edge. Hence, T contains at least $|L'| - |V_{\geq 2}|$ leaves. Using $|L'| \geq \mathrm{OPT}$ and the definition of α, we can conclude that

$$|L(T)| \geq |L'| - |V_{\geq 2}| \geq \mathrm{OPT} - \alpha\mathrm{OPT} = (1 - \alpha)\mathrm{OPT}\,.$$

This proves bound (2).

Now we balance bounds (1) and (2) to prove the approximation ratio 4. If $\alpha \geq 3/4$ then bound (1) yields $|L(T)| \geq \mathrm{OPT}/4$. On the other hand, if $\alpha \leq 3/4$ then bound (2) yields $|L(T)| \geq \mathrm{OPT}/4$. □

3 Expansion Algorithm

In this section, we present a linear-time 2-approximation algorithm for acyclic digraphs. Our algorithm and its analysis bear resemblances with the 3-approximation algorithm of Lu and Ravi [13] for undirected graphs. Therefore, we start with a brief outline of their algorithm.

3.1 Expansion Algorithm for Undirected Graphs

The algorithm of Lu an Ravi is based on a two-stage expansion strategy that works roughly as follows.

Stage I constructs a *leafy* subforest F of the input graph G. A forest is leafy if and only if any degree-2 node is adjacent to two nodes of degree at least 3. The leafy subforest F is constructed by *processing* the nodes of G iteratively in an arbitrary order. Processing a node u means to *expand* u if u has degree at least 3 after the expansion. The expansion of u adds to F a maximal set E_u of edges $(u, v) \in E(G)$ such that F remains a forest.

Stage II connects the subtrees created in stage I to a spanning tree of G in an arbitrary manner.

The total running time of the algorithm is $O(m\alpha(m, n))$, where $\alpha(\cdot\,, \cdot)$ is the inverse Ackermann function.

The 2-approximation algorithm of Solis-Oba [14] can be understood as a special case of the algorithm of Lu and Ravi, in which the nodes are processed in a particular order. More precisely, only leaves of F or singletons are expanded. Also, the connected components of F grow one by one rather than simultaneously. The particular node order does not only yield the better performance guarantee but also linear running time.

Our 2-approximation algorithm for acyclic digraphs closes up to the result of Solis-Oba for undirected graphs. Our analysis, however, is a lot simpler than that of Solis-Oba. In fact, our algorithm and its analysis are closer to the work of Lu and Ravi. We remark that a (straightforward) adaption of Solis-Oba's algorithm to DAGs does not yield better results (confer Section 3.4).

3.2 Expansion Algorithm for Acyclic Digraphs

Similar to the algorithm of Lu and Ravi, our expansion algorithm for acyclic digraphs consists of an expansion stage in which a subforest F of G is created and a connection stage where this forest F is completed to a spanning tree.

A detailed description of our algorithm can be found in Algorithm 1 and in the procedures expansion and connection that implement the expansion and the connection stage. We use a node-marking technique. If a node is marked in these stages it indicates that the node already has an incoming edge belonging to F or is the root of F.

The connection stage is similar to the undirected case. Basically, the connected components of F are connected to each other in an arbitrary manner.

The expansion stage, however, has to be adapted to digraphs appropriately. Instead of requiring degree at least 3 as in the undirected case, we expand a node if it obtains outdegree at least 2. Also the implementation of the expansion operation simplifies. Whenever an edge (u, v) is added to F, we only have to make sure that v has indegree 0 in F. We accomplish this by means of node markings. The algorithm of Lu and Ravi has to check whether u and v lie in different connected components. This is why we can improve the running time from $O(m\alpha(m, n))$ to $O(m)$.

Algorithm 1. MaxLeafTwoApprox(G)

```
Input: acyclic digraph $G$ with root $r$
Output: spanning tree $T$
mark $r$
$F \leftarrow$ expansion($G$)
$T \leftarrow$ connection($G, F$)
return $T$
```

Lemma 2. *Given an acyclic digraph G,* MaxLeafTwoApprox(G) *computes, in $O(m)$ time, a spanning tree of G.*

Proof. Recall that a node u is marked if and only if it has (exactly) one incoming edge or if $u = r$. No marked node can get further incoming edges. Hence, when the algorithm terminates, each node has either indegree 0 or 1 depending on whether it is marked or not. Since the connection stage marks all yet unmarked nodes, the result of the algorithm, F, is a subgraph of G that is acyclic (because G is) and in which every node except r has exactly one incoming edge. Thus, F is a spanning tree of G.

The linear running time can be achieved if the graph is represented by an adjacency list. Determining, for every $v \in V$, the set U_v of unmarked neighbors in procedure expansion takes $O(\sum_v \text{outdeg}(v)) = O(m)$ time in total.

Procedure expansion(G)

```
$F \leftarrow \emptyset$                { empty forest }
foreach node $v$ in $G$ do
    if $v \notin F$ then
        $F \leftarrow F + v$
    $U_v \leftarrow$ unmarked endpoints of outgoing edges of $v$ in $G$
    if $|U_v| \geq 2$ then
        $F \leftarrow F + U_v$
        foreach $u \in U_v$ do
            $F \leftarrow F + (v, u)$
            mark $u$
return $F$
```

Procedure connection(G, F)

```
foreach unmarked node $v$ do
    choose an arbitrary incoming edge $e$ of $v$ in $G$
    $F \leftarrow F + e$
    mark $v$
return $F$
```

In procedure connection, connecting all yet unmarked nodes with an arbitrary incoming edge takes $O(n)$ time. □

3.3 Performance Guarantee

The expansion stage (procedure expansion) of our algorithm creates a forest F that possibly contains isolated nodes. Let $\bar{F}$ be the forest obtained by removing all isolated nodes from F. The forest $\bar{F}$ consists of a set $\{T_0, \ldots, T_k\}$ of node disjoint, non-trivial subtrees $T_i = (V_i, E_i)$, $i = 0, \ldots, k$. Let r_i be the root of subtree T_i.

Since procedure expansion expands only nodes of outdegree at least 2, none of the trees T_i, $i = 0, \ldots, k$ contains an interior node of outdegree 1. In other words, $\bar{F}$ contains only leaves and nodes of outdegree at least 2. This implies that at least half of the nodes in $\bar{F}$ are leaves as we show now.

Lemma 3. *For $i = 0, \ldots, k$, any subtree $T_i \in \bar{F}$ has at least $(|V_i| + 1)/2$ leaves.*

Proof. It is well known that a binary tree on n nodes has at least $(n + 1)/2$ leaves. Internal nodes of outdegree greater than 2 can only increase the number of leaves. □

We first consider only the leaves of an optimal spanning tree T^* that lie in $V(\bar{F})$. A trivial upper bound on the number of these leaves is $|V(\bar{F})|$. The forest $\bar{F}$, in turn, has at least $(|V(\bar{F})| + k + 1)/2$ leaves (because of Lemma 3) and is thus a good intermediate step in obtaining our desired 2-approximation algorithm for MLST.

To prove the overall performance guarantee we face, however, the following two problems. The first problem is that the procedure connection may connect leaves of $\bar{F}$ with outgoing edges thereby "killing" those leaves. The second problem is that the optimum T^* may well have additional leaves outside of $\bar{F}$. Concerning the first problem, we now show that connection kills at most k leaves of $\bar{F}$.

Lemma 4. *Procedure* connection *creates a tree with at least* $|L(\bar{F})| - k$ *leaves.*

Proof. Let n_0 denote the number of outdegree-0 nodes in F (that is, leaves and isolated nodes) at the beginning of an iteration of procedure connection, and let n_{cc} denote the current number of (possibly trivial) connected components of F. Note that n_{cc} drops by 1 and n_0 increases by 1 in each iteration of connection. This means that the value of $n_0 - n_{\text{cc}}$ remains constant during the execution of the procedure.

This implies the claim since $n_0 - n_{\text{cc}} = |L(\bar{F})| - (k+1)$ holds at the beginning of the procedure and, hence, also at the end when we have that n_0 equals the number of tree leaves and $n_{\text{cc}} = 1$. □

The following lemma resolves the second above-mentioned problem—leaves outside of $\bar{F}$ cannot effectively increase the total number of leaves—and shows that the optimum kills *at least* k leaves in $V(\bar{F})$.

Lemma 5. *It holds that* $OPT \leq |V(\bar{F})| - k$.

Proof. Let T^* be an optimum spanning tree, and let R be the set of all roots $r_0, \dots, r_k$ of $\bar{F}$ that are different from the "global" root r. Our proof works as follows. We identify a unique node for each root $r_i \in R$, its *witness* $q(r_i)$. We will make sure that each witness is an *internal* node of T^* that lies in $\bar{V} := V(\bar{F}) \cup \{r\}$. This shows that T^* has at most $|V(\bar{F})| - k$ leaves in $\bar{V}$. It does not rule out, however, that T^* has additional leaves outside of $\bar{V}$. To this end, we will additionally identify, for each leaf l of T^* outside of $\bar{V}$, a witness $q(l)$, that is, a unique internal node in T^* that lies in $\bar{V}$. We will then show that the map q is injective. This proves the claim: if T^* has ℓ leaves outside of $\bar{V}$, then T^* can have at most $|V(\bar{F})| - k - \ell$ leaves inside of $\bar{V}$.

To define the map q, consider a node z that is either a leaf of T^* not contained in $\bar{V}$ or a root in R. We define the node $q(z)$ to be the closest ancestor of z in T^* (excluding z itself) that lies in $\bar{V}$. Since the root r lies in $\bar{V}$ such a witness $q(z)$ always exists.

Let z and z' be distinct nodes in the domain of q. It remains to show that $q(z) \neq q(z')$. Assume to the contrary that $q(z) = q(z')$. Let P and P' be the paths in T^* from $s := q(z) = q(z')$ to z and to z', respectively. We distinguish two cases.

First, we consider the case that s, z and z' lie on a common path in T^*. Then we can assume without loss of generality that z is an internal node on the path P', which implies that z is not a leaf in T^*. Since z lies in the domain of q, z must be the root of some subtree T_i in $\bar{F}$. In particular, $z \in \bar{V}$. Thus, z is an ancestor of z' in T^* that lies in $\bar{V}$ and is closer to z' than $q(z') = s$. This contradicts the choice of $q(z')$.

Now, we consider the case that s, z and z' do *not* lie on a common path. Then there is a node u at which the paths P and P' split; see Fig. 1. Let v and v' be the successors of u on paths P and P', respectively. Either v or v' is marked by procedure expansion. For, if v and v' are still unmarked when node u is processed then u will be expanded thereby marking v and v'. We assume without loss of generality that v is the node marked by procedure expansion.

We claim that $z \neq v$. If z is a leaf of T^* that lies outside of $\bar{V}$, then $z \neq v$ because v—being marked—lies in $\bar{V}$. If z is the root of a subtree T_i for any $i \in \{0, \dots, k\}$, then the claim follows because v has an incoming edge belonging to $\bar{F}$. Therefore, v is an ancestor of z in T^* that lies in $\bar{V}$ and is closer to z than $q(z)$. Again, this is a contradiction.

Both cases yield the desired contradiction. This completes the proof. □

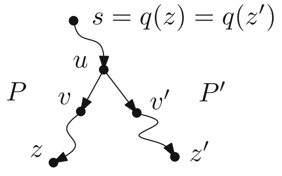


Fig. 1. Illustration of the case where s, z and z' do not lie on a common path

From Lemmas 2 to 5, we can deduce the main result of this paper.

Theorem 2. *The expansion algorithm for acyclic digraphs is a 2-approximation algorithm. It runs in linear time.*

Proof. Let T be the tree created by the expansion algorithm. Then we have

$$\frac{\mathrm{OPT}}{|L(T)|} \leq \frac{|V(\bar{F})| - k}{|L(\bar{F})| - k} \leq \frac{|V(\bar{F})| - k}{(\sum_{i=0}^{k}(|V_i| + 1)/2) - k} = \frac{2(|V(\bar{F})| - k)}{|V(\bar{F})| - k + 1} \leq 2,$$

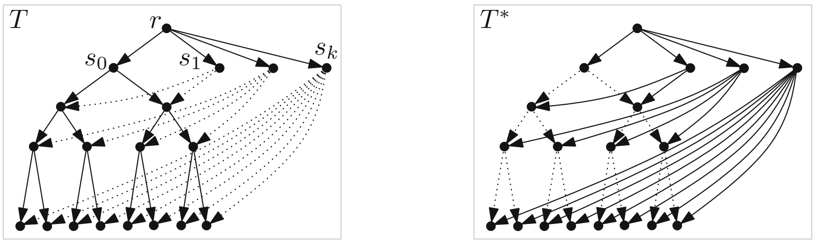
where the first inequality is due to Lemmas 4 and 5, the second inequality is due to Lemma 3, and the equality follows from the fact that $\sum_{i=0}^{k} |V_i| = |V(\bar{F})|$. □

3.4 Tight Example

We construct an infinite sequence $G_1, G_2, \dots$ of rooted acyclic digraphs such that the performance ratio of Algorithm 1 on this sequence tends to 2.

For any positive integer k, let the root r of G_k have $k+1$ successors $s_0, \dots, s_k$, see Fig. 2. The node s_0 is the root of a perfect binary tree B_k with $k+1$ levels $L_0 = \{s_0\}, L_1, \dots, L_k$. For $i = 1, \dots, k$, there is an edge from s_i to each node in level L_i. This completes the description of G_k.

Since the order in which our algorithm expands the nodes is not specified, we can assume that the algorithm first expands the root r and then the perfect



(a) result T of our algorithm applied to G_k (b) optimum spanning tree T^* of G_k

Fig. 2. Tight example G_k (drawn for $k = 3$) with two different spanning trees; solid edges represent tree edges, dotted edges represent non-tree edges

binary tree B_k. Then the spanning tree that our algorithm outputs has $2^k + k$ leaves in total; 2^k leaves in B_k plus the k leaves $s_1, \ldots, s_k$.

On the other hand, in the optimum solution T^* every node of B_k is a leaf. Thus, OPT $= 2^{k+1} - 1$. Clearly, the performance ratio $(2^{k+1} - 1)/(2^k + k)$ of our algorithm approaches 2.

Note that the above suboptimal spanning tree can also be obtained when we apply (a straightforward adaption of) Solis-Oba's algorithm [14] to G_k, that is, if we expand always at the leaves of the current subtree. This demonstrates that Solis-Oba's algorithm, too, does not yield better results for DAGs. Finally, this example remains valid even if the algorithm expands the nodes in *topological* order (which appears most natural).

4 MaxSNP-Hardness

Galbiati et al. [10] prove that the undirected MLST problem is MaxSNP-hard, which implies that there is no PTAS for undirected MLST (unless $\mathcal{P} = \mathcal{NP}$). Their hardness proof consists of a so-called *L-reduction* in which they use a special class of instances for undirected MLST. We now show that, for this special class, the undirected and the acyclic directed case are equivalent.

The undirected graphs that Galbiati et al. use in their proof have the structure depicted in Fig. 3 (a). These graphs consist of three levels of nodes. Each level has the same cardinality. Each node in level 1 is connected to the root r and each node in level 3 is connected to its counterpart in level 2. Additional edges connect only nodes between level 1 and level 2. Let G be an undirected graph with such a structure. Galbiati et al. show, that for any spanning tree T in G there is a spanning tree T' with the same number of leaves such that any node in level 1 is directly connected to the root r. We call the tree T' *valid*. Confer Fig. 3 (b) for a valid spanning tree. Galbiati et al. only use valid spanning trees in their L-reduction.

Given G, we construct an acyclic digraph D by orienting the edges of G so that r is the root of D and each edge starting in level i ends in level $i + 1$, where $i = 1, 2$. The remaining edges emanate from r; see Fig. 3 (c).

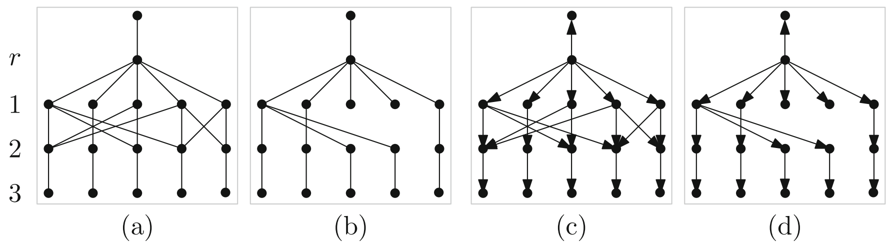


Fig. 3. Example of a graph used in the MaxSNP-hardness proof of Galbiati et al. [10]

We observe that there is a one-to-one correspondence between valid spanning trees of G and spanning trees of D. The unique orientation of any valid spanning tree of G yields a spanning tree of D with the same number of leaves. Conversely, each spanning tree T of D must contain all edges between r and level 1 and all edges between level 2 and level 3 since nodes in levels 1 and 3 have indegree one. Hence, the undirected tree corresponding to T is a valid spanning tree of G with the same number of leaves.

To sum up, the above equivalence shows that MLST on acyclic digraphs is MaxSNP-hard.

5 Concluding Remarks

Summarizing, we have given two linear-time approximation algorithms for solving the acyclic directed MLST problem with ratios of 4 and 2, respectively. The 4-approximation algorithm uses a result of Daligault and Thomassé [8] for MLST in acyclic digraphs. The 2-approximation algorithm is inspired by Lu and Ravi's 3-approximation algorithm for MLST in undirected graphs. Our result provides an example of a tree-based network design problem where acyclicity in digraphs can be exploited very well. Finally, we observed that MLST in acyclic digraphs is MaxSNP-hard.

References

1. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Parameterized Algorithms for Directed Maximum Leaf Problems. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 352–362. Springer, Heidelberg (2007)
2. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Spanning directed trees with many leaves. SIAM J. Discrete Math. 23(1), 466–476 (2009)
3. Binkele-Raible, D., Fernau, H.: A Faster Exact Algorithm for the Directed Maximum Leaf Spanning Tree Problem. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 328–339. Springer, Heidelberg (2010)
4. Bonsma, P.S., Zickfeld, F.: A 3/2-Approximation Algorithm for Finding Spanning Trees with Many Leaves in Cubic Graphs. SIAM J. Disc. Math. 25(4), 1652–1666 (2011)

5. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved LP-based approximation for Steiner tree. In: Proc. 42nd ACM Symp. Theory Comput. (STOC), pp. 583–592 (2010)
6. Charikar, M., Chekuri, C., Cheung, T.Y., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed Steiner problems. In: Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 192–200 (1998)
7. Daligault, J., Gutin, G., Kim, E.J., Yeo, A.: FPT algorithms and kernels for the directed k-leaf problem. J. Comput. Syst. Sci. 76(2), 144–152 (2010)
8. Daligault, J., Thomassé, S.: On Finding Directed Trees with Many Leaves. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 86–97. Springer, Heidelberg (2009)
9. Drescher, M., Vetta, A.: An approximation algorithm for the maximum leaf spanning arborescence problem. ACM Trans. Algorithms 6(3), 1–18 (2010)
10. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Inform. Process. Lett. 52(1), 45–49 (1994)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York (1979)
12. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discrete Math. 4(1), 99–106 (1991)
13. Lu, H.I., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithms 29(1), 132–141 (1998)
14. Solis-Oba, R.: 2-Approximation Algorithm for Finding a Spanning Tree with Maximum Number of Leaves. In: Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)
15. Zelikovsky, A.: A series of approximation algorithms for the acyclic directed Steiner tree problem. Algorithmica 18(1), 99–110 (1997)