Proceedings of International Conference
on Algorithms

An Approximation Algorithm for Bounded Length Tree Linear Broadcast Routing

H.F. Ting, W.H. Wong
Department of Computer Science
The University of Hong Kong
Email: hfting@cs.hku.hk, whwong@cs.hku.hk

Abstract

In this paper, we study the problem of bounded length tree linear broadcast routing (BLTLB). Chou and Gopal [2] proved that this problem is NP-complete. Our contribution is to present the first approximation algorithm which achieves a constant performance ratio.

Keywords. Fiber Optic Communication Network, Broadcast Routing, Approximation Algorithm, Lower Bound.

1 Introduction

With the advance of fiber optic technology, the bandwidth of communication network has increased tremendously. However, the pace of enhancing the processing power of the computers lags far behind. Hence, we can no longer assume that the processing time is negligible compared with the communication time and this violates the fundamental assumption of the traditional packet-switched network. To relay a message in a packet-switched network, a processing node needs to replicate the message and choose a suitable outgoing link for the message. The replication involves hundreds or even thousands of machine instructions and memory accesses. So the processors waste too much time in relaying messages. To solve this problem, new hardware (e.g., ATM switch) is designed to implement the switching function automatically and efficiently. The processors need not be involved in relaying the message.

With this new hardware, the network becomes very different from the traditional packet-switched model. To capture the characteristics of this new development, a new model was proposed by Gopal, Cidon, and Kutten [1]. In the model, every node has two components: a network control unit (NCU) and a switching subsystem (SS). The NCU is a general-purpose processor and the SS is a high-speed special-purpose hardware to off-load the packet-switching function. In each node, the NCU is connected to the SS by a link. The SS of different nodes are connected by communication links and there is no direct link between the NCU of different nodes. Every node has a unique address. These addresses will be used for routing messages. Suppose we send a message from node A to node B through some intermediate nodes. The addresses of the destination node B and all the intermediate nodes are placed in the message as a prefix. Whenever an SS receives a message, the first address information is removed from the message and it is used to determine the outgoing link. The message, with the first address information removed, is relayed to the appropriate link. Moreover, the message can be forwarded to the corresponding NCU so that all the NCUs along the routes can also read the message.

The new telecommunication technology has great impact not only on the model of the network but also on the algorithms for network problems. In this paper, we study one of the basic network applications, message broadcasting, i.e. sending messages from a source node to every other node in the network. In this problem, we assume that, for every link, there is a positive cost for sending a message through that link. We are to find a way to send the message from the source to all the nodes in the network such that the total induced cost is minimum. In the traditional packet-switched network, the problem can be solved by finding a minimum-cost spanning tree of the network and the message is forwarded over the edges of the tree. But this is not a solution for the new model because the SS does not support replication of message and simultaneous sending of the same message to all neighbors of a node. Instead, every copy of the same message to be broadcasted should be generated by the source and follow a linear route to the destinations. Hence, to broadcast a message in the new model, we have to find a set of paths (not necessary simple) starting from the source node and covering all the nodes in the network so that the total induced cost is minimized. Moreover, there may be some constraints on the path-set because of the limitations of

the network, e.g., bandwidth limitations on the communication links, the space constraint on the header of the message. With bandwidth limitations, every communication link e has a capacity $c(e)$ and there can be at most $c(e)$ paths passing through e simultaneously. With a limit on the header space of the message, the number of nodes along a route is limited and so the path length should not exceed a predetermined value $\mathcal{L}$. Given any set of paths emanated from the source, we say that it is a feasible broadcast path-set if every node is covered by at least one path in the set and all the paths satisfy the above mentioned constraints. The constrained linear broadcast routing problem is to find a feasible broadcast path-set with minimum weight. However, this problem is very difficult. In fact, to determine whether there is a feasible broadcast path-set for a network with tree topology is already NP-complete.

1.1 Previous work

In [2], Chou and Gopal studied the problem without the path length constraints and the capacity constraints (i.e. $\mathcal{L} = \infty$, and for every edge e, $c(e) = \infty$). They proved that this simplified problem is still NP-complete for general network topologies. They gave an $O(N^2)$ time algorithm solving the unconstrained problem for tree topologies, where N is the number of nodes in the tree. The algorithm was improved by Bitan and Zaks [3] to $O(N)$ time. The next step towards the problem is to find a minimum weight broadcast path-set satisfying only one of the constraints. In [4], Bitan and Zaks studied the problem with the capacity constraints and designed an $O(N^2)$ algorithm which returns a minimum weight broadcast path-set. Later, we [5] improved the time complexity of their algorithm to $O(N\frac{m}{\log_d m})$ time, where d is the maximum degree of the tree, $m = \min(h, n_c)$, h is the height of the tree, and n_c is the number of distinct capacities in the network. In this paper, we study the problem with the other constraint, namely path length constraint. However, this problem is NP-complete even for tree topologies. It is unlikely to find a polynomial time algorithm for the problem. Hence, it is desirable to find an approximation algorithm which returns a near-optimal solution. In this paper, we present an approximation algorithm which returns a broadcast path-set whose total weight is at most three times the weight of the optimal solution.

2 Problem formulation

The problem studied in this paper is called the **Bounded Length Tree Linear Broadcast Routing** (BLTLB) problem. The input of the problem is specified by a tuple $(T, w, \mathcal{L})$ where

- $T = (V, E)$ is a rooted tree where V represents the set of processors and E represents the set of bidirectional communication links;
- $w : E \rightarrow \mathcal{R}^+$ is a weight function which measures the communication cost of sending messages over the links;
- $\mathcal{L}$ is a bound on the maximum length of the routes allowed. (Note that $\mathcal{L}$ must be no less than the depth of the tree.)

We formalize the concept of feasible broadcast path-set with path length constraint $\mathcal{L}$ as follows. Given any set Ψ of paths (not necessary simple) on T rooted at s, we call it an $\mathcal{L}$-cover for $(T, w, \mathcal{L})$ if

1. all the paths start from the root;
2. for every node $v \in V$, there exists at least one path $\rho \in \Psi$ which covers v (i.e. $\rho = \langle sv_1v_2 \cdots v \cdots v_k \rangle$);
3. for every path $\rho \in \Psi$, the length of $\rho \leqslant \mathcal{L}$.

The **size** of an $\mathcal{L}$-cover is defined to be the number of paths in the cover. For any path $\rho = \langle v_1v_2 \cdots v_k \rangle$ in Ψ, the weight of ρ is equal to $\sum_{1 \leqslant i \leqslant k-1} w((v_i, v_{i+1}))$. The weight of Ψ is defined to be the sum of the weight of all the paths in Ψ.

> *The BLTLB problem is to find an $\mathcal{L}$-cover for $(T, w, \mathcal{L})$ with minimum weight.*

To emphasize the fact that all the paths in any $\mathcal{L}$-cover start from the root, are not necessary simple and their lengths are no greater than $\mathcal{L}$, we call this kind of paths **$\mathcal{L}$-trails** in the rest of the paper.

3 Our result

In [2], Chou and Gopal proved that the decision version of the BLTLB problem is NP-complete. It is very unlikely that there exists a polynomial time algorithm for the problem. However, due to the practical significance of the problem in network applications, it is too important to abandon merely because obtaining an optimal solution is intractable. It is desirable to design an approximation algorithm which returns a provable good though not optimal solution. A straight forward heuristic may be to cover each leaf by a separate $\mathcal{L}$-trail. Then all the nodes in T are covered. However, the performance of this heuristic may be very poor because the ratio of the weight of the $\mathcal{L}$-cover to the weight of the optimal one can be as bad as $O(|V|)$. In this paper, we present the first approximation algorithm for the problem which achieves a constant ratio. To be specific,

we present an approximation algorithm which finds a solution to the BLTLB problem and prove that

$$\max_{(T,w,\mathcal{L})} \frac{W(T,w,\mathcal{L})}{W_{opt}(T,w,\mathcal{L})} \leqslant 3,$$

where $W(T,w,\mathcal{L})$ is the weight of the $\mathcal{L}$-cover returned by our algorithm on input instance $(T,w,\mathcal{L})$ and $W_{opt}(T,w,\mathcal{L})$ is the weight of the optimal $\mathcal{L}$-cover for $(T,w,\mathcal{L})$.

The crux of the algorithm is to partition the leaves of T into groups such that (1) a single $\mathcal{L}$-trail cannot cover all the leaves in a group; (2) all the leaves in a group can be covered by two $\mathcal{L}$-trails. The first property implies a lower bound on the number of $\mathcal{L}$-trails required to cover all the nodes in T, and this further implies a lower bound on the weight of any $\mathcal{L}$-cover. The second property suggests a way to find an $\mathcal{L}$-cover with good performance ratio.

4 $\mathcal{L}$-Partition

In this section, we define a special partition, called $\mathcal{L}$-partition, of the leaves and proved that if the leaves of T have an $\mathcal{L}$-partition of size n, they can never be covered by less than $\frac{2}{3}n$ $\mathcal{L}$-trails. From this constraint, we derive a lower bound on the weight of any $\mathcal{L}$-cover of T.

First of all, let us have some notations. Let $T = (V,E)$ be any rooted tree. For any node v in T, we denote by d_v the depth of v in T (the root has depth zero, its children have depth one and so on), T_v the subtree rooted at v, and L_v the set of leaves in T_v. Given any subset of leaves, G, let $T|_G = (V|_G, E|_G)$ denote the subtree of T induced by G, where the root of $T|_G$, w, is the least common ancestor of all the leaves in G; $V|_G$ is the set of nodes in T which are descendants of w and have leaf descendants in G; $E|_G$ is the subset of edges (u,v) of E where both u and v are in $V|_G$. Note that, contrast to the usual definition, we do not contract any node with degree two in $T|_G$. (We cannot contract any edge because it changes the length of the paths.)

Definition 4.1 *Let G be any subset of leaves of T and $P = \{G_1, G_2, \ldots, G_n\}$ be a partition of G. We say that P is an $\mathcal{L}$-**partition** for T if*

1. *all the $E|_{G_i}$'s are disjoint;*
2. *for every G_i, no single $\mathcal{L}$-trail can cover all the leaves in G_i.*

*Given any tree T, it is **$(\mathcal{L},n)$-difficult** if and only if it has an $\mathcal{L}$-partition of size n.*

Given any set of leaves H, define $\mathrm{gd}(H) = \max_{v\in H}\{d_v\}$, and $\mathrm{Len}(H) = 2d_r + 2S - \mathrm{gd}(H)$, where r is the root of $T|_H$ and S is the number of edges in $T|_H$. For every $e = (u,v)$ in T, define $\mathrm{c_P}(e)$ to be the number of G_i that is a subset of L_v. Consider the tree T in

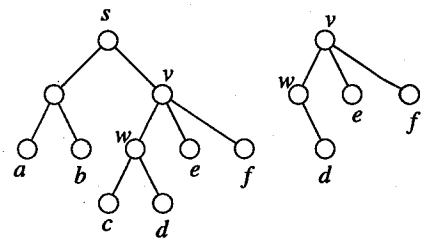


Figure 1: T and $T|_H$.

Figure 1, if $H = \{d,e,f\}$, the tree on the right hand side is $T|_H$. Suppose $\mathcal{L} = 7$. We have $\mathrm{gd}(H) = 3$, $d_r = 1$, $S = 4$ and $\mathrm{Len}(H) = 7$. We can cover all the leaves in H by an $\mathcal{L}$-trail, $\langle s,v,f,v,e,v,w,d\rangle$. However, if $H' = \{c,d,e,f\}$, then $\mathrm{Len}(H') = 9$ and a single $\mathcal{L}$-trail cannot cover all the leaves in H'. Followings are some important facts.

Fact 4.2 *For any set of leaves H, H can be covered by a single $\mathcal{L}$-trail if and only if* $\mathrm{Len}(H) \leqslant \mathcal{L}$.

Fact 4.3 *Let $P=\{G_1, G_2, \ldots, G_n\}$ be an $\mathcal{L}$-partition for T. We have $\sum_{e\in T} c_P(e) = \sum_{i=1}^{n} d_{r_i}$, where r_i is the root of $T|_{G_i}$,*

Proof (Sketch): For each G_i, it contributes one unit to $\mathrm{c_P}(e)$ for every edge e along the path from the root of T to r_i, and nothing to the other edges. Hence, G_i contributes d_{r_i} units to $\sum_{e\in T} \mathrm{c_P}(e)$. The fact follows. □

Lemma 4.4 *Let $P=\{G_1, G_2, \ldots, G_n\}$ be an $\mathcal{L}$-partition for tree T. For every edge $e = (u,v)$, T_v is $(\mathcal{L}-d_v, c_P(e))$-difficult.*

Proof: We claim that $P' = \{G_i \mid G_i \in P \text{ and } G_i \subseteq L_v\}$ is an $(\mathcal{L}-d_v)$-partition for T_v. Then it follows immediately that T_v is $(\mathcal{L}-d_v, \mathrm{c_P}(e))$-difficult. (Note that $|P'| = \mathrm{c_P}(e)$.) First, we see that $T_v|_{G_i}$ are edge disjoint because $T|_{G_i}$ are edge disjoint. Second, we argue that for any G_i, there is no $(\mathcal{L}-d_v)$-trail starting from v which covers all the leaves in G_i. Otherwise, we can extend this $(\mathcal{L}-d_v)$-trail from v to the root of T and form an $\mathcal{L}$-trail starting from the root and covering all the leaves in G_i. This contradicts the fact that G_i is in an $\mathcal{L}$-partition for T. □

4.1 Lower bound on the size of any $\mathcal{L}$-cover

In this section, we prove that, for any constant $\mathcal{L}$, if T is $(\mathcal{L}, n)$-difficult, then the size of any $\mathcal{L}$-cover of T is more than $\frac{2}{3}n$. To simplify our discussion, we assume that the root of T, u, has degree at least two. If, contrast to our assumption, there is only one edge (u, v) coming out of the root, then we can restrict our attention to T_v, which is, from Lemma 4.4, ($\mathcal{L}$-1,n)-difficult. It should be obvious that any lower bound on any ($\mathcal{L}$-1)-cover of T_v is a lower bound on any $\mathcal{L}$-cover of T.

We first consider the case when there is some restriction on the $\mathcal{L}$-cover. Then we handle the general case. Let $P=\{G_1, \ldots, G_n\}$ be an $\mathcal{L}$-partition for T. For any $\mathcal{L}$-cover of T, we say that it is an $\mathcal{L}$-cover that is simple w.r.t. P if and only if, for every G_i, there is at most one $\mathcal{L}$-trail stops at some node in $T|_{G_i}$. Otherwise, it is an $\mathcal{L}$-cover that is complex w.r.t. P. We first show that the size of any $\mathcal{L}$-cover of T that is simple w.r.t. P is at least n+1. Note that the bound does not hold for $\mathcal{L}$-cover that is complex w.r.t. P (See Appendix). For $\mathcal{L}$-cover that is complex w.r.t. P, we derive a bound of $\frac{2}{3}n$ on its size.

Theorem 4.5 *If a tree T is $(\mathcal{L}, n)$-difficult and $P=\{G_1, G_2, \ldots, G_n\}$ is an $\mathcal{L}$-partition for T, then the size of any $\mathcal{L}$-cover of T that is simple w.r.t. P is at least $(n+1)$.*

Proof: We prove the theorem by induction on n. Consider the case when T is $(\mathcal{L}, 1)$-difficult and $P = \{G_1\}$ is an $\mathcal{L}$-partition for T. By definition, G_1 cannot be covered by an $\mathcal{L}$-trail. So at least two $\mathcal{L}$-trails are required to visit all the leaves in G_1, i.e. the size of any $\mathcal{L}$-cover of T that is simple w.r.t. P is at least 2 (for any constant $\mathcal{L}$). Therefore, the hypothesis is true for $n = 1$.

Assume that the hypothesis is true for $n = k$. In other words, for any constant $\mathcal{L}$, if tree T is $(\mathcal{L}, k)$-difficult and C is an $\mathcal{L}$-cover that is simple w.r.t. some $\mathcal{L}$-partition for T, then the size of C is at least $k + 1$. Now consider any tree T which is $(\mathcal{L}, k{+}1)$-difficult with an $\mathcal{L}$-partition $P = \{G_1, \ldots, G_{k+1}\}$. We are going to show that for any $\mathcal{L}$-cover of T that is simple w.r.t. P, its size is at least $k + 2$.

Let r_i be the root of $T|_{G_i}$ and S_i be the number of edges in $T|_{G_i}$. Consider any $\mathcal{L}$-cover C of T that is simple w.r.t. P, we claim that for any edge e, at least $(c_P(e) + 1)$ $\mathcal{L}$-trails pass down e. Note that $0 \leqslant c_P(e) \leqslant k$ because we assume that the root has degree at least two. Consider any edge $e = (u, v)$. From Lemma 4.4, T_v is $(\mathcal{L}{-}d_v, c_P(e))$-difficult. Moreover, $P' = \{G_i \mid G_i \in P \text{ and } G_i \subseteq L_v\}$ is an $(\mathcal{L}{-}d_v)$-partition for T_v. As $|P'| = c_P(e) \leqslant k$, from the induction hypothesis, any $(\mathcal{L}{-}d_v)$-cover of T_v that is simple w.r.t. P' has size at least $c_P(e)+1$. It is obvious that the $\mathcal{L}$-trails in C passing down (u, v) induce an $(\mathcal{L}{-}d_v)$-cover of T_v that is simple w.r.t. P'. This implies that there must be at least $c_P(e) + 1$ $\mathcal{L}$-trails in C passing down e.

Now, we prove that the size of C is at least $k + 2$. Assume to the contrary that the size of C is less than $k + 2$. For every $\mathcal{L}$-trail $t \in C$, let $E(t)$ be the set of edges covered by t, and $v(t)$ be the node where t terminates ($v(t)$ may be a leaf or an internal node). We can see that t visits all the edges in $E(t)$ twice except for the edges on the path from the root to $v(t)$. Remember that all the $\mathcal{L}$-trails of a cover that is simple w.r.t. P stop at nodes in different $T|_{G_i}$. This implies the $v(t)$'s are located in different $T|_{G_i}$. If $v(t) \in T|_{G_i}$, then $d_{v(t)} \leqslant \text{gd}(G_i)$. Hence, the sum of the length of all the $\mathcal{L}$-trails in the simple cover C is at least twice the sum of $(c_P(e) + 1)$ of all the edges $e \in T$ minus the sum of $\text{gd}(G_i)$, $1 \leqslant i \leqslant k + 1$. Because the $T|_{G_i}$'s are edge disjoint, the sum of the number of edges of $T|_{G_i}$ is no greater than the number of edges in T. By Fact 4.3, the sum of $c_P(e)$ of all the edges in T is equal to the sum of d_{r_i}, for $1 \leqslant i \leqslant k + 1$. So the total length of C is at least $\sum_{1\leqslant i\leqslant k+1}(2d_{r_i} + 2S_i - \text{gd}(G_i)) = \sum_{1\leqslant i\leqslant k+1} \text{Len}(G_i)$. By Fact 4.2, $\text{Len}(G_i)$ is greater than $\mathcal{L}$ and the total length of C is greater than $(k+1)\mathcal{L}$. This contradicts the assumption that C is a simple $\mathcal{L}$-cover of T of size $\leqslant k+1$. Therefore, the size of C must be greater than or equal to $k+2$. The hypothesis is true for $n = k+1$ and the theorem is proved. □

Theorem 4.6 *If a tree T is $(\mathcal{L}, n)$-difficult, then any $\mathcal{L}$-cover of T has size $> \frac{2}{3}n$.*

Proof: Let $P = \{G_1, G_2, \ldots, G_n\}$ be an $\mathcal{L}$-partition for T and C is an $\mathcal{L}$-cover of T. We divide P into two sets P_1 and P_2, where

$$P_1 = \{G_i | \text{ at most one } \mathcal{L}\text{-trail in } C \text{ stops in } T|_{G_i}\}$$

$$P_2 = \{G_i | \text{ at least two } \mathcal{L}\text{-trails in } C \text{ stop in } T|_{G_i}\}$$

The size of P_2 is at most half the size of C. This implies the size of P_1 is no less than $n - \frac{1}{2}|C|$. Obviously, P_1 is an $\mathcal{L}$-partition for T and T is $(\mathcal{L}, |P_1|)$-difficult. Furthermore, by the definition of P_1, C is an $\mathcal{L}$-cover of T that is simple w.r.t. P_1. By Theorem 4.5, $|C| > |P_1| \geqslant n - \frac{1}{2}|C|$ and hence $|C| > \frac{2}{3}n$. □

4.2 Lower bound on the total weight of any $\mathcal{L}$-cover

A lower bound on the weight of any $\mathcal{L}$-cover can be derived by using the result in the previous section which has proved the size of any $\mathcal{L}$-cover of an $(\mathcal{L}, n)$-difficult tree is at least $\frac{2}{3}n + 1$. This is used to calculate the minimum number of times an edge is visited by an $\mathcal{L}$-cover of T and the sum of this number is a lower bound on the total weight.

Theorem 4.7 *Let $P=\{G_1, G_2, \ldots, G_n\}$ be an $\mathcal{L}$-partition for tree T. The total weight of any $\mathcal{L}$-cover of T is at least $\sum_{e \in T}(\frac{2}{3} c_P(e) + 1)\, w(e)$.*

Proof: Let C be any $\mathcal{L}$-cover of T. Consider any edge e in T with $c_P(e) = 0$, e is visited by at least one $\mathcal{L}$-trail in C, i.e. it is visited at least $c_P(e) + 1$ times. For edges $e = (u, v)$ with $1 \leqslant c_P(e) \leqslant n$, T_v is $(\mathcal{L}\text{–}d_v, c_P(e))$-difficult. By Theorem 4.6, any $(\mathcal{L}\text{–}d_v)$-cover of T_v has size at least $\frac{2}{3}c_P(e) + 1$. In other words, e is visited at least $\frac{2}{3}c_P(e) + 1$ times. So the total weights of $\mathcal{L}$-trails in C is at least the sum of $(\frac{2}{3}c_P(e) + 1)\, w(e)$ over all edges e in T, i.e. $\sum_{e \in T}(\frac{2}{3}c_P(e) + 1)\, w(e)$. □

5 Approximation algorithm

In this section, we describe an approximation algorithm which constructs an $\mathcal{L}$-cover of T. We also give a proof of correctness of the algorithm and show that its performance ratio (i.e. the ratio of the weight of the $\mathcal{L}$-cover found by the algorithm to the optimal one) is no greater than three.

Basically, the execution of our algorithm is divided into two phases. First, it finds an $\mathcal{L}$-partition of leaves $P = \{G_1, G_2, \ldots, G_n\}$ (for some n) for T. Then, based on this partition, it constructs an $\mathcal{L}$-cover of T.

5.1 Partition of the leaves

Our algorithm finds the G_i one by one, starting from G_1. Assume that the algorithm has found G_1, G_2, ..., G_{i-1}. Let us explain how it finds G_i. We need to define some notations first. Denoted by R the set of leaves of T not in $\bigcup_{1 \leqslant h < i} G_h$. For any node $v \in T$, let $R_v = R \cap L_v$. We say that node v is **heavy** if $\mathrm{Len}(R_v) > \mathcal{L}$, else it is **light**. Note that all the leaves of T are light. Hence, if there is a heavy node in T, there must exist a heavy node with no heavy children. To select G_i, the algorithm checks if there is any heavy nodes. If not, the execution enters the phase of constructing $\mathcal{L}$-cover (See Section 5.2.) Otherwise, it chooses a heavy node w where all its children w_1, w_2,..., w_k are light. From the definition, we have, for $1 \leqslant j \leqslant k$, $\mathrm{Len}(R_{w_j}) \leqslant \mathcal{L}$ and $\mathrm{Len}(R_w) = \mathrm{Len}(\bigcup_{1 \leqslant j \leqslant k} R_{w_j}) > \mathcal{L}$. Hence, there exists an integer m, $1 < m \leqslant k$, such that $\mathrm{Len}(\bigcup_{1 \leqslant j \leqslant m} R_{w_j}) > \mathcal{L}$ and $\mathrm{Len}(\bigcup_{1 \leqslant j < m} R_{w_j}) \leqslant \mathcal{L}$. G_i is set to be $\bigcup_{1 \leqslant j \leqslant m} R_{w_j}$. We call, for $1 \leqslant j \leqslant m$, the subtree T_{w_j} a "chosen subtree" for G_i, and for ease of referencing, we also let $H_i = R_{w_m}$. Following is an important fact about the leaves in a chosen subtree.

Fact 5.1 *If T_{w_j} is a chosen subtree for G_i, then $L_{w_j} \subseteq \bigcup_{1 \leqslant h \leqslant i} G_h$.*

Proof: If T_{w_j} is a chosen subtree for G_i, $R_{w_j} \subseteq G_i$. Moreover, since R_{w_j} is the set of leaves not in $\bigcup_{1 \leqslant h < i} G_h$, we have $(L_{w_j} - R_{w_j}) \subseteq \bigcup_{1 \leqslant h < i} G_h$. Thus, $L_{w_j} \subseteq \bigcup_{1 \leqslant h \leqslant i} G_h$. □

Procedure PARTITION
  $i = 1$
  R is the set of leaves in T
  while there are heavy nodes in T **do**
    Find a heavy node w such that all of its sons $w_1, w_2, \ldots, w_k$ are light.
    $j = 0$, $G_i = \varnothing$;
    while $\mathrm{Len}(G_i) \leqslant \mathcal{L}$ **do**
      $j = j + 1$;
      $R_{w_j} = R \cap L_{w_j}$, $R = R - R_{w_j}$;
      $G_i = G_i \cup U_{w_j}$;
    $i = i + 1$;

Let $P = \{G_1, G_2, \ldots, G_n\}$ be the final partition of leaves found by the algorithm and R be the set of leaves of T not in any G_i. The following two lemmas capture some important properties about the G_i's and R.

Lemma 5.2 *For $1 \leqslant i \leqslant n$, all the leaves in G_i can be covered by two $\mathcal{L}$-trails but cannot be covered by one $\mathcal{L}$-trail.*

Proof: From the algorithm, we have $\mathrm{Len}(G_i) > \mathcal{L}$ and both $\mathrm{Len}(G_i - H_i)$ and $\mathrm{Len}(H_i)$ are less than or equal to $\mathcal{L}$. By Fact 4.2, the lemma follows. □

Lemma 5.3 *A single $\mathcal{L}$-trail can cover all the leaves in R.*

Proof: The algorithm enters the phase of constructing $\mathcal{L}$-cover only when there is no heavy node left. This implies that the root of the tree is light, and hence, $\mathrm{Len}(R) \leqslant \mathcal{L}$. The lemma follows. □

Theorem 5.4 *$P=\{G_1, \ldots, G_n\}$ is an $\mathcal{L}$-partition for T.*

Proof: From Lemma 5.2, all the G_i's cannot be covered by one single $\mathcal{L}$-trail. Hence, to prove that P is an $\mathcal{L}$-partition, it suffices to prove for $1 \leqslant i, j \leqslant n$, $i \neq j$, $E|_{G_i} \cap E|_{G_j} = \varnothing$. We prove this by contradiction. Assume that there exist i and j such that $E|_{G_i} \cap E|_{G_j} \neq \varnothing$. Suppose that $i < j$. According to the algorithm, the set G_i is constructed before G_j. Let (u, v) be an edge in the intersection and w be the heavy node selected by the algorithm for constructing G_i. From the definition of $E|_{G_i}$, v has leaf descendants in G_i and is not a common ancestor of the leaves in G_i. As w is the least common ancestor of all the leaves in G_i, we can conclude that v is a proper descendant of w. Hence, v is in some chosen subtree for G_i. Consequently, no leaf in L_v can be in G_j (because $j > i$) and (u, v) cannot be in $E|_{G_j}$. This leads to a contradiction. □

5.2 Constructing an $\mathcal{L}$-cover

The two lemmas in Section 5.1 suggest a way to construct an $\mathcal{L}$-cover of T with size at most $2n + 1$. From the proof of Lemma 5.2, we know that each G_i can be divided into two sets H_i and $G_i - H_i$ such that both of them can be covered by a single $\mathcal{L}$-trail. To cover all the leaves in G_i, we use one $\mathcal{L}$-trail starting from the root and traverses $T|_{H_i}$ inorderly stopping at the deepest leaf of H_i; the other covering the leaves of $G_i - H_i$ in a similar way. From Lemma 5.3, we have $\mathrm{Len}(R) \leqslant \mathcal{L}$ and we use an $\mathcal{L}$-trail to traverse R inorderly. Hence, the size of the $\mathcal{L}$-cover is at most $2n + 1$. (In fact, $2n$ $\mathcal{L}$-trails are sufficient if R is empty.)

5.3 Time complexity

Given $P = \{G_1, G_2, \ldots, G_n\}$, it is easy to construct all the $T|_{G_i}$'s, as well as $T|_R$, in $O(|V|)$ time (see, e.g., [6]). Then, it takes $O(|E|) = O(|V|)$ to construct the $\mathcal{L}$-cover. To construct P, we know that there are at most $|V|$ different G_i's. To find each G_i, we need to decide, for every node v, whether v is heavy or light. We also need to calculate $\mathrm{Len}(G_i)$ in the inner **while** loop of the procedure PARTITION. All these can be done in $O(|V|)$ time if we know, for every v, the value of $\mathrm{gd}(L_v)$. All these $\mathrm{gd}(L_v)$'s can also be computed in $O(|V|)$ time. Hence, the total time complexity of our algorithm is $O(|V|^2)$. Note that by a careful implementation, we can reduce the time complexity of our algorithm to $O(|V|)$.

5.4 Performance ratios

In this section, we prove that the performance ratio of our algorithm is no greater than three. Let $P = \{G_1, G_2, \ldots, G_n\}$ be the $\mathcal{L}$-partition found by our algorithm and R be the set of remaining leaves. Let C be the $\mathcal{L}$-cover constructed by the algorithm. From the construction, it follows that, for every G_i, two $\mathcal{L}$-trails are used to cover it. Hence, every edge e along the path from the root of T to the root of $T|_{G_i}$ is traversed by two $\mathcal{L}$-trails. So e will be traversed $2\mathrm{c}_P(e)$ times by these $\mathcal{L}$-trails. For any edge in $T|_{G_i}$, it is traversed at most twice by the two $\mathcal{L}$-trails covering G_i. For the last group R, an $\mathcal{L}$-trail is used to cover it and so the edges in $T|_R$ are also traversed at most twice. As a result, the total weight of C is upper bounded by $\sum_{e \in T} 2(\mathrm{c}_P(e) + 1)\,\mathrm{w}(e)$. Since $P = \{G_1, G_2, \ldots, G_n\}$ is an $\mathcal{L}$-partition, by Theorem 4.7, the size of any $\mathcal{L}$-cover of T is greater than $\sum_{e \in T} (\frac{2}{3}\mathrm{c}_P(e) + 1)\,\mathrm{w}(e)$. It follows immediately that the performance ratio of the algorithm is less than or equal to three.

References

[1] I.S. Gopal, I. Cidon, S. Kutten. "New Models and algorithms for future networks." in *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing, 75-89, Toronto, CANADA, August 1988.*

[2] C.T. Chou and I.S. Gopal. "Linear broadcast routing." in *Journal of Algorithms, 10(4):490-517, 1989.*

[3] S. Bitan and S. Zaks. "Optimal linear broadcast." in *Journal of Algorithms, 14:288-315, March 1993.*

[4] S. Bitan and S. Zaks. "Optimal linear broadcast routing with capacity limitations." in *Proceedings of the 4th International Symposium ISAAC '93, 287-296, Hong Kong, December 1993.*

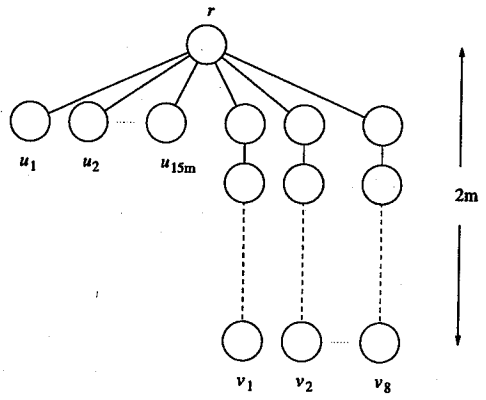
[5] H.F. Ting, W.H. Wong, M.H. Yau. "An efficient algorithm for optimal linear broadcast routing." in *Proceedings of the Fifth Italian Conference on Theoretical Computer Science, 235-249, Ravello, ITALY, November 1995.*

[6] D. Harel and R. Tarjan "Fast algorithms for finding nearest common ancestor." in *SIAM Journal on Computing, 13:338-355, 1984.*

Appendix

Below is an example showing that an $(\mathcal{L}, n)$-difficult tree can be covered by a complex $\mathcal{L}$-cover of size strictly smaller than n. Let us consider the tree T in the following figure.
There are two kinds of leaves u_i and v_i in T.

$$\begin{array}{lcll} d_{u_i} & = & 1 & 1 \leqslant i \leqslant 15m, \\ d_{v_i} & = & 2m & 1 \leqslant i \leqslant 8, \qquad \text{where } m \geqslant 8 \end{array}$$



If $\mathcal{L} = 6m - 2$, we have an $\mathcal{L}$-partition $P = \{G_1, \ldots, G_5, G'_1, \ldots, G'_4\}$ for T where

$$\begin{aligned} G_i &= \{u_j \mid 3m(i-1)+1 \leqslant j \leqslant 3mi\} \\ G'_i &= \{v_{2i-1}, v_{2i}\} \end{aligned}$$

The minimum length of trails to cover G_i and G'_i are $6m-1$ and $6m$ respectively. Note that $|P| = 9$ and the tree T is $(\mathcal{L}, 9)$-difficult. However, T can be covered by eight $\mathcal{L}$-trails as follows: we divide the set of leaves into $H_1, \ldots, H_8$ such that

$$\begin{aligned} H_i &= \{v_i\} \cup \\ &\quad \{u_j \mid (i-1)(2m-1)+1 \leqslant j \leqslant i(2m-1)\} \end{aligned}$$

Obviously each of the H_i can be covered by a single $\mathcal{L}$-trail.