The Power of Local Optimization: Approximation Algorithms for Maximum-Leaf Spanning Tree

Hsueh-I Lu* R. Ravi†

Brown University, Providence, RI 02912

Abstract

Given an undirected graph G, finding a spanning tree of G with the maximum number of leaves is NP-complete [5]. We use the simple technique of local optimization to provide the first approximation algorithms for this problem. Our algorithms run in polynomial time to produce locally optimal solutions. We prove that such locally optimal solutions to this problem are globally near-optimal. In particular, we prove that two such algorithms have performance ratios of 5 and 3. The latter algorithm employs more powerful local-improvement steps than the former and hence has higher running time. This may indicate an interesting trade-off between the performance ratios and the running times of the series of algorithms we describe.

1 Introduction

Given a simple, undirected graph $G = (V, E)$, suppose we wish to find a spanning tree of G with the maximum number of leaves. This problem finds applications in communication networks, circuit layouts and in other graph-theoretic problems[17]. An interesting application is mentioned in [12]: In [3], E. W. Dijkstra studied the problem of self-stabilizing a set of processors in spite of distributed control and proposed a solution based on mutual exclusion. A variant of this solution [18] seeks a spanning tree of a graph such that the product of the degrees of all the nodes in the tree is minimum. We can use a spanning tree with the maximum number of leaves as a heuristic solution for the latter problem.

The maximum-leaf spanning tree problem is NP-complete [5]. This suggests that there might be no polynomial time algorithms for this problem. So we seek fast approximation algorithms that provide a good worst case performance ratio on the quality of the solution. In this paper, we present the first such approximation algorithms for this problem. These algorithms use the simple technique of local optimization: perform a series of local-improvement steps until a local optimum is reached. This notion of applying local-improvement heuristics to hard optimization problems was around [2] even long before the invention of NP-completeness [8]. It has been applied heuristically to solve a variety of NP-hard problems in combinatorial optimization [14]. Among all such applications of this technique, there were no positive results which proved that this technique yielded an efficient approximation algorithm with an acceptable performance guarantee until the recent work of Fürer and Raghavachari [4]. They showed that a variation of this technique applied to the minimum-degree spanning tree problem runs in polynomial time to produce near-optimal solutions. Our work is inspired by that of Fürer and Raghavachari. We demonstrate that local optimization can be used to design polynomial-time approximation algorithms for the maximum-leaf spanning tree problem with constant performance ratios.

*Research supported by a Graduate Fellowship from the Ministry of Education, Taiwan R. O. C. Additional support provided by NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

†Research supported by an IBM Graduate Fellowship. Additional support provided by NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

We propose algorithms that perform local changes that increase the number of leaves in the resulting spanning tree. For the problem at hand, there is a natural notion of such a local change, namely, swapping a tree edge for a non-tree edge. This notion can also be extended to include swapping many tree edges for an equal number of non-tree edges in a single local-change operation. Our algorithms perform such local changes that increase the number of leaves until no improvement in the solution is possible and output a locally optimal solution as the approximate solution.

By varying the limit on the number of tree edges that can participate in a single local-change operation, we derive a series of approximation algorithms. Increasing this limit corresponds to allowing for more powerful local-improvement steps. Thus, the higher is this limit, the higher is the running time of the corresponding algorithm. A local-change operation involving at most k tree edges is termed a *k-change*. Intuitively, k-changes are costlier to implement than $(k-1)$-changes. However we expect an algorithm using k-changes to yield a better solution than an algorithm using only $(k-1)$-changes. In this paper, we corroborate this intuition and prove that local-improvement algorithms that use 1-changes and 2-changes have performance ratios of 5 and 3 respectively. The following are the main results of this paper.

Theorem 1 *There is an $O(n^4)$ algorithm using 1-changes for finding a spanning tree of any undirected graph G with n nodes such that the number of leaves in this spanning tree is at least as many as a fifth of the number of leaves in any spanning tree of G.*

Theorem 2 *There is an $O(n^7)$ algorithm using 2-changes for finding a spanning tree of any undirected graph G with n nodes such that the number of leaves in this spanning tree is at least as many as a third of the number of leaves in any spanning tree of G.*

We prove that the running time of the approximation algorithm using k-changes increases with k as follows.

Lemma 1 *The running time of the approximation algorithm using k-changes on a graph with n nodes is $O(n^{3k+1})$.*

The pattern exhibited in Theorems 1 and 2 leads us to conjecture that there is a trade-off between the running times and the performance ratios exhibited in the series of algorithms we have defined.

Conjecture 1 *The performance ratio of the approximation algorithm using k-changes is strictly better than that of the algorithm using $(k-1)$-changes.*

In the next section, we discuss related work. Then we present the algorithm formally and describe the basic properties of locally optimal solutions. We then derive a rough bound on the performance guarantee of solutions optimal under 1-changes and use these ideas to prove the bound in Theorem 1. We omit the proof of the performance guarantee in Theorem 2 in this abstract. We close with suggestions for future work.

2 Related work

The idea of applying local improvements to obtain good solutions to hard optimization problems is not new [2]. It has been applied to provide heuristic solution for a variety of hard problems in combinatorial optimization. Chapter 19 of [14] examines a few applications of this technique. Some notable examples are its applications to the graph partitioning [9] and the Traveling Salesperson problems [11]. A number of complexity results relating to the paradigm of local optimality and the time complexity of computing a locally optimal solution are presented in [7, 13]. In this paper, we are interested in the application of local-search techniques to design efficient approximation algorithms, namely, those that run in *polynomial* time and provide *provably* good solutions with values close to that of the optimum.

Fürer and Raghavachari [4] showed such an application of local search to the problem of computing a spanning tree of a given graph whose maximum degree is minimum. This is termed the

minimum-degree spanning tree problem and is NP-complete [5]. Fürer and Raghavachari show that local optimization can be applied to produce spanning trees and even Steiner trees of nearly minimum degree. They prove that the degree of the resulting solutions is within an additive logarithmic error of optimum. Their techniques have been generalized recently to obtain approximation algorithms for a variety of minimum-degree network design problems in [16]. In this paper, we add to the list of problems that are amenable to good approximate solutions by local-search methods.

Previous work on finding spanning trees with many leaves have focused on graphs with minimum degree at least k for some fixed $k \geq 3$. For such graphs, good lower bounds on the number of leaves achievable in a spanning tree are derived in [10, 6, 12, 17]. These lower bounds are typically proved algorithmically by constructing a spanning tree with the desired number of leaves. Thus these algorithms can be interpreted as approximation algorithms for the maximum-leaf spanning tree problem for special classes of graphs in which the minimum degree of a node is at least k. However, to the best of our knowledge, no previous approximation algorithms for this problem are known in the general case that we consider in this paper. The current best lower bound for the number of leaves achievable in a spanning tree of a n-node graph with minimum degree k is $(1 - b \ln k/k)n$ where b is any constant exceeding 2.5 and k is sufficiently large[10]. The best lower bounds for the cases $k = 3$ and 4 are $\Omega(n/4)$ and $\Omega(2n/5)$ respectively [10].

3 Definitions

In this section, we introduce some definitions that we use throughout the paper. The following definitions are relevant to any spanning tree of the input graph G. Henceforth we shall use the term degree of a node to refer to its degree in the tree under consideration. So leaves are just nodes of degree one. A node is *internal* if it is not a leaf. We call a node of degree greater than two a *high degree node*. A leaf is *special* if it is adjacent to a high degree node. All other leaves are termed *normal*. Thus normal leaves are exactly the leaves that are adjacent to nodes of degree two in the tree. A tree path containing only nodes of degree two is called a *2-path*. Its length is the number of nodes in it. A 2-path is *short* if its length is one; otherwise it is *long*. We shall refer to an edge $e = (u, v)$ as an edge *between* u and v.

For a spanning tree T, we use $\lambda(T)$ to denote the number of leaves in T. We omit the T where it is understood. We use n_i to denote the number of nodes of degree i. Suppose the input graph G has n nodes. We define $N_i = n - \sum_{1 \leq j < i} n_j$, the number of nodes with degree at least i. Thus for instance, N_3 is the number of high degree nodes. The number of short and long 2-paths are represented by P_ℓ and P_s respectively. Let λ_s and λ_n be the number of special and normal leaves respectively. We have the following observations.

Observation 1 *The number of high degree nodes in a tree is at most the number of leaves in the tree minus 2. In other words,* $N_3 \leq \lambda - 2$.

Observation 2 *The number of 2-paths in any tree is at most twice the number of leaves in the tree.*

$$P_\ell + P_s \leq N_3 + \lambda_n - 1 \leq 2\lambda - 3. \tag{1}$$

4 The Algorithm

In this section, we formally define k-changes and describe the approximation algorithms.

4.1 Edge Changes, Improvements and LOTs

Our algorithm starts with an arbitrary spanning tree T of the given graph G. Let $e = (u, v)$ be an edge in $G - T$ and f be an edge in the unique tree path of T connecting u and v. We call making e a tree edge and f a non-tree edge an (*edge*) *change* $(e; f)$ with respect to T. The spanning tree

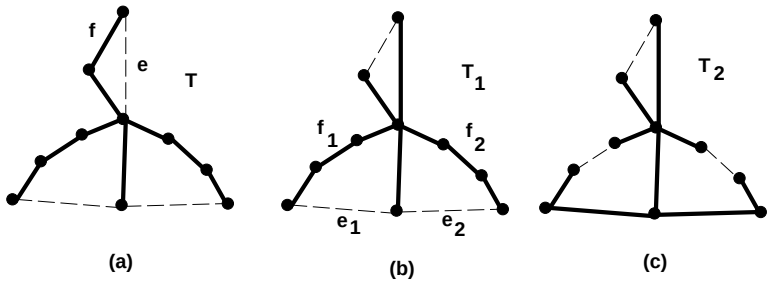

Figure 1: Examples of local improvements. In (a), the tree T is shown in dark lines and the non-tree edges are dotted. The change $(e; f)$ is a 1-improvement on T resulting in T_1 with one more leaf as shown in (b). T_1 does not admit any 1-improvement but admits the 2-improvement $(e_1, e_2; f_1, f_2)$ to give T_2 shown in (c).

obtained by applying change $(e; f)$ on T is denoted by $T(e; f)$. If $T(e; f)$ has more leaves than T, then we call the change $(e; f)$ an *improvement*.

We can generalize the notion of performing single edge swaps to allow for multiple edge swaps in a single change step. Suppose $e_0 = e$, $f_0 = f$, and $T_0 = T$. Let T_{i+1} denote $T_i(e_i; f_i)$, where $(e_i; f_i)$ is a change with respect to T_i for $0 \leq i < j$. We use $T(e_0, e_1, \ldots, e_{j-1}; f_0, f_1, \ldots, f_{j-1})$ to denote T_j. For any $k \geq j$, we define

$$(e_0, e_1, \ldots, e_{j-1}; f_0, f_1, \ldots, f_{j-1})$$

to be a k-*change* with respect to T. A k-change with respect to a spanning tree T is a k-*improvement* if the resulting spanning tree on applying the k-change has more leaves than T.

Note that a k-change is more powerful than k 1-changes. It is a batched version of any sequence of k 1-changes. In Figure 1, we illustrate a spanning tree T that admits a 1-improvement. On applying the 1-improvement we get a spanning tree T_1 that does not admit a 1-improvement but has a 2-improvement.

A k-*locally optimal tree* (k-LOT) of a given graph G is a spanning tree that does not admit any k-improvement. By definition, an i-LOT is also a j-LOT for any $j \leq i$. Since a spanning tree has exactly $n - 1$ edges, an optimal spanning tree with the maximum number of leaves is just a $(n - 1)$-LOT.

4.2 The algorithm and termination

Our idea is simply to use k-LOTs to approximate the maximum-leaf spanning tree for small values of k. We start with an arbitrary spanning tree and keep applying k-improvements on the current spanning tree until we obtain a k-LOT. Clearly, the second step dominates the time complexity. Whether a spanning tree is a k-LOT can be decided in polynomial time for constant k, since a spanning tree cannot have more than $\binom{\binom{n}{2}-n+1}{k} \times \binom{n-1}{k}$ k-changes. The algorithm itself has no more than $n - 3$ iterations because the number of leaves should be at least 2 to begin with and at most $n - 1$, and this number increases with every iteration. Therefore, the running time to obtain a k-LOT is $O(n^{3k+1})$ which is a polynomial in n for constant k. This proves Lemma 1.

Our major concern in this paper is the performance guarantee of this approximation algorithm. In other words, how much less can the number of leaves in a k-LOT be from that of the optimum? In the following sections we shall prove that the performance ratio of 1-LOTs is 5.

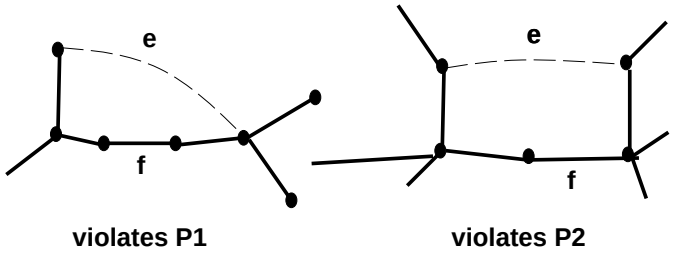

Figure 2: Illustrating properties P1 and P2 for 1-LOTs. In the above figure, dark edges represent tree edges and the dashed edge is a non-tree edge violating the property P1 or P2. In both the above cases, a 1-improvement $(e; f)$ can be identified.

5 Basic properties of 1-LOTs and a rough bound

In this section, we illustrate our proof method by proving a rough bound on the performance ratio of 1-LOTs. We begin by examining a few basic properties that are useful in proving the bounds.

5.1 Basic Properties

By definition, a 1-LOT has the following properties (See Figure 2).

P1 In a 1-LOT T, any cycle formed by adding a non-tree edge between a leaf and an internal node does not contain two adjacent nodes of degree two.

P2 In a 1-LOT T, any cycle formed by adding a non-tree edge between two internal nodes does not contain a node of degree two.

The following lemma shows that P1 and P2 together are necessary and sufficient conditions for 1-optimality.

Lemma 2 *A tree is a 1-LOT if and only if it satisfies P1 and P2.*

Proof ($\Rightarrow$) If P1 or P2 were not true for a tree T, then it is straightforward to identify a 1-improvement for T showing that T is not a 1-LOT.

($\Leftarrow$) We prove the contrapositive and show that if a spanning tree is not a 1-LOT then it violates either P1 or P2. If a spanning tree is not a 1-LOT then there exists a 1-improvement for this spanning tree. It is easy to see that any non-tree edge incident on two leaves cannot be involved in any 1-improvement. So any non-tree edge involved in a 1-improvement must be an edge between an internal node and a leaf or an edge between two internal nodes. In the first case, the tree violates property P1 and in the second case, it violates P2. ∎

Using Lemma 2, it is easy to derive the following corollaries.

Lemma 3 *In a 1-LOT T, there is no non-tree edge between a normal leaf and an internal node.*

Any 2-path in a spanning tree partitions the remaining nodes of the tree into two pieces on its removal. We refer to the sets of nodes representing these two pieces as the two *sides* of the 2-path.

Lemma 4 *In a 1-LOT T, only the first two nodes on a long 2-path, counting from a particular side, have non-tree edges to nodes in that side. Furthermore, both the endpoints of any non-tree edge between nodes in two sides of a long 2-path must be leaves.*

Lemma 5 *In a 1-LOT, there is no non-tree edge between two nodes in the same long 2-path.*

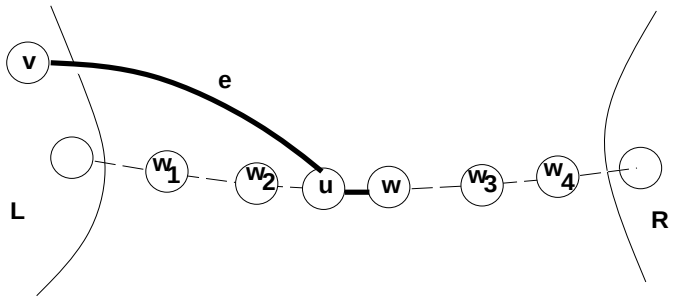

Figure 3: Proof of Lemma 6 by contradiction: The dark edges are edges of T' and the dashed edges are edges of T. If there are five leaves of T' in path $\mathcal{P}$, then the middle leaf must be connected in T' to one of the two sides via a non-tree edge e as shown above. This edge e violates the assumption that the tree T is a 1-LOT (Lemma 4).

5.2 The strategy: Proving a rough bound

In this subsection, we illustrate the basic strategy used in proving Theorem 1 by proving a weaker version of it. For this, we prove the following lemma.

Lemma 6 *Let G be an undirected graph and let T be a 1-LOT of G. Then for any spanning tree T' of G and any 2-path $\mathcal{P}$ of T, the number of leaves of T' in $\mathcal{P}$ is at most four.*

Proof The proof is by contradiction. Assume for a contradiction that there are no less than five leaves of T' in $\mathcal{P}$. Let five of them be w_1, w_2, w, w_3, and w_4, where w is the middle one along $\mathcal{P}$, and w_1 and w_2 are on the same side of w (See Figure 3). Note that by Lemma 5 there are no non-tree edges between nodes in $\mathcal{P}$. Let v be the node closest to w in T' that is not in $\mathcal{P}$. Let the last edge in the path from w to v in T' be $e = (u, v)$ where $u \in \mathcal{P}$. Since w is the middle leaf in T' along $\mathcal{P}$, the cycle closed by e contains at least two nodes of degree two, i.e., either w_1 and w_2 or w_3 and w_4. Without loss of generality, let these two nodes be w_1 and w_2. Since they are both in $\mathcal{P}$, the nodes between them in $\mathcal{P}$, if any, also have degree two in T. This implies that the cycle closed by e contains at least two adjacent nodes of degree two, which contradicts the fact that T is a 1-LOT by Lemma 4. ∎

Recall that $\lambda(T)$ denotes the number of leaves in a tree T.

Theorem 3 *Let G be an undirected graph. Let T^* be a maximum-leaf spanning tree of G and let T be a 1-LOT of G. Then $\lambda(T^*) \leq 10 \cdot \lambda(T)$.*

Proof Consider any spanning tree T' of G. We bound the number of leaves of T' that are (a) leaves, (b) high degree nodes, and (c) nodes in 2-paths in T independently. Note that these three categories account for every node in G. The total number of leaves in T is $\lambda(T)$ by definition. Also, the total number of high degree nodes in T is at most $\lambda(T)$ by Observation 1. Also, we can bound the *number* of 2-paths in T by $2\lambda(T)$ using Observation 2. Using Lemma 6, we can bound the number of leaves of T' in each 2-path of T by four. Thus the total number of leaves in T' is at most $\lambda(T) + \lambda(T) + 2\lambda(T) \cdot 4 = 10\lambda(T)$. Since this bound holds for any tree T', it also holds for T^* in particular and our theorem is proved. ∎

6 Performance Guarantee of 1-LOTs

In this section we prove the following restatement of Theorem 1.

Theorem 4 *Let G_0 be a given simple undirected graph. Let T_0^* be a maximum-leaf spanning tree of G_0 and let T_0 be a 1-LOT of G_0. Then $\lambda(T_0^*) \leq 5 \cdot \lambda(T_0)$.*

6.1 Augmentation

In order to prove Theorem 4, we augment G_0 together with T_0 as follows. For every normal leaf in T_0 adjacent to a short 2-path, we subdivide the tree edge incident on the normal leaf by inserting a new node of degree 2 in this edge. We have the following lemma.

Lemma 7 *Let G and T be the augmented versions of G_0 and T_0 respectively. If T_0 is a 1-LOT of G_0 then T is a 1-LOT of G. Furthermore, if T_0^* is a maximum-leaf spanning tree of G_0 and T^* is a maximum-leaf spanning tree of G, then $\lambda(T) = \lambda(T_0) \leq \lambda(T_0^*) \leq \lambda(T^*)$.*

Proof-sketch We can prove that T is a 1-LOT of G by contradiction using Lemma 3. Now we consider the relations among $\lambda(T)$, $\lambda(T_0)$, $\lambda(T_0^*)$, and $\lambda(T^*)$. The equality is trivial since augmentation does not change the number of leaves. The first inequality follows from the definition of T_0^*. The last inequality can be proved by constructing a spanning tree of G with at least $\lambda(T_0^*)$ leaves. ∎

6.2 Partition into regions

For the sake of proof, based on the 1-LOT T, we partition the nodes in G depending on whether they occur in a long 2-path in T. We further partition the nodes of G that are not in any long 2-path in T into *regions*. These regions are exactly the connected components of T obtained on removing the nodes in long 2-paths from T. A region is *special* if it contains at least one special leaf of T; otherwise it is *normal*. Let us use S and R to stand for the number of special and normal regions, respectively. The degree of a region is the number of long 2-paths incident on it. Note that as a result of augmentation, each normal leaf is a single-node normal region by itself. So a normal region is either a single leaf or a set of high degree nodes interconnected via short 2-paths. We term these two kinds of normal regions *external* and *internal*, respectively. It is easy to prove the following relation for a 1-LOT T. Recall that P_ℓ is the number of long 2-paths in T.

$$P_\ell \quad = \quad S + R - 1 \tag{2}$$

6.3 Accounting for leaves in long 2-paths

The key observation which helps proving the tighter bound in Theorem 4 is that not every long 2-path in T contributes four leaves in any spanning tree as proved in Lemma 6. The following lemma is a refined version of Lemma 6 and is proved in a similar manner.

Lemma 8 *Let T' be a spanning tree of G and $\mathcal{P}$ be a long 2-path of a 1-LOT T of G. Suppose $\mathcal{P}$ is incident on two regions $\mathcal{L}$ and $\mathcal{R}$ in T. The number of nodes in $\mathcal{P}$ which are leaves of T' is at most (i) four, if $\mathcal{L}$ and $\mathcal{R}$ are both special, (ii) three, if only one of $\mathcal{L}$ and $\mathcal{R}$ is special, and (iii) two, if $\mathcal{L}$ and $\mathcal{R}$ are both normal.*

We can think of the above lemma as providing an upper bound on the number of leaves of T' in any long 2-path $\mathcal{P}$ of T in the following way. Assume that a normal region contributes a charge of one and a special region a charge of two to the upper bound for any incident long 2-path. Then the upper bound on the number of leaves of T' in any long 2-path $\mathcal{P}$ is exactly the sum of the charge contributions of both its adjoining regions.

6.4 Invalid regions and better bounding

Even the careful counting of the leaves of T' in long 2-paths in the above lemma is not sufficient to prove Theorem 1. For example, consider an internal normal region. To connect any node in such a region in T' to any region outside, we must use one of the long 2-paths incident on it since there are no non-tree edges going from this region to any other region. This implies that this normal region

cannot charge a contribution of one to *all* its incident long 2-paths. To capture this we define the notion of an *invalid* region.

Definition: A region of T is invalid in T' if it is either an internal normal region or it is a special region and every special leaf in this region is also a leaf in T'. We now define long 2-paths for which the bounds on the number of leaves of T' is more stringent than those stated in Lemma 8. Let $\mathcal{P}$ be a long 2-path of T. We call $\mathcal{P}$ *deficient* in T' if the number of nodes in $\mathcal{P}$ that are leaves in T' is at most two less than the upper bound stated in Lemma 8. We can prove the following using the properties of an invalid region of a 1-LOT.

Lemma 9 *Let k be the number of regions of a 1-LOT T that are invalid in T'. Then we can identify at least $(k-1)$ distinct long 2-paths of T that are deficient in T'.*

Now we prove our final bound on the number of leaves of any spanning tree T' that are nodes in long 2-paths in a 1-LOT T.

Lemma 10 *For any spanning tree T', the number of leaves of T' that are nodes of long 2-paths in a 1-LOT T is at most*

$$4S + 3\lambda_n - \lambda_n^* - 2S^*,$$

where λ_n^* is the number of normal leaves of T which are also leaves in T', and S^* is the number of special regions of T that are invalid in T'.

Proof Let S_i and R_i denote the number of special and normal regions of degree i for any $i \geq 1$, respectively. Then we have $S = \sum_{i\geq 1} S_i$ and $R = \sum_{i\geq 1} R_i$. If we regard T as a tree of regions, the sum of degrees of regions is at most two times the number of long 2-paths. Namely, $\sum_{i\geq 1} i \cdot S_i + \sum_{i\geq 1} i \cdot R_i \leq 2P_\ell$. We interpret Lemma 8 using the notion of charge contributions of regions to incident long 2-paths. Using this interpretation, the total number of leaves of T' in long 2-paths of T is at most $2\sum_{i\geq 1} i \cdot S_i + \sum_{i\geq 1} i \cdot R_i \leq 4P_\ell - \sum_{i\geq 1} i \cdot R_i$, by the previous inequality.

Note that a single normal leaf is also a normal region. If a normal leaf of T is also a leaf in T', then we can verify that this normal region cannot contribute any charge in the scheme above to its incident long 2-path. So we subtract one for each such region to get a bound of $4P_\ell - \sum_{i\geq 1} i \cdot R_i - \lambda_n^* = 4(R + S - 1) - \sum_{i\geq 1} i \cdot R_i - \lambda_n^*$ using (2).

Note that the number of invalid regions in T' is precisely sum of the number of internal normal regions and invalid special regions. The number of such regions is $R - R_1 + S^*$. By Lemma 9 and the definition of deficiency, we can then subtract at least $2(R - R_1 + S^* - 1)$ from the bound derived above. Thus the final upper bound on the number of leaves in T' from long 2-paths in T is

$$4(R+S) - \sum_{i\geq 1} i \cdot R_i - \lambda_n^* - 2(R - R_1 + S^*).$$

Simplifying using $R = \sum_{i\geq 1} R_i$ and $R_1 = \lambda_n$ proves the lemma. ∎

6.5 Proof of Theorem 4

We allow all high degree nodes in T and all nodes in short 2-paths of T to be leaves in T^*. Using Lemma 10 with $T' = T^*$, we infer that the number of leaves in T^* on long 2-paths of T is at most $4S + 3\lambda_n - \lambda_n^* - 2S^*$. The remaining nodes in G are just special leaves and normal leaves. Since exactly λ_n^* normal leaves are still leaves in T^*, and at least $S - S^*$ special leaves are not leaves in T', the number of leaves in T^* is bounded by

$$N_3 + P_s + (4S + 3\lambda_n - \lambda_n^* - 2S^*) + (\lambda_s - (S - S^*) + \lambda_n^*).$$

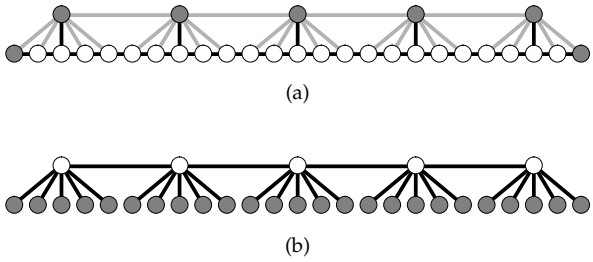

Figure 4: (a) An example of 1-LOT. (b) The maximum-leaf spanning tree of the graph in (a).

Note that $S \leq \lambda_s$ because each special region contains at least one special leaf. From (2) and (1), we get $S + P_s \leq N_3 + \lambda_n - R$. It follows that

$$\begin{array}{lll} \lambda(T^*) & \leq N_3 + \lambda_s + 3\lambda_n + 3S + P_s & \\ & \leq (N_3 + 3\lambda_s + 3\lambda_n) + (N_3 + \lambda_n - R) & \text{using } S \leq \lambda_s \\ & = (2N_3 + 3\lambda_s + 3\lambda_n) + (R_1 - R) & \leq 2N_3 + 3\lambda_s + 3\lambda_n \\ & \leq 2n_1 + 3n_1 (\text{ since } \lambda_s + \lambda_n \text{ is just } n_1) & = 5\lambda(T). \end{array}$$

By Lemma 7, we know

$$\frac{\lambda(T_0^*)}{\lambda(T_0)} \leq \frac{\lambda(T^*)}{\lambda(T)} \leq 5.$$

Therefore, Theorem 4 is proved. ∎

We conclude this section by showing an example of a 1-LOT in which the bound of 5 in Theorem 1 is approachable arbitrarily closely. It is easy to verify that the tree in Figure 4-(a) is a 1-LOT. The maximum-leaf spanning tree for this graph is shown in Figure 4-(b). Clearly, the longer this 1-LOT is, the closer to five the ratio is.

7 Future work

We believe that the performance ratio of 2-LOTs is better than what we prove in Theorem 2 and conjecture the following.

Conjecture 2 *The performance guarantee of 2-LOTs is 2.5.*

Working on the cases $k \geq 3$ to obtain better performance guarantees is a direction for future work.

Papadimitriou and Yannakakis [15] identified a class of NP-hard optimization problems interreducible to one another using approximation-preserving reductions and thus took a step towards classifying NP-complete problems with respect to the hardness of approximating them. They called this class MAX-SNP. Recent work [1] has shown that problems complete for this class do not permit a fully polynomial-time approximation scheme unless $P = NP$. It is an intriguing open problem to determine if the maximum-leaf spanning tree problem is complete for MAX-SNP or if it does allow a $(1 + \epsilon)$-approximation. Along this direction, it would be interesting to examine if k-LOTs can be used in deriving such a polynomial-time approximation scheme.

Acknowledgments

We gratefully acknowledge the support, guidance and encouragement of our advisor, Prof. Philip Klein.

References

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy, "Proof Verification and Hardness of Approximation Problems," (to appear in the Thirty-third annual IEEE Foundations of Computer Science Conference, 1992).

[2] G. A. Croes, "A method for solving traveling-salesman problems," *OR 6*, No. 6, (Nov.-Dec. 1958), pp. 791-812.

[3] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Comm. ACM 17*, 11 (Nov. 1974), pp. 643-644.

[4] Martin Fürer and Balaji Raghavachari, "Approximating the minimum degree spanning tree to within one from the optimal degree", *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 317-324.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[6] J. R. Griggs, D. J. Kleitman, and A. Shastri, "Spanning trees with many leaves in cubic graphs, *J. Graph Theory*, 13 (1989), pp. 669-695.

[7] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?," *J. Comp. Syst. Sci.*, 37(1988), pp. 79-100.

[8] R. M. Karp, "Reducibility among combinatorial problems," in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*. Plenum Press, New York (1972), pp. 85-103.

[9] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *BSTJ* 49 (1970), pp. 291-308.

[10] D. J. Kleitman and D. B. West," Spanning trees with many leaves," *SIAM J. Disc. Math.* Vol. 4, No. 1, (February 1991), pp. 99-106.

[11] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the Traveling-Salesman Problem," *Oper. Res.* 21 (1973), pp. 498-516.

[12] C. Payan, M. Tchuente, and N. H. Xuong, "Arbres avec un nombres maximum de sommets pendants, *Discrete Math.*, 49 (1984), pp. 267-273.

[13] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis, "On the complexity of local search (Extended Abstract)," in *Proc. of the twenty second annual ACM Symposium on the Theory of Computing*, pp. 438-445 (1990).

[14] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc. (1982).

[15] C. H. Papadimitriou, and M. Yannakakis, "Optimization, Approximation, and Complexity classes," in *Proc. of the twentieth annual ACM Symposium on the Theory of Computing*, pp. 229-234 (1988).

[16] R. Ravi, B. Raghavachari, and P. N. Klein, "Approximation through local optimality: Designing networks with small degree," in *Proceedings, Twelfth annual conference on Foundations of Software Technology and Theoretical Computer Science* (1992), to appear.

[17] J. A. Storer, "Constructing full spanning trees for cubic graphs," *Inform. Process. Lett.* 13 (1981), pp.8-11.

[18] M. Tchuente, "Sur l'auto-stabilisation dans un réseau d'odinateurs," *R. A. I. R. O. Informatique Théorique 15*, (1) (1981), pp. 47-66.