Hotlink Enhancement Algorithms for Web Directories

(Extended Abstract)

Ori Gerstel[1], Shay Kutten[2], Rachel Matichin[3], and David Peleg[3]*

[1] Optical Networking Group, IBM T. J. Watson Research Center, Hawthorne, NY.
[2] Information Systems Group, Faculty of Industrial Engineering and Management, Technion, Haifa, Israel. kutten@ie.technion.ac.il
[3] Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel. {rachelm,peleg}@wisdom.weizmann.ac.il.

Abstract. Consider a web site containing a collection of web pages with data. Each page is associated with a weight representing the frequency that page is accessed by users. In the tree hierarchy representation, accessing each page requires the user to travel along the path leading to it from the root. By enhancing the index tree with additional edges (hotlinks) one may reduce the access cost of the system. That is, the hotlinks reduce the expected number of steps needed to reach a leaf page from the tree root, assuming that the user knows which hotlinks to take. The *hotlink enhancement* problem involves finding a set of hotlinks minimizing this cost. The paper proposes a hotlinks structure allowing a user with limited a-priori knowledge to determine which hotlink to use at every given point. Then, a polynomial algorithm is presented for solving the hotlink enhancement problem for such hotlinks on trees of logarithmic depth. The solution is first presented for binary trees and then extended to arbitrary degree trees. It is also shown how to generalize the solution to situations where more than one hotlink per node is allowed. The case in which the distribution on the leaves is unknown is discussed as well, and is given an algorithm guaranteeing (an optimal) logarithmic upper bound on the expected number of steps down the tree.

1 Introduction

Finding desired information in a large and diverse database is a complex task. When such a function is needed in a chaotic and large data collection such as the World Wide Web, such a function becomes even harder yet crucial. There are two basic ways to handle information finding in such a collection. One is a “flat” approach which views the information as a non-hierarchical structure and provides a query language to extract the relevant data from the database. An example of this approach on the Web is the Google search engine [1]. The other method is based on a hierarchical index to the database according to a

* Supported in part by a grant from the Israel Science Foundation.

taxonomy of categories. Examples of such indices on the Web are Yahoo [6] and the Open Directory Service [7].

An advantage of the flat approach over the hierarchical one is that the number of human operations required to find a desired piece of information is much lower (if the right query is used). As opposed to that, in the hierarchical approach it is necessary to traverse a path in the taxonomy tree from the root to the desired node in the tree. Human engineering considerations further aggravate this problem since it is very hard to choose an item from a long list (a typical convenient number is 7–10). Thus, the degree of the taxonomy tree should be rather low and the average depth of it is therefore high. Another problem of the hierarchical approach is that the depth of an item in the taxonomy tree is not based on the access pattern. See, e.g., [8]. As a result, items which have very high access frequency may require long access paths each time they are needed, while items which are "unpopular" may still be very accessible in the taxonomy tree. We would like a solution that does not change the taxonomy tree itself, since this taxonomy is likely to be meaningful and useful for the user. The solution proposed in this paper leaves the tree unchanged, but adds an auxiliary structure that helps to user reach the destination faster.

A partial solution to this problem is currently used in the Web, and consists of a list of "hot" pointers which appears in the top level of the index tree and leads directly to the most popular items. We refer to a link from a hotlist to its destination as a *hotlink*. This approach is not scalable in the sense that only a small number of items can appear in such a list.

In the current paper we study a generalization of this "hotlist", allowing us to have such lists in multiple levels in the index tree — not just the top level. The resulting structure is termed a *hotlink-enhanced index structure* (or *enhanced structure* for short). It is proposed to base the decision concerning the hotlinks to be added on the statistics of visited items in the index. The goal is to minimize the expected number of links one has to follow from the root to an item. It is therefore possible to consider static systems where the hotlinks do not get updated often, and dynamic systems which follow the access pattern and dynamically change hotlinks on the fly. In this paper we consider the former, simpler system.

Let us formally define our problem. An index system is based on a fixed tree which classifies the data items in a hierarchical manner. For the sake of simplicity we assume that data resides only in leaves of the tree [1]. To this tree, hotlinks are added and updated dynamically, based on access statistics to the data items, yielding the hotlink-enhanced index structure.

When searching for an item, the user starts from the root of the enhanced structure and advances along tree edges and hotlinks towards the required destination. The original index tree contains a unique path leading from the root of the tree to the desired leaf. An implicit assumption underlying the common hierarchical approach is that at any node along the search in the tree, the user

[1] The case where data resides in internal nodes is easily modeled by adding leaves to the tree in the simpler model.

is able to select the correct link leading towards the desired leaf. This does not necessarily mean that the user knows the tree topology, but rather that the user has some general knowledge about the domain, and the links the user finds at any node reflect some natural partitioning of the domain. Thus, when the user sees several tree edges and hotlinks at a node, the user is capable of selecting the right link downwards in the tree.

Once hotlinks are added, the situation becomes more complex, as the resulting hotlink-enhanced index structure is no longer a tree but a directed acyclic graph (DAG), with multiple alternative paths for certain destinations. Again, an underlying assumption at the basis of the hotlink idea is that when faced with a hotlink in the current page, the user will be able to tell whether or not this hotlink may lead it to a closer point on the path to the desired destination. However, the considerations that led to adding the various hotlinks to the tree are not known to the user. Thus, when the user is at some node, the user can know only the hotlinks emanating from the current node (or, in the best case, also hotlinks emanating from previous nodes the user visited on the way from the root to the current node). In particular, the user cannot know whether any hotlinks emanate from descendents of the current node, or where do they lead.

The above discussion implies that any approach taken for designing a hotlink-enhanced index structure must take into account certain assumptions regarding the user's search policy. There could be a number of different models concerning the particular choices taken by the user. At the extreme lies a natural model that is probably too strong to be used in reality. This model captures situations where the user somehow knows the topology of the enhanced structure. Henceforth we refer to this model as the "clairvoyant" user model, which is based on the following assumption.

The clairvoyant user model: At each node in the enhanced structure, the user can infer from the link labels which of the tree edges or hotlinks available at the current page is on a shortest path (in the enhanced structure) to the desired destination. The user always chooses that link.

In contrast, the model proposed here is based on the assumption that the user does not have this knowledge. This forces the user to deploy a *greedy* strategy.

The greedy user model: At each node in the enhanced structure, the user can infer from the link labels which of the tree edges or hotlinks available at the current page leads to a page that is closest *in the original tree structure* to the desired destination. The user always chooses that link.

Note that by this assumption, whenever a user is looking at the page at node v in the hotlink-enhanced structure, the user is aware of all the tree edges and hotlinks in that page, but the user's knowledge about the rest of the tree corresponds only to the logical partitioning of the domain, namely, the original tree structure. In other words, the user is not aware of other hotlinks that may exist from other pages in the tree. This means that the user's estimate for the quality of a tree edge or hotlink leading from v to some node u is based solely on the height of u in the original tree (or equivalently the distance from u to the desired destination in the tree). An important implication is that following the greedy strategy does not necessarily lead to an optimal path in the hotlink-enhanced index structure.

This paper addresses the optimization problem faced by the index designer, namely, to find a set of hotlinks that minimizes the expected number of links (either tree edges or hotlinks) traversed by a greedy user from the root to a leaf. More formally, given a tree T, representing an index, a *hotlink* is an edge that does not belong to the tree. The hotlink *starts* at some node v and *ends* at (or *leads* to) some node u that is a descendant of v. (One may possibly consider a different model which allows to have hotlinks from a node v to a non-descendant node u residing in another subtree. In our model, however, such hotlinks will never be used, due to the greedy assumption.) We assume, without loss of generality, that u is not a child of v. Each leaf x of T has a weight $p(x)$, representing the proportion of the user visits to that leaf, compared with the total set of user's visits. Hence if normalized, $p(x)$ can be interpreted as the probability that a user wants to access leaf x. Another parameter of the problem is an integer K, specifying an upper bound on the number of hotlinks that may start at any given node. (There is no a-priori limit on the number of hotlinks that lead to a given node).

Let S be a set of hotlinks constructed on the tree (obeying the bound of K outgoing hotlinks per node) and let $D_S(v)$ denote the greedy path (including hotlinks) from the root to node v. The expected number of operations needed to get to an item is $f(T,p,S) = \sum_{v\in\mathcal{L}(T)} |D_S(v)| \cdot p(v)$. The problem of optimizing this parameter is referred to as the *hotlink enhancement* problem. Two different *static* problems arise, according to whether the probability distribution p is known to us in advance or not. Assuming a *known distribution*, our goal is to find a set of hotlinks S which minimizes $f(T,p,S)$ and achieves the optimal cost $\hat{f}(T,p) = \min_S\{f(T,p,S)\}$. Such a set is termed an *optimal* set of hotlinks. On the other hand, under the *unknown distribution* assumption, the worst-case expected access cost on a tree T with a set of hotlinks S is $\tilde{f}(T,S) = \max_p\{f(T,p,S)\}$, and our goal is to find a set of hotlinks S minimizing $f(T,S)$ and achieving the optimal cost $\tilde{f}(T) = \min_S\{\tilde{f}(T,S)\}$.

For the latter problem, there exists an equivalent formulation, independent of the probability distributions, based on the observation that for every tree T and hotlink function S, $\tilde{f}(T,S) = \max_{v\in\mathcal{L}(T)}\{|D_S(v)|\}$, and therefore:

Lemma 1. *For every tree T, $\tilde{f}(T) = \min_S\{\max_{v\in\mathcal{L}(T)}\{|D_S(v)|\}\}$.*

Note that the above definitions can be repeated for the clairvoyant user model.

The clairvoyant user model, not used in the current paper, was discussed in previous papers. A proof of NP-hardness for adding hotlinks on DAGS was presented in [2] by a reduction from the problem of Exact Cover by 3-Sets (which is known to be NP-Complete) to that of hotlink enhancement for DAGs. An interesting analogy was presented for the clairvoyant user model between the problem of adding hotlinks and coding theory. One can think of the index tree as the coding of words (where in a binary tree, for example, a left move corresponds to '0' and a right move corresponds to '1'). Thus any leaf is a codeword in the code-alphabet. By adding a hotlink we actually add another letter to the alphabet. Consequently, Shannon's theorem suggests a lower bound for the problem. In particular, in binary trees, denoting by $H(p)$ the entropy of the access distribution on the leaves, and denoting by T^A the hotlink-enhanced index structure resulting from the original tree T with the hotlinks added by the algorithm A, $E[T^A, p] \geq H(p)/\log 3 = \frac{1}{\log 3} \sum_{i=1}^{N} p_i \log(1/p_i)$, and in trees of maximal degree Δ, $E[T^A, p] \geq H(p)/\log \Delta$. An approximation algorithm for adding hotlink to bounded degree trees for a clairvoyant user is presented in [3]. It turns out that this algorithm can also be used under our greedy user model. The approximation ratio of the algorithm depends on Δ, the maximum degree of the tree, and on the entropy of the access distribution (and is in general at least $\log(\Delta + 1)$). Recently, a polynomial time algorithm for approximating the hotlink assignment problem in the clairvoyant model was presented in [5]. This algorithm uses greedy choices at each iteration, and achieves an approximation ratio of 2. Another recent article [4] discusses the use of hotlink assignments in asymmetric communication protocols to achieve better performance bounds.

In this paper we present an algorithm for optimally solving the hotlink enhancement problem on trees in the greedy user model. The algorithm uses dynamic programming and the greedy assumption to limit the search operations. We also show how to generalize the solution to arbitrary degree trees and to hotlink enhancement schemes that allow up to K hotlinks per node. In contrast with the approximation algorithm of [3], which is polynomial for trees of bounded degree but arbitrary depth, our (exact) algorithm can be used for trees with unbounded degree, but its time complexity is polynomial only on trees of logarithmic depth.

Finally, we give an algorithm for handling settings when the distribution on the leafs is unknown. This algorithm provides a logarithmic bound on the expected number of steps to reach the desired leaf. Deriving a lower bound on the expected tour length of optimal solution, we then conclude that our algorithm ensures constant ratio approximation.

In the full paper we show that the NP-hardness proof of [2] for the hotlink enhancement problem on DAGs in the clairvoyant user model can be easily augmented to prove also the NP-hardness of the problem in the greedy model.

In what follows, Section 2 presents our algorithm for finding an optimal set of hotlinks and its analysis, and Section 3 discusses the particular case where the frequencies of visiting each page are unknown.

2 Known Distribution Model

Our algorithm makes use of the following properties of the problem (whose proofs are deferred to the full paper).

Lemma 2. *There exists an optimal solution to the hotlink enhancement problem in which no two hotlinks arrive at the same node.*

It is convenient to consider a hotlink from node u to node v as a pair of parentheses, where the start-node u of the hotlink marks the left parenthesis and the end-node v marks the right parenthesis. Using this representation, the hotlinks placed along any path from the root over the tree form a sequence of parentheses.

We say that a set of hotlinks S is *well-formed* if on any path from the root over the tree, the parentheses sequence of S on this path is well-formed, namely, hotlinks do not cross each other. Observe that for a well-formed set of hotlinks S, the greedy path from the root to any leaf v coincides with the shortest path in the hotlink-enhanced index structure. This means that the costs of a well-formed S in the greedy user model and in the clairvoyant user model are the same.

Lemma 3. *For every index tree T, there exists an optimal solution to the hotlink enhancement problem in the greedy user model which is well-formed.*

The usefulness of this lemma stems from the fact that it helps to narrow down the domain of potential solutions that needs to be searched (in the greedy user model) in order to find the optimal one.

We first restrict our discussion to n-node binary trees of depth $O(\log n)$ and to the case $K = 1$. We represent a solution S as a function $S : V \mapsto V$, with $S(v)$ being the endpoint of the hotlink starting at v.

For every node v in the tree T, let T_v denote the subtree of T consisting of v and all its descendants, and let $\mathcal{P}_v$ denote the path leading from the root to v in T. We generalize the definition of S to a function $S : V \mapsto 2^V$. For a node v, the set $S(v)$ is referred to as an *undetermined hotlink*, and interpreted as the collection of candidates to be the final endpoint of the hotlink from v. An (undetermined) hotlink function S is said to be *determined* or *fixed for* a node set $W \subseteq V$ if $S(w)$ is a singleton for every $w \in W$.

The cost associated with the solution S on the subtree T_v, assuming S is determined for $\mathcal{P}_v \cup T_v$, is $c(v, S) = \sum_{w \in \mathcal{L}(T_v)} |D_S(w)| \cdot p(w)$. The cost of S over the entire tree T with root r is thus $f(T, p, S) = c(r, S)$.

An (undetermined) hotlink function S is said to be *separated w.r.t.* the node v (or simply *v-separated*) if the nodes of $\mathcal{P}_v$ are partitioned into three disjoint sets, $\mathcal{P}_v = \mathcal{P}_v^F(S) \cup \mathcal{P}_v^I(S) \cup \mathcal{P}_v^O(S)$, called the *fixed-set*, *in-set* and *out-set*, such that (1) for every $w \in \mathcal{P}_v^F(S)$, $S(w)$ is a single descendant of w in $\mathcal{P}_v$, (2) for every $w \in \mathcal{P}_v^I(S)$, $S(w) \subseteq T_v$, and (3) for every $w \in \mathcal{P}_v^O(S)$, $S(w) \subseteq V \setminus (\mathcal{P}_v \cup T_v)$.

We remark that our algorithm will consider candidate solutions separated w.r.t. v in which for every node w in the in-set we have equality, i.e., $S(w) = T_v$, and similarly, for every node w in the out-set we have $S(w) = V \setminus (\mathcal{P}_v \cup T_v)$.

For two solutions S_1 and S_2 and a node set W, we say that S_1 *is compatible with S_2 on W* if $S_1(v) \subseteq S_2(v)$ for every node $v \in W$. S_1 is compatible with S_2 if it is compatible with S_2 on the entire node set of T.

Note that for an undetermined hotlink function S, the greedy path $D_S(w)$ is not necessarily defined for every leaf w of T. For the path to be uniquely defined as $D_S(w) = \langle root = v_0, v_1, \ldots, v_q = w \rangle$, it is required that the sequence of nodes $v_0, \ldots, v_q$ satisfies $S(v_i) = \{v_{i+1}\}$ for $i = 1, 2, \ldots, q-1$. In this case We say that S is *route-determined* for w. This means, in particular, that the cost of accessing w is determined.

Lemma 4. *(a) If a hotlink function S is determined for $\mathcal{P}_v \cup T_v$, then it is route-determined for every leaf w in T_v. (b) If a v-separated hotlink function S is determined for $\mathcal{P}_v^I \cup T_v$, then it is route-determined for every leaf w in T_v.*

Lemma 4 facilitates the use of dynamic programming on the problem, as it leads to the observation that the cost of any solution S over the subtree T_v is determined solely on the basis of its values on $\mathcal{P}_v \cup T_v$, and more importantly, the cost of any v-separated solution S over the subtree T_v is determined solely on the basis of its values on $\mathcal{P}_v^F \cup \mathcal{P}_v^I \cup T_v$. More formally, we have:

Lemma 5. *Consider two hotlink functions S_1 and S_2 that are both v-separated with the same fixed-set, $\mathcal{P}_v^F(S_1) = \mathcal{P}_v^F(S_2)$, and the same in-set, $\mathcal{P}_v^I(S_1) = \mathcal{P}_v^I(S_2)$. If S_1 and S_2 are determined in the same way on $\mathcal{P}_v^F \cup \mathcal{P}_v^I \cup T_v$, i.e., $S_1(w) = S_2(w)$ for every $w \in \mathcal{P}_v^F \cup \mathcal{P}_v^I \cup T_v$, then $c(v, S_1) = c(v, S_2)$.*

The recursive procedure Proc employed by the algorithm receives as its input a node v in the tree, and a v-separated partial hotlink function S of a specific form. Suppose v is of depth d from the root, and let $\mathcal{P}_v = (root = v_0, v_1, \ldots, v_d = v)$. Then S will be specified as a vector $\bar{s} = \langle s(0), \ldots, s(d-1) \rangle$, where

$$s(i) = \begin{cases} j, & S(v_i) = v_j \text{ for } i+2 \le j \le d, \\ I, & S(v_i) = T_v, \\ O, & S(v_i) = V \setminus (\mathcal{P}_v \cup T_v). \end{cases}$$

The goal of Procedure $\mathsf{Proc}(v, \bar{s})$ is to calculate the optimal completion of S on $\mathcal{P}_v^I(S) \cup T_v$, and its cost.

Procedure Proc operates as follows. If T_v is small enough (e.g., it contains fewer than K_0 nodes, for some constant K_0), then the best determination and its cost are found by exhaustive search, examining all possible completions.

Now suppose the tree T_v is larger than K_0 nodes. Denote the children of v by v_L and v_R. We aim to generate all possible v_L and v_R separated partial hotlinks that are compatible with S. The procedure goes over all possible ways of partitioning the set $\mathcal{P}_v^I$ (including v itself) into four disjoint sets named H_L, H_R, B_L and B_R. The set H_L will be interpreted as the set of nodes that have a hotlink directed to v_L (in any solution it is enough to have only one such hotlink). If $H_L = \emptyset$, then in the current completion there is no hotlink to be ended directly in node v_L. The set B_L will be interpreted as the collection of start points of hotlinks to be ended at the nodes of T_{v_L} except v_L itself (the left side sub tree). The sets B_R and H_R are defined analogously for the right hand side of the tree T_v. Thus we get all v_R separated and v_L separated hotlink functions possible from S.

For each such partition (H_L, H_R, B_L, B_R) constructed from S, the procedure does the following. It first generates the (partial) solution S_L derived from S by specifying that the hotlink from the node of H_L (if exists) ends at v_L, the hotlinks from the nodes of B_L (if exist) end inside T_{v_L}, and the hotlinks from the nodes of $H_R \cup B_R$ end outside $\mathcal{P}_{v_L} \cup T_{v_L}$. I.e., the vector representation of the generated S_L is

$$s_L(i) = \begin{cases} d+1, & i \in H_L, \\ O, & i \in B_R \cup H_R, \\ s(i), & \text{otherwise.} \end{cases}$$

(Note that in particular, $s_L(i)$ remains I for nodes v_i of B_L, and maintains its previous value (which is either O or some $i+2 \leq j \leq d$) for nodes outside $\mathcal{P}_v^I$.

The procedure similarly generates the partial solution S_R derived from S by following the specifications of the partition for the right subtree, T_{v_R}.

Then, the procedure is invoked recursively on $(v_L, \bar{s}_L)$ and $(v_R, \bar{s}_R)$, and returns cost values x_L and x_R respectively (accompanied with the determinations yielding them).

Of all the partitions examined, the procedure then selects the one yielding the lowest combined cost $x_L + x_R$ (along with the determination yielding it).

Note that the algorithm invokes the procedure by dynamic programming, rather than plain recursion. Namely, it maintains a table of determinations and costs $A(v, \bar{s})$, for every node v and partial solution $\bar{s}$ of the type described above. Whenever the procedure requires an answer for some pair $(v, \bar{s})$, the procedure first consults the table A. Hence each entry in the table must be computed only once, on the first occasion it is requested. (For example, B_L at v may be the same when in two different computations that differ in B_R.)

Lemma 6. *For every node v in T, and for every v-separated hotlink function S, procedure* Proc *returns a determination for $(v, \bar{s})$ of the minimal cost $c^*(v, S)$.*

To find the optimal cost, $\min_S\{f(T, p, S)\}$, we need to run $\mathsf{Proc}(r, \bar{0})$ where $\bar{0}$ is an empty vector and r is the root of the tree (there are no nodes in $\mathcal{P}_r$).

Let us first estimate the number of entries in table A. There are n possible values for v. For each v, we need to bound N_v, the number of legal vectors $\bar{s}$ that need to be considered. Note that the length of $\bar{s}$ is bounded by $Depth(T) = O(\log n)$. Observe also that for every $0 \leq i \leq d-2$, where v is at depth d, there are at most $d-i$ possible values for $s(i)$ (namely, any $i+2 \leq j \leq d$, plus the values I and O). Each $s(i)$ may assume up to $O(\log n)$ different values, and hence it seems as though there might be up to $(c \log n)^{c \log n}$ different $\bar{s}$ configurations overall, for constant c, which is superpolynomial. Fortunately, the number of legal solutions is restricted by the parenthesis requirement of Lemma 3. In particular, since the number of legal sequences of m parenthesis pairs is the m'th Katalan number, $K_m = \frac{1}{m+1}\binom{2m}{m}$, and there are $\binom{d+1+2m-1}{2m}$ ways to choose placements for those parentheses around d nodes taken $2m$ at a time with repetition allowed, the number of legal choices of m pairs of parentheses over d letters is bounded by $K_m \cdot \binom{d+2m}{2m}$. It can be readily verified that N_v, the number of entries corresponding to v, is bounded above by

$$N_v \leq \sum_{m=1}^{d} 2^{d-m} \frac{1}{m+1}\binom{2m}{m} \cdot \binom{d+2m}{2m} \leq 2^d \sum_{m=1}^{d} \binom{2m}{m} \cdot \binom{d+2m}{2m}$$
$$\leq 2^d \cdot d \cdot \binom{2d}{d} \cdot \binom{3d}{2d} \leq d \cdot 2^{6d} = O(\log n) \cdot 2^{O(\log n)} = O\left(n^{O(1)}\right) .$$

Hence the number of invocations of the procedure is polynomial in n. Each invocation requires us to go over all the possible partitions of a set of size at most $O(\log n)$ into four subsets (two of which are at most singletons), hence its complexity is polynomial in n. Consequently, the entire algorithm is polynomial as well.

Theorem 1. *There exists a polynomial time algorithm for solving the hotlink enhancement problem with known probability distribution on n-leaf binary trees of depth $O(\log n)$.*

The solution as described above applies only to binary trees. In the full paper we outline the way to generalize it to trees of arbitrary degree. We also consider the case $K > 1$, i.e., where more than one outgoing hotlink is allowed at each node. Finally, in a dynamically changing environment, it may happen that after enhancing the tree once with new hotlinks we wish to repeat the process again and add additional hotlinks. In the full paper we show that our algorithm can be extended to handle this setting as well, by handling tree-dags, namely, DAGs created from a tree by adding only edges from a node to one of its descendants. (Note that the DAG generated by applying our algorithm to a tree is indeed a tree-DAG.)

Corollary 1. *There exists a polynomial time algorithm for solving the hotlink enhancement problem with known probability distribution on arbitrary degree n-leaf tree-DAGs of depth $O(\log n)$ where every node can have up to $K = O(1)$ outgoing hotlinks.*

3 Unknown Distribution

In this section we consider the case that the probabilities associated with each leaf are not known to us. In this case, constructing the best set of hotlinks for the instance at hand is out of the question. Nonetheless, it is possible to construct a generic set of hotlinks that will guarantee a global upper bound of $O(\log n)$ on the access cost. In fact, this can be achieved using a single hotlink per node.

We rely on the well-known fact that that given an n-node tree T, it is always possible to find a separating node v, whose removal breaks the tree into subtrees of size at most $n/2$. Using this fact, we define the hotlinks for our tree T rooted at r as follows. Let w_L and w_R be r's children in T. First, find a separator node v for T as in the lemma, and add a hotlink from r to v. Next, generate hotlinks for T_v, T_{w_L} and T_{w_R} by recursively applying the same procedure.

Letting $f(n)$ denote the maximum root-to-leaf distance in an n-node tree with hotlinks generated by the above procedure, we have to prove that $f(n) \leq c \log n$ for some constant c. This is established by noting that the construction guarantees that $f(n) \leq 1 + f(n/2)$. It follows that no matter what the probabilities are, the resulting cost using this construction is always $O(\log n)$, namely, for any n-node tree T, the algorithm described above constructs a hotlink function S such that $\tilde{f}(T, S) = O(\log n)$. Thus we have:

Theorem 2. *The hotlink enhancement problem with (known or unknown) probabilities has a polynomial time approximation algorithm with ratio $O(\log n)$.*

We now prove a lower bound on the expected access cost under an unknown probability distribution on the leaves. We assume that the tree T is Δ-ry, i.e., its degree is at most Δ, for some constant $\Delta \geq 1$, and that K hotlinks are allowed from each node.

Observe that if there is a hotlink leading to some node w in T, then the tree edge leading to w from its parent in T will never be used in a greedy route to any leaf of T_w. This observation implies that for any set of hotlinks S, the solution resulting from adding S to T is equivalent in cost to some $(\Delta + K)$-ry tree T' with *no hotlinks* at all. The tree T' can be obtained from the pair (T, S) by performing the following modification, for each hotlink leading from v to w: Eliminate the edge connecting w to its parent from the tree, and replace the hotlink by a tree edge connecting v to w. Hence the cost of any solution using up to K hotlinks on a Δ-ry tree is bounded from below by the cost of the best solution using no hotlinks on a $(\Delta + K)$-ry tree.

For an integer $\ell \geq 1$, the maximum number of distinct leaves reachable from the root in ℓ steps on a $(\Delta + K)$-ry tree is bounded by $(\Delta + K)^\ell$. This implies that a solution S in which each of the n nodes is reachable by a path of length ℓ or less, i.e., with $D_S(v) \leq \ell$, must satisfy $(\Delta + K)^\ell \geq n$, or, $\ell \geq \frac{\log n}{\log(\Delta+K)}$. Hence for constant Δ and K, $\ell = \Omega(\log n)$. Using Lemma 1 we get

Theorem 3. *(a) For any n-leaf Δ-ry tree T, if at most K hotlinks are allowed from each node, then $\tilde{f}(T) = \frac{\log n}{\log(\Delta+K)} - 1$. In particular, for constant Δ and K, $\tilde{f}(T) = \Omega(\log n)$. (b) For bounded degree trees, the hotlink enhancement problem with unknown probabilities and constant K has a polynomial time constant ratio approximation algorithm.*

References

1. `http://www.google.com/`.
2. Bose, P., Czywizowicz, J., Gasieniec, L., Kranakis, E., Krizanc, D., Pelc, A., and Martin, M. V., Strategies for hotlink assignments. *Proc. 11th Symp. on algorithms and computation (ISAAC 2000)*, pp. 23–34.
3. Kranakis, E., Krizanc, D., and Shende, S., Approximating hotlink assignments, *Proc. 12th Symp. on algorithms and computation (ISSAC 2001)*, pp. 756–767.
4. Bose, P., Krizanc, D., Langerman, S. and Morin, P., Asymmetric communication protocols via hotlink assignments, *Proc. 9th Colloq. on Structural Information and Communication Complexity*, June 2002, pp. 33–39.
5. Matichin, R., and Peleg, D., Approximation Algorithm for Hotlink Assignments in Web Directories, *8th Workshop on Algorithms and Data Structures*, Ottawa, Canada, Aug. 2003.
6. `http://www.yahoo.com/`.
7. www.dmoz.org
8. Attardi G., Di Marco S., Salvi D, Categorization by Context, *Journal of Universal Computer Science*, (1998), Springer Verlag. 4:9:719–736.