

Near-Entropy Hotlink Assignments

Karim Douïeb * § Stefan Langerman † §

Abstract

Consider a rooted tree T of arbitrary maximum degree d representing a collection of n web pages connected via a set of links, all reachable from a source home page represented by the root of T . Each web page i carries a probability p_i representative of the frequency with which it is visited. By adding hotlinks — shortcuts from a node to one of its descendants — we wish to minimize the expected number of steps l needed to visit pages from the home page, expressed as a function of the entropy $H(p)$ of the access probabilities p . This paper introduces several new strategies for effectively assigning hotlinks in a tree. For assigning exactly one hotlink per node, our method guarantees an upper bound on l of $1.141H(p) + 1$ if $d > 2$ and $1.08H(p) + 2/3$ if $d = 2$. We also present the first efficient general methods for assigning at most k hotlinks per node in trees of arbitrary maximum degree, achieving bounds on l of at most $\frac{2H(p)}{\log(k+1)}$ and $\frac{H(p)}{\log(k+d) - \log d}$, respectively. All our methods are *strong*, i.e. they provide the same guarantees on all subtrees after the assignment. We also present an algorithm implementing these methods in $O(n \log n)$ time, an improvement over the previous $O(n^2)$ time algorithms. Finally we prove a $\Omega(n \log n)$ lower bound on the running time of any strong method which guarantee an average access time strictly better than $2H(p)$.

1 Introduction

Netcraft [1] reported recently that the number of web sites in the *World Wide Web* has grown steadily, from approximately 23000 in 1995 to over 100 million by the ending of 2006. Finding information in such a large database is becoming a complex task.

There are many ways to speed up the access to information on the Web. The solution discussed in this paper doesn't change the original hyperlink structure but enhances it with additional hyperlinks in order to speed up the access to a destination. This addition of hyperlinks is called a *hotlink*

§Département d'Informatique, Université Libre de Bruxelles, CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium.

*Boursier FRIA, kdouieb@ulb.ac.be

†Chercheur qualifié du FNRS, stefan.langerman@ulb.ac.be

assignment. The *hotlink assignment* problem was originally introduced by Perkowitz and Etzioni [15] to improve the search in Web sites.

Hotlinks are defined as additional pointers to a structure with the goal of improving its design by reducing the expected number of steps to reach an element. A hotlink can be seen as a shortcut from a web page to another one that is accessible from it (see Fig. 1.b).

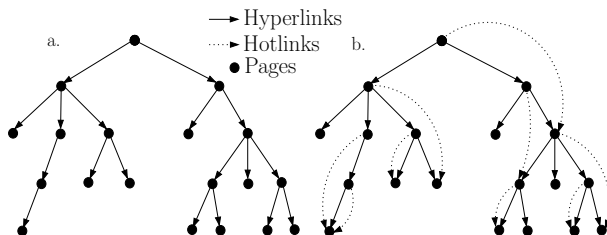


Figure 1: a. Modeled site web, b. Example of hotlink assignment.

Hotlinks can help solving many other optimization problems. For example, hotlinks were used with success for designing asymmetric communication protocols [4]. Other possibilities of applications exist such as optimization of information architecture, static or dynamic: knowledge management and document management platform, web index and more generally any large library index systems or arborescent directories.

Other different methods have been developed to facilitate and accelerate the search on the web, such as promoting and demoting pages, highlighting links, and clustering related pages in an adaptive fashion depending on user access patterns [15, 8]. These are not considered in this paper.

The problem: Formally, a *web site* can be modeled as a directed graph $\mathcal{G} = (V, E)$ where the nodes V correspond to the web pages and the edges E represent the links. Each node carries a weight representative of its access frequency. We assume that all web pages are reached starting from the *homepage* r . Our goal in adding hotlinks (one or up to k directed edges from a node to one accessible from it) is to minimize the expected number of steps to reach a page from the homepage r .

We restrict our attention to the case when \mathcal{G} is a rooted directed tree T with n nodes and maximum degree d (maximum number of children for a node). The stated results extend to general graphs by taking T to be the shortest-path tree of \mathcal{G} from the homepage r . Every leaf i in T is associated with a weight w_i representative of its access frequency, and $W = \sum_{i \in T} w_i$.

We thus assume that only the leaves of the tree are accessed. This restriction can easily be removed by adding a leaf child to all nodes, with a weight corresponding to the access frequency of the node. This transformation only increases the length of the search paths by 1. We use T_x to denote the subtree rooted at x and $W(T_x)$ to denote its weight, i.e. the sum of the weights of its leaves.

Following the *greedy user* model assumption [11], we assume that from a node the user always takes the pointer that leads him as close as possible to the desired destination. Due to that assumption, the assignment of one hotlink which points to a node i can be seen as the deletion of the other hyperlink that ends in i (i.e. an adoption) because if the user doesn't follow the hotlink then he will not access this subtree (see Fig. 2).

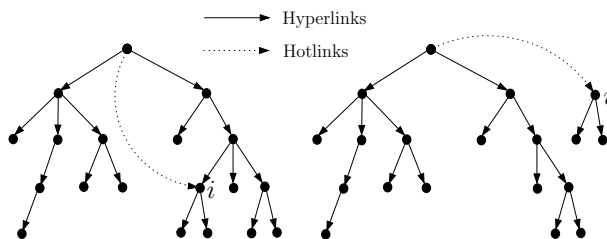


Figure 2: Consequence due to the *greedy user* model assumption.

Let T^A be the tree resulting from an assignment A of hotlinks. A measure of the average access time to the nodes is

$$E[T^A, p] = \sum_{i=1}^n d_A(i) p_i,$$

where $d_A(i)$ is the distance of the node i from the root, and $p = \langle p_i = w_i/W : i = 1, \dots, n \rangle$ is the probability distribution on the nodes of the original tree T . We are interested in finding an assignment A which minimizes $E[T^A, p]$.

A lower bound on the average access time $E[T^A, p]$ was given in [3] using information theory [14]. Let $H(p)$ be the entropy of the probability distribution p , defined by

$$H(p) = \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right),$$

then for any assignment of at most k hotlinks per node the expected number of steps to reach a node from the root of a tree of maximum degree d is at

least $H(p)/\log(d+k)$ in the best case. The tree could be a list, in which case we have a lower bound of $H(p)/\log(1+k)$.

We focus on recursive algorithms which first choose the hotlink(s) of the root of the tree T , perform the adoption (see Fig. 2), and recursively assign hotlinks to the children of the root (including the hotlink). We characterize these algorithms as *top-down* if the hotlink assignment of a subtree only depends on the subtree itself minus the subtrees adopted by its own ancestors.

Related work: The idea of hotlinks was suggested by Perkowski and Etzioni [15] to improve the search in Web sites (seen as DAGs). Later Bose et al. [3] proved that finding the optimal hotlink assignment for a DAG is NP-hard, and analyzed several heuristics for assigning hotlinks.

The problem might become easier when the graph considered is a rooted tree. Kranakis, Krizanc and Shende [13] give a $O(n^2)$ time algorithm for assigning one hotlink per node so that the expected number of steps to search a node from the root of the tree attains the entropy bound within a constant factor. Several results on adding hotlinks to nodes of d -regular complete trees are also reported by Fuhrmann et al.[10]. Recently Gerstel et al.[11] and Pessoa et al.[16] independently discovered a polynomial time dynamic programming algorithm for finding the optimal placement of hotlinks on a tree whose depth is logarithmic in the number of nodes, the running time of the algorithm of Gerstel et al. is $O(n3^D)$ where D is the height of the tree. Experimental results showing the validity of the hotlinks approach was presented by Czyzowicz et al.[6], and a software tool to structure web-sites efficiently by automatic assignment of hotlinks has been developed by Kranakis et al.[12].

The concept of hotlinks can be applied to other problems than that of web structuring. For instance, Bose et al.[4] use hotlink assignments to design efficient asymmetric communication protocols. Hotlinks can also be used to design data structures as was demonstrated by Brönnimann, Cazals and Durand [5] with their *jumplist* dynamic dictionary data structure. The jumplist structure can be seen as randomized hotlink assignment on a list, and is meant as a simplification of the skiplist structure [17]. A deterministic version of the randomized jumplist was developed by Elmasry [9] and by Douïeb and Langerman [7], independently.

Using this deterministic jumplist, we recently introduced a linear time algorithm [7] to allow the assignment of one hotlink per node in such a way that the expected number of steps to reach a node i from the root of a tree is bounded by the entropy, namely by $3H(p)$. The method was then dynamized to maintain hotlinks when nodes are added, deleted or their weights modified, in amortized time $O(\log W/w_i)$ per update.

Our results: Known exact algorithms [11, 16] for finding the optimal assignment of hotlinks have a polynomial running time only for trees of logarithmic depth, and are slow, so our work was focused on finding an assignment approaching the entropy bound. The best previous algorithm, the KKS method [13], guarantees that the average access time to the elements is at most $\frac{H(p)}{\log(d+1) - (d/(d+1)) \log d} + \frac{d+1}{d}$, its asymptotic behavior is $H(p) \frac{d}{\log d}$ for sufficiently large values of d (maximum degree of the tree). This upper bound is shown to be tight in section 3.1.

Two new top-down methods for assigning one hotlink per node are then presented: The MinMax method, an intuitive variant of the previous method is presented in section 3.2 and shown not to improve significantly over KKS. In section 3.3, the h/p_h method guarantees an average access time of at most $1.141H(p) + 1$. This near-entropy bound, in contrast to that of KKS or MinMax, is completely independent of the maximum degree of the tree and is better than KKS for all values of $d > 2$. Furthermore, h/p_h method matches the bound of KKS for $d = 2$.

In Section 4, we present a natural generalization of the algorithm of Bose *et al.* [4] for assigning k hotlinks per node in trees of arbitrary maximum degree d instead of binary trees, it guarantees an upper bound on the average access time of $\frac{H(p)}{\log(k+d) - \log d}$. As the performance guarantee of this method degrades when d grows, we show a second method whose average access cost is at most $\frac{2H(p)}{\log(k+1)}$ constituting the first multiple hotlink assignment method giving a near-entropy bound that is independent of the degree.

All our methods are strong in the sense that they guarantee the same average access time for each subtree of the tree after assignment.

In the Section 5 we develop a fast algorithm for the methods seen in the preceding sections; it uses an enhanced version of the link-cut trees of Sleator and Tarjan [18] and performs the hotlink assignment in $O(n \log n)$ time for all our methods and the KKS method [13]. This is an improvement over the previous $O(n^2)$ algorithms.

Finally in Section 6 we give a $\Omega(n \log n)$ lower bound on running time of any strong near-entropy hotlink assignment methods which guarantee an average access time of $\alpha H(p)$ with $\alpha < 2$.

2 Top-Down Methods

Before giving some hotlinks assignment methods and their analysis we present a useful Lemma concerning entropy. Consider a probability dis-

tribution $p = \langle p_1, p_2, \dots, p_n \rangle$ and a partition A_1, A_2, \dots, A_k of the index set $\{1, 2, \dots, n\}$ into k non-empty subsets. Define $S_i = \sum_{j \in A_i} p_j$ for $i = 1, 2, \dots, k$. Consider the new distributions: $p^{(i)} = \langle p_j^{(i)} := \frac{p_j}{S_i} : j \in A_i \rangle$ for $i = 1, 2, \dots, k$. Kranakis, Krizanc and Shende [13] proved the following lemma:

Lemma 1 *For any partition A_1, A_2, \dots, A_k of the index set of the probability distribution we have the identity*

$$H(p) = \sum_{i=1}^k S_i H(p^{(i)}) - \sum_{i=1}^k S_i \log S_i,$$

where S_i and $p^{(i)}$ are defined in the above equations.

A hotlink method \mathcal{A} determines the hotlink assignment $A = \mathcal{A}(T)$ to be applied on any tree T . Let T^A be the tree T enhanced by the hotlink assignment A and $T^{\mathcal{A}} = T^{\mathcal{A}(T)}$. Consider that a selection of successive hotlinks starting from the root node partitions the leaves of the tree $T^{\mathcal{A}}$ into several subsets or subtrees $T_1^{\mathcal{A}}, T_2^{\mathcal{A}}, \dots, T_k^{\mathcal{A}}$ with corresponding weights S_1, S_2, \dots, S_k . These subtrees have a depth in the tree corresponding to the number of pointers that we must follow to reach them, called $d(T_i^{\mathcal{A}})$.

We defined a *top-down* hotlink assignment method \mathcal{A} to be a method beginning by the assignment of the hotlink of the root of a tree and where the hotlink assignment of any subtree T_i only depends on the subtree itself minus the subtrees adopted by its own ancestors.

Lemma 2 *Given a top-down hotlink assignment method \mathcal{A} , if we can fix a constant a such that for all tree T there exists a partition of the leaves in the subtrees $T_1^{\mathcal{A}}, T_2^{\mathcal{A}}, \dots, T_k^{\mathcal{A}}$ of weights S_1, S_2, \dots, S_k which satisfies*

$$a \geq - \frac{\sum_{i=1}^k S_i d(T_i^{\mathcal{A}})}{\sum_{i=1}^k S_i \log S_i},$$

then the expected number of steps needed to reach a leaf from the root of a tree $T^{\mathcal{A}}$ is $E[T^{\mathcal{A}}, p] \leq aH(p) + 1$.

Proof: By induction on the depth of the tree. If the depth is equal to 1 then the lemma is true for any positive value of a , the average access time is always 1. Assume the induction hypothesis is valid for the subtrees $T_i^{\mathcal{A}}$ for all i . Now we calculate the access time for the tree $T^{\mathcal{A}}$. We obtain using

the Lemma 1 (where A_i in Lemma 1 corresponds to the set of leaves in the subtree T_i^A),

$$\begin{aligned}
E[T^A, p] &= \sum_{i=1}^k S_i(d(T_i^A) + E[T_i^A, p^{(i)}]) \\
&= \sum_{i=1}^k S_i d(T_i^A) + \sum_{i=1}^k S_i E[T_i^A, p^{(i)}] \\
&\leq \sum_{i=1}^k S_i d(T_i^A) + \sum_{i=1}^k S_i (aH(p^{(i)}) + 1) \\
&= \sum_{i=1}^k S_i d(T_i^A) + aH(p) + a \sum_{i=1}^k S_i \log S_i + \sum_{i=1}^k S_i \\
&\leq aH(p) + 1
\end{aligned}$$

The last inequality is valid if we choose the constant a such that

$$\begin{aligned}
\sum_{i=1}^k S_i d(T_i^A) + a \sum_{i=1}^k S_i \log S_i &\leq 0 \\
-\frac{\sum_{i=1}^k S_i d(T_i^A)}{\sum_{i=1}^k S_i \log S_i} &\leq a.
\end{aligned}$$

□

Finally we generalize Lemma 5 of [13]:

Lemma 3 *For any fixed constant $0 \leq \alpha \leq 1/2$, the solutions of the optimization problem maximize $f(s_1, s_2, \dots, s_k) = \sum_{i=1}^k s_i \log s_i$ subject to $\{0 \leq s_i \forall i, \sum_{i=1}^k s_i = 1, \alpha \leq s_k \leq 1 - \alpha\}$ are obtained, when $s_k = \alpha$ and one among the quantities s_1, s_2, \dots, s_{k-1} attains the value $1 - \alpha$ and all the rest are equal to 0.*

Proof: This optimization problem is similar to the minimization of the entropy (equal to $-f$) of $p\langle s_1, s_2, \dots, s_k \rangle$ with the same constraints. It is known that the entropy is a concave function implying that f is convex. Thus the optimal value of the function f is at a vertex of the polytope defined by the constraints of the optimization problem. The polytope corresponding to all constraints excepted the last one is a simplex and its vertices are the unit vectors $(s_i = 1, s_{j \neq i} = 0)$ for all $i = 1, \dots, k-1$. None of these vertices satisfy this last constraints. Thus the vertices of the full polytope are points which

have their component s_k equal either to α or $1 - \alpha$ (the end point values of s_k). Once s_k fixed, the constraints become $s_i \geq 0$ and $\sum_{i=1}^{k-1} s_i = 1 - s_k$ which is a simplex as well. If a vertex has its component s_k equal to α then one of its other component is exactly equal to $1 - \alpha$ and all the rest is equal to 0. Else if a vertex has its component s_k equal to $1 - \alpha$ then one of its other component is exactly equal to α and all the rest is equal to 0. Thus the value of the function f for all those vertices is the same and corresponds to $\alpha \log \alpha + (1 - \alpha) \log(1 - \alpha)$. \square

3 Single hotlink assignment

3.1 KKS method

The KKS method [13] assigns one hotlink per node for trees with constant maximum degree d . It is a top-down method which simply chooses as hotlink of the root of the tree a node h defining a subtree T_h of weight satisfying

$$\frac{W(T)}{(d+1)} \leq W(T_h) \leq \frac{dW(T)}{(d+1)}. \quad (1)$$

This algorithm has a quadratic running time in the number of vertices of the tree and assigns for any probability distribution $p = \langle p_1, p_2, \dots, p_n \rangle$ on the n leaves of the tree one hotlink per node in such a way that the expected number of steps to reach a leaf from the root is at most $\frac{H(p)}{\log(d+1) - (d/(d+1)) \log d} + \frac{d+1}{d}$ (see [13]). Here we show that the upper bound of this method is asymptotically tight.

Lemma 4 *There is a tree for which the expected number of steps to reach a leaf from the root of the tree after the hotlink assignment given by the KKS method [13] is $\frac{H(p)}{\log(d+1) - (\frac{d}{d+1}) \log d + \tau}$, for any $\tau \geq 0$.*

Proof:

Consider a caterpillar $T(n)$ with n internal elements each pointing to one leaf, the distribution on the leaves is uniform, so $H(p) = \log n$.

Assume by induction that for any caterpillar $T(k)$ of size $k < n$, its KKS hotlink assignment gives an average access time $E[T^A(k)] \geq c \log k$. So we can express the average access time of a caterpillar of n elements by the following recurrence:

$$\begin{aligned}
E[T(n)] &\geq 1 + \frac{1}{d+1}E\left[T\left(\frac{n}{d+1} - d\right)\right] + \frac{d}{d+1}E\left[T\left(\frac{nd}{d+1} - d\right)\right] \\
&\geq 1 + \frac{1}{d+1}c\log\left(\frac{n}{d+1} - d\right) + \frac{d}{d+1}c\log\left(\frac{nd}{d+1} - d\right) \\
&\geq 1 + c\log(n) + c\left[\frac{1}{d+1}\log\left(\frac{1}{d+1}\right) + \frac{d}{d+1}\log\left(\frac{d}{d+1}\right)\right] \\
&\quad + \underbrace{c\frac{1}{d+1}\log\left(1 - \frac{d(d+1)}{n}\right) + c\frac{d}{d+1}\log\left(1 - \frac{d+1}{n}\right)}_{\approx -\frac{c}{\ln 2}\frac{2d}{n}}
\end{aligned}$$

Note that we consider the case where the hotlink of the head of a caterpillar of size k divides it in two subcaterpillars of size $\{k/(d+1), dk/(d+1)\}$ (see figure 3). This partition is not possible if k is not divisible by $(d+1)$, so to force the division we remove at most d element of a caterpillar.

The last expression of the recurrence above is larger than $cH(p)$ when constant c is chosen such that

$$\begin{aligned}
c &\leq \frac{-1}{\frac{1}{d+1}\log\left(\frac{1}{d+1}\right) + \frac{d}{d+1}\log\left(\frac{d}{d+1}\right) - \frac{2d}{n\ln 2}} \\
&= \frac{1}{\log(d+1) - \left(\frac{d}{d+1}\right)\log d + \frac{2d}{n\ln 2}}
\end{aligned}$$

We know that d is a constant, if we choose a caterpillar of size $n \geq \frac{2d}{\tau\ln 2}$ then the last term of the expression above is smaller than τ . \square

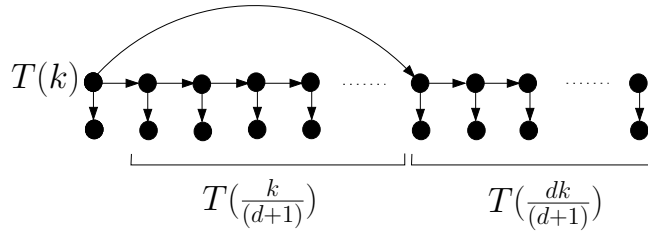


Figure 3: The hotlink of the head of the caterpillar divides it in two subcaterpillars.

3.2 MinMax method

A first intuition that could possibly be used to improve this top-down method is to consider that at each hotlink assignment we try to minimize the maximum weight of the children or the hotlink of the root. This can be done by assigning the hotlink of the root to a node h which partitions the heaviest subtree T_{max} of the root in two subtrees of equivalent weight, i.e. among all subtrees in T_{max} , T_h has weight nearest to the value $W(T_{max})/2$. We call this the *MinMax method*.

Lemma 5 *The MinMax hotlink assignment method guarantees an upper bound on the average access time of at most $\frac{H(p)}{\log(d+1) - (\frac{d}{d+1}) \log d} + \frac{d+1}{d}$.*

Proof: According to the MinMax method, the hotlink of the root of T will point to a node h in T_{max} of weight $W(T_h)$ nearest to $W(T_{max})/2$ among every other subtree in T_{max} , thus

$$\frac{W(T_{max})}{d+1} \leq W(T_h) \leq \frac{dW(T_{max})}{d+1}. \quad (2)$$

As $W(T_{max}) \leq W(T)$, the weight $W(T_h)$ could not have a value greater than $\frac{d}{d+1}$. If we assume that $W(T_h) \geq \frac{1}{d+1}$ then the method can be seen as a specification of the KKS hotlink assignment [13] method. The KKS method only specifies that the subtree defined by the hotlink of the root must be included in the range (1). Thus the performances of the MinMax method in this case are either equal or better than the KKS hotlink assignment.

In the opposite case where $W(T_h) \leq \frac{1}{d+1}$, we know that after the adoption of the subtree T_h by the hotlink of the root the weight of the subtree T_{max} will become $(W(T_{max}) - W(T_h)) \leq \frac{dW(T_{max})}{d+1} \leq \frac{d}{d+1}$. By the definition of T_{max} , which is the heaviest subtree defined by a child of the root, we know that none of the other children of the root can define a subtree of weight greater than $\min(W(T_{max}), 1 - W(T_{max})) \leq \frac{1}{2}$. Thus after one step of search in the tree T^A we never reach a subtree of weight greater than $\frac{d}{d+1}$. One of the $d+1$ subtrees of the root has a weight greater than $\frac{1}{d+1}$, by the previous argument it has also a weight smaller than $\frac{d}{d+1}$. Thus modulo a renumbering of the leaves we can guarantee the same constraints as in the KKS method. The MinMax method is so either equivalent or better than the KKS method.

We can conclude that the MinMax method can not be worst than the KKS hotlink assignment [13], i.e. the expected number of steps to reach a

leaf from the root of a tree is bounded by

$$\frac{H(p)}{\log(d+1) - \left(\frac{d}{d+1}\right) \log d} + \frac{d+1}{d}.$$

□

Lemma 6 *There is a tree for which the expected number of steps to reach a leaf from the root of the tree after applying the hotlink assignment according to the MinMax method is $H(p) \frac{d^2+3d}{2(d+1) \log(d+1)}$.*

Proof: Consider a tree T of $n = (d+1)^k$ leaves each of weight $\frac{1}{(d+1)^k}$ ($W(T) = 1$), T is recursively structured as follows: A simple list of d elements where the last one points to another element p connecting the subtrees $\{T_1, T_2, \dots, T_d\}$ each of weight $\frac{1}{(d+1)}$ and also points to a subtree T_{d+1} of weight $\frac{1}{(d+1)}$ (see figure 4). The structure is repeated in each of the subtrees T_1, \dots, T_{d+1} .

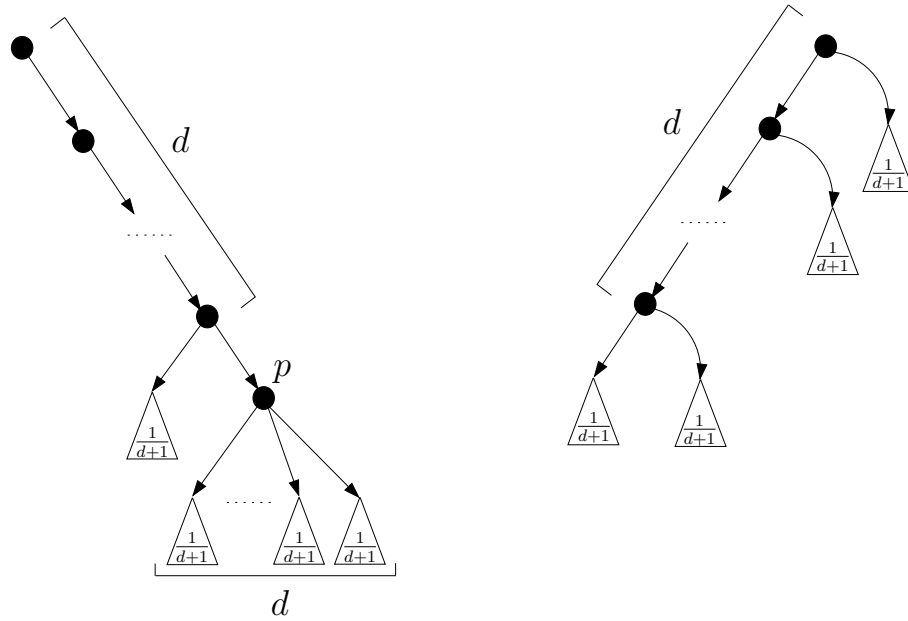


Figure 4: Recursive schema illustrating the lower bound of the *MinMax Method*.

According to the MinMax method, the hotlink of the root of T must be assigned to a subtree of weight nearest to $1/2$. Only T_p and $\{T_1, \dots, T_d, T_{d+1}\}$

respectively of weights $\frac{d}{(d+1)}$ and $\frac{1}{(d+1)}$ satisfy this constraint, consider that T_1 is chosen as the hotlink of the root.

Assume now that the hotlinks of the first $k < d$ elements of T point respectively to $\{T_1, T_2, \dots, T_k\}$. At the $k + 1$ element the weight of the resulting tree is $1 - \frac{k}{(d+1)}$, thus the hotlink of this element must be assigned to a the subtree of weight nearest to $\frac{d+1-k}{2(d+1)}$. Again T_p and $\{T_{k+1}, \dots, T_d, T_{d+1}\}$ satisfy this constraint, thus T_{k+1} can be chosen. That proves by induction that the hotlink of the first d elements of T could each point to a subtree of weight $\frac{1}{(d+1)}$ (see figure 4). After the hotlink assignment of T , the depth of the augmented subtrees $d(T_i^A) = i$ for $i = 1, \dots, d$ and $d(T_{d+1}^A) = d$.

Thus we can express the expected access time of T^A as

$$\begin{aligned}
E[T^A, p] &= \sum_{i=1}^d W(T_i^A) \left(d(T_i^A) + E[T_i^A, p^{(i)}] \right) \\
&= \sum_{i=1}^d \frac{1}{d+1} \left(i + E[T_i^A, p^{(i)}] \right) + \frac{1}{d+1} \left(d + E[T_{d+1}^A, p^{(d+1)}] \right) \\
&= \frac{d(d+1)}{2(d+1)} + \frac{d}{(d+1)} + E[T_i^A, p^{(i)}] \\
&= \frac{d^2 + 3d}{2(d+1)} + E[T_i^A, p^{(i)}]
\end{aligned}$$

As the number of leaves of T^A is equal to $n = (d+1)^k$, then each subtree T_i^A contains $\frac{n}{d+1}$ leaves. Thus the recurrence is solved as follows:

$$E[T^A, p] = \frac{d^2 + 3d}{2(d+1)} \frac{\log n}{\log(d+1)} = \frac{d^2 + 3d}{2(d+1)} \frac{H(p)}{\log(d+1)}.$$

□

So we conclude that the *MinMax method* of hotlink assignment could eventually offer an improvement to the *KKS* hotlink assignment, but the improvement would be at most about a factor 2 for the asymptotic behavior which is $\frac{d}{2 \log d}$ for the *MinMax method* and $\frac{d}{\log d}$ for the *KKS method*.

The problem of those two methods is that their average access time degrades as the maximum degree d of the trees considered grows. To avoid this increase in the expected number of steps to reach a leaf from the root of a tree, we introduce a new method in the next section.

3.3 h/p_h Method

The idea of this hotlink assignment method remains the same that the previous ones, the difference lies on the number of candidate nodes that we consider for the choice of the hotlink of the root of a subtree T . Namely, the candidates are firstly the node h of weight $w_{1/2}$ the nearest to $W(T)/2$ and secondly the parent node of h , denoted p_h . The method that determines how to choose among those candidates is called the h/p_h method: Let α be the unique solution of $\frac{\alpha}{1-\alpha} = \alpha^{\frac{1}{2(1-\alpha)}}$ (i.e. $\alpha \approx 0.2965$), if the weight $w_{1/2}$ of the node h is greater than the threshold α we take h as the hotlink of the root and we take p_h otherwise.

Before beginning the analysis of the method, we give a property of the nodes of weight $w_{1/2}$. Define the heavy path of a tree to be the path from the root to a leaf such that each node on the path is the heaviest child of its parent.

Lemma 7 *If $w_{1/2}$ is the weight nearest to $W(T)/2$ among all nodes in the subtree T , then there is a node of weight $w_{1/2}$ on its heavy path.*

Proof: Notice first that the value $w_{1/2}$ is not necessarily unique. It is clear that a node x of weight $w_{1/2}$ will be found on the heavy path if its weight is above $\frac{W(T)}{2}$. Else we know that the parent of x must have a weight greater than $W(T) - W(T_x) \geq \frac{W(T)}{2}$ implying that this parent node is on the heavy path. Consider now that x is not the heaviest of its brother, by the definition of $w_{1/2}$ a node heavier than x must have a weight greater or equal to $W(T) - W(T_x)$. But a brother of x can clearly not have a weight greater than $W(T) - W(T_x)$. Thus the only situation where x is not on the heavy path occurs when it has a unique brother on the heavy path of weight $W(T) - W(T_x)$ which is as close as $W(T)/2$ than the weight of x . \square

We can now begin to analyze the expected access time to reach a leaf from the root of a tree after the hotlink assignment according to the h/p_h method.

Theorem 1 *Consider a tree T of arbitrary maximum degree and T^A the same tree after the hotlink assignment of the h/p_h method. The maximum average access time to the leaves of T^A is at most $\frac{H(p)}{\log 3 - (2/3)} + 2/3$ if $d = 2$ and $H(p)\frac{2}{\log 1/\alpha} + 1 \approx 1.141H(p) + 1$ if $d > 2$.*

Proof: We can make an analysis of the worst average access time by selecting 3 ranges for the value of $w_{1/2}$:

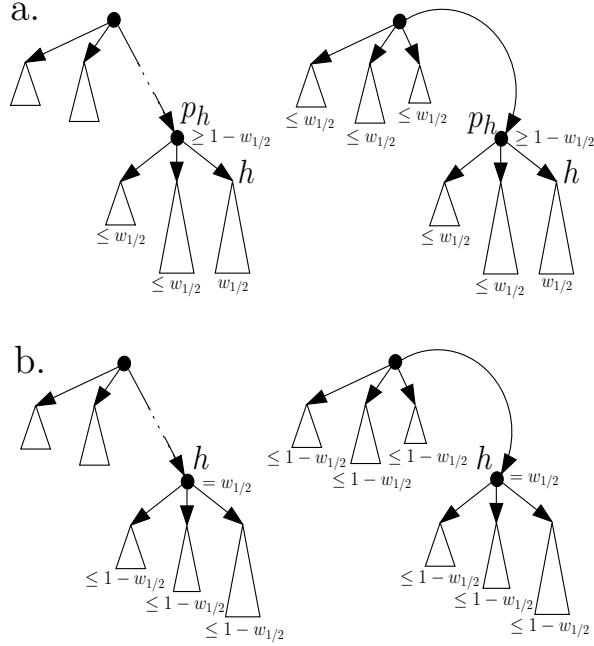


Figure 5: Before and after assigning the hotlink of the root to the node p_h (a.) or h (b.)

1. $0 \leq w_{1/2} < \alpha$, we are in the case where we must choose p_h as hotlink of the root. We know that the node h defines a subtree of weight $w_{1/2}$ nearest to $1/2$, thus the weight of the subtree defined by its parent node p_h is greater than $1 - w_{1/2}$ and the brother nodes of h define subtrees of weight smaller than $w_{1/2}$. After the assignment of the hotlink of the root to the node p_h , we know that none of the direct children of the root can have a weight greater than $1 - W(T_{p_h}) \leq w_{1/2}$.

That guarantees that after two steps of search from the root of the tree after the hotlink assignment we can not reach a subtree of weight greater than $w_{1/2} \leq \alpha$ (see Fig. 5.a). Using the notation of Lemma 2, we can express the worst expected number of steps to reach a leaf from the root of the tree in the case where we choose the p_h node as hotlink of the root in the current range: $E[T^A, p] \leq aH(p) + 1$ with $a \geq -\frac{2}{\log \alpha}$.

2. $\alpha \leq w_{1/2} < 1 - \alpha$, we are in a range where we must choose the node h as hotlink of the root. Note that the children $\{c_1, c_2, \dots, c_k\}$ of the root of T^A other then h have a weight of at most $(1 - w_{1/2})$ and

the node h has weight $w_{1/2}$. All subtrees of the root have a depth of 1. This partition gives a worst average access time to the leaves equal to $E[T^A, p] \leq aH(p) + 1$ with $a \geq -\frac{1}{(w_{1/2}) \log(w_{1/2}) + \sum_{i=1}^k (W(T_{c_i})) \log W(T_{c_i})}$ (see Lemma 2). The maximum value of this last function subject to the constraints $\{\alpha \leq w_{1/2} \leq 1 - \alpha, \sum_{i=1}^k W(T_{c_i}) = 1 - w_{1/2}\}$ is given by Lemma 3, i.e. when $w_{1/2} = \alpha$, $W(T_{c_1}) = 1 - \alpha$ and $W(T_{c_i}) = 0$ for all $2 \leq i \leq k$. Thus the maximum expected number of steps to reach a leaf in this current range is $\frac{H(p)}{-(\alpha \log \alpha + (1-\alpha) \log(1-\alpha))} + 1$.

3. $1 - \alpha \leq w_{1/2} \leq 1$, we choose h as hotlink of the root. The node h defines a subtree of weight $w_{1/2}$ nearest to $1/2$, thus its heaviest child has a weight smaller than $1 - w_{1/2} \leq \alpha$, and the weight of the direct children of the root of the tree after the hotlink assignment can not exceed $1 - w_{1/2} \leq \alpha$ (see Fig. 5.b). By those facts, we know that none of the subtrees reachable after two steps of search can have a weight greater than α . That is exactly the same situation as in the first case where $0 \leq w_{1/2} \leq \alpha$, thus the worst average access time to the leaves will be the same, i.e. $-\frac{2H(p)}{\log \alpha} + 1$.

We saw that if $\alpha \leq w_{1/2} \leq 1 - \alpha$ then the worst access time is equal to $\frac{H(p)}{-(\alpha \log \alpha + (1-\alpha) \log(1-\alpha))} + 1$, and for any other value of $w_{1/2}$ we have $-\frac{2H(p)}{\log \alpha} + 1$. Thus we can compute the value of α for which both expressions are equal, i.e. for which value of α the choice of h or p_h is equivalent. This occurs when $\frac{\alpha}{1-\alpha} = \alpha^{\frac{1}{2(1-\alpha)}}$ (i.e. $\alpha \approx 0.2965$), and the maximum expected number of steps needed to reach a leaf from the root of a tree T^A is no more than $H(p) \frac{2}{\log \frac{1}{\alpha}} + 1 \approx 1.141H(p) + 1$.

Thus for any tree with a maximum degree $d > 2$, the h/p_h method gives a better ratio for the approximation of the optimum hotlink assignment than the *KKS* method. But we can remark that if $d = 2$ then the h/p_h method cannot be worse than the *KKS* method. Indeed, in this case the value of $w_{1/2}$ is bounded above by $1/3$ and below by $2/3$, that implies that the h/p_h method always chooses the node h as hotlink of the root. This choice will be better or at least equivalent to the choice of the *KKS* hotlink assignment. So the h/p_h method is better in all the cases. \square

4 Multiple hotlink assignment

In the preceding sections we saw the hotlink assignment problem in the case where just one hotlink per node of a tree is allowed. Now we consider the addition of k hotlinks for each node. Some studies have already been done on this topic, namely S. Fuhrmann *et al.* [10] present algorithms to

reduce the height of a tree by a constant factor. The algorithms for optimal hotlink assignment by dynamic programming allow k hotlinks assignments per node [11, 16]. The KKS method [13] has been generalized by Bose *et al.* [4] to assign k hotlinks per node, but is restricted to binary trees, it guarantees an average access time at most $\frac{H(p)}{\log(k+2)-1} + 1$.

We introduce in this paper a recursive top-down method which performs up to k hotlink assignments, seen as adoptions, to the root of a tree T of arbitrary degree d to obtain an enhanced tree T' . Then the procedure is iterated for each child of the root in T' . This method is a natural generalization of the algorithm of Bose *et al.* [4] for trees of arbitrary maximum degree. When processing a node x , we perform hotlink assignments of the node x until each original child y of x is either a leaf or its weight satisfies $W(T_y) \leq dW(T_x)/(k+d)$. To determine which descendant z to assign next for a hotlink of the node x , we start at the non-leaf original heaviest child of x and we traverse its heavy path until reaching the node z of maximum weight smaller than $dW(T_x)/(k+d)$.

Thus all the hotlink nodes h_i of x have a weight greater than $W(T_x)/(k+d)$ implying that at most k hotlinks can be assigned by node, indeed the original non-leaf children of x after k assignments cannot have a weight greater than $W(T_x) - kW(T_x)/(k+d) = dW(T_x)/(k+d)$ which is the condition to stop.

Theorem 2 *Consider a tree T of maximum degree d and T^A the same tree after the hotlink assignment of the generalized method of [4]. The maximum average access time to the leaves of T^A is at most*

$$\frac{H(p)}{\log(k+d) - \log d}.$$

Proof: After assignment, all non-leaf children of each node x of T^A have a weight smaller than $dW(T_x)/(k+d)$. By Lemma 2, this implies that T^A has an expected access time to its leaves that can be expressed as $E[T^A, p] \leq aH(p) + 1$, at the condition that the constant a is chosen such that $a \geq \frac{1}{\log(k+d) - \log d}$. \square

The performances of this generalized method degrades as d grows. The next method avoids this dependence on d . Here, we perform hotlink assignments for the node x until each original child y of x is either a leaf or its weight satisfies $W(T_y) \leq W(T_x)/(k+1)$. To determine which descendant z to assign next for a hotlink of x , we start at the non-leaf original heaviest child of x and we traverse its heavy path until reaching the node z of

minimum weight greater than $W(T_x)/(k+1)$. While processing a node x , at most k hotlinks are assigned. Indeed the assignment stops when each original child of x is either a leaf or its weight is smaller than $W(T_x)/(k+1)$. After k hotlink assignments, a non-leaf child of the node x cannot have a weight greater than $W(T_x) - kW(T_x)/(k+1) = W(T_x)/(k+1)$.

Theorem 3 *Consider a tree T of maximum degree d and T^A the same tree after the hotlink assignment of the above multiple hotlink assignment method. The maximum average access time to the leaves of T^A is at most $2H(p)/\log(k+1)$ for $d > \sqrt{k+1}$, and $H(p)/(\log(k+1) - \log d)$ otherwise.*

Proof: After applying the above method, each node h_i chosen as hotlink for the root of the tree T defines a subtree of minimum weight greater than $W(T)/(k+1)$. That implies that the heaviest child of the nodes h_i has a weight smaller than $W(T)/(k+1)$ and its $d-1$ brothers cannot have a weight greater than it. Thus the weight of each node h_i doesn't exceed $dW(T)/(k+1)$. We also know by the previous lemma that the hotlink assignment stops when all the non-leaf original children of the root have a weight smaller than $W(T)/(k+1)$. So after one step of search in T^A , the tree after assignment, it is impossible to reach a subtree of weight greater than $dW(T)/(k+1)$.

By Lemma 2, this implies that the expected access time can be expressed as $E[T^A, p] \leq aH(p) + 1$, at the condition that the constant a is chosen such that $a \geq -\frac{1}{\log(k+1) - \log d}$.

This method of hotlink assignment guarantees that after two steps of search from the root of T^A we never find a subtree of weight greater than $W(T^A)/(k+1)$. Indeed, we know that all the direct children of the node h_i have a weight smaller than $W(T^A)/(k+1)$ and none of the non-leaf original children of the root could have a weight greater than $W(T^A)/(k+1)$. Again using Lemma 2 we obtain $a \geq -\frac{2}{\log k+1}$. The first bound on a is better when $d \leq \sqrt{k+1}$. \square

5 Fast hotlink assignment algorithm

In order to perform the hotlink assignment according to the methods introduced previously, a naive $O(n^2)$ running time algorithm can be easily found. Here we present an $O(n \log n)$ running time algorithm which uses an enhanced version of the *Link-Cut Trees*.

The *Link-Cut Trees* or *ST Trees* of D.D.Sleator and R.E.Tarjan is a data structure for the *Dynamic trees problem* [18]. Namely, we are given a

collection of vertex-disjoint rooted trees. We want to represent the trees by a data structure that allows us to easily extract certain informations (the cost of an edge, the minimum cost on a precise path, the parent of a node, the root of a node) about the trees and to easily update the structure to reflect changes in the trees caused by these two kinds of operations: *link* the root of a tree to any node of an other tree making this node the parent of the root, and *cut* a tree into two trees by deleting the edge from a selected node to its parent.

They develop a solution to the dynamic trees problem by using an implicit representation of the forest, which sees dynamic trees as sets of *solid paths* connected together with *dashed* edges (see Fig. 7.a). Each tree operation is carried out by means of one or more path operations. These dynamic solid paths are represented as biased binary trees (BBT) [2] (or splay trees [19]) whose external nodes correspond to the vertices of the solid paths and internal nodes correspond to subpaths (see Fig. 7.b). This data structure guarantees that each dynamic tree operation takes $O(\log n)$ time in the worst-case but only if the partition in solid paths is done by size (number of leaves inside the tree defined by a node), i.e. if the *solid paths* are defined to be the paths from the root of a subtree to a leaf where each node is the child of its parents which has the greatest size.

The remainder of this section modifies the Link-Cut tree structure, we refer the reader to [18] for more details.

Enhanced Link-Cut trees: Remember that the weight $W(T_v)$ of a vertex v is the sum of the weight of its children in the original tree T , where each leaf i in T is associated with a weight w_i representative of its access frequency.

Now we shall see how to enhance the Link-Cut Trees to use it for the hotlink assignment. First we add an extra part to the structure. For each vertex v on a solid path, we maintain a vertex set containing the children of v excepted the child corresponding to the next vertex $next[v]$ on the solid path of v (if it exists). We allow three kinds of operations on the vertex sets: (1) *maxw(vertex v)*, return the vertex of maximum weight in the vertex set of v ; return **null** if the vertex set is empty. (2) *insert(vertex u, vertex v)*, insert vertex u into the vertex set of v . (3) *delete(vertex u, vertex v)*, delete vertex u from the vertex set of v . We represent the vertex set of a vertex by a globally biased binary tree [2], the vertices appearing as external nodes, exactly as for the structures used in the original Link-Cut trees.

Finally we add one more field to each internal node or leaf x in the associated BBT of the solid paths: If the node x is a leaf of a BBT then the value wt_x is set to the sum of the weights of its children in the original tree

excepted its next vertex on the solid path. Else the node x is an internal node of the BBT and wt_x is set to the sum of the value wt of its children in the BBT. We note that this information can be updated in a constant time after any rotation operation.

If the node x in the original tree T is contained in the solid path S , then the weight $W(T_x)$ can be computed with the value wt stored in the nodes of the BBT associated to the solid path S , i.e. $W(T_x) = \sum_{i \in R(x)} wt_i$ where $R(x)$ represent the right siblings (set of right child of nodes) on the path to the root of the BBT associated to S .

Note that these two extra structures, i.e. for the vertex sets and the values wt , are nearly identical to some structures present in the original Link-Cut trees, used to maintain the solid paths and to compute the size (in number of nodes) of the subtree of a node. Thus the added structures will be updated using the same techniques, achieving the same performances.

Search: Consider now the hotlink assignment and see how to use the enhanced Link-Cut tree to perform the search of the candidate node which will be pointed to by one of the k hotlinks of the root of the original tree. For all methods presented here, this candidate node will be found from a node h which defines a subtree of minimum weight greater than $W(T)/c$ for any fixed constant $c \geq 1$ depending on the method used. We can deduce from Lemma 7 or from the method itself that this node h is always located on the *heavy path*, this heavy path is defined as the path from the root of T to a leaf connecting each node on the path to its heaviest child. But in the Link-Cut tree, the decomposition of the initial tree is done by *solid paths* (decomposition by size), thus the heavy path could traverse several solid paths (see

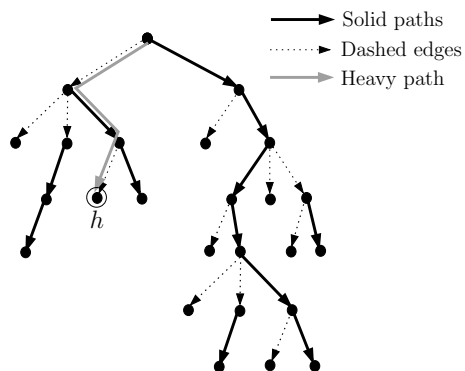


Figure 6: The heavy path could intersect several solid paths.

Fig. 6). The search of the node h is thus a succession of searches in multiple BBTs each associated to a solid path which intersects the heavy path.

The search is performed as follows: we begin from the BBT associated with the solid path containing the root of the original tree T , and use it to locate the lowest vertex v greater than $W(T)/c$. In order to perform this search efficiently we use the information wt stored in the nodes of this associated BBT. We walk down the BBT from its root r and we maintain a value $Z = \sum_{i \in R(j)} wt_i$ where j is the current node, i.e. Z is equal to the sum of the value wt of the right siblings of nodes on the path from the current node j to the root r . Thus $Z + wt_j$ is the maximum weight of any leaf reachable from the node j . We initially start from the root r and we set $Z = 0$. If $Z + wt_{right[r]} \geq W(T)/c$ we go down by the right child $right[r]$ of the root else we go by the left child and we update $Z = Z + wt_{right[r]}$. We iterate the process until we find the vertex v on the solid path of minimum weight greater than $W(T)/c$. Note that the value Z is equal to $W(T_{next[v]})$ when the node v is found. An illustration of this search is shown in Fig. 7.b.

If the the weight of the next vertex of v in its solid path is greater than $maxw(v)$, i.e. if $W(T_{next[v]}) \geq maxw(v)$ then v corresponds to the node h that we are looking for. Else we must check if the node h is present in the next solid path beginning by the vertex of weight $maxw(v)$. For that, we perform the same search in the associated BBT of this next solid path. We iterate the process until we find the node h .

According to the Link-Cut tree performance, we can find a node i contained in an associated BBT rooted at r in $O(\log \frac{Size(r)}{Size(i)})$ time, corresponding to the height of the BBT. The sum of the running times of the successive searches in the different solid paths is $\log \frac{Size(T)}{Size(x_1)} + \log \frac{Size(x_1)}{Size(x_2)} + \dots + \log \frac{Size(x_k)}{Size(h)} \leq \log \frac{Size(T)}{Size(h)} \leq \log n$, where x_1, x_2, \dots, x_k are vertices leading to the successive solid paths traversed by a search. We must add to that the number of times that we use $maxw()$ for a vertex set to check if the node h is deeper in the tree (takes $O(1)$ time), this number is bounded by $\log n$ because of the definition of the solid path. Thus the maximum total running time needed to find the node h which gives the necessary information to find the candidate for the hotlink assignment of the root is $O(\log n)$.

Cut: To perform the hotlink assignment, we just need the *Cut* operation which consists in cutting a tree T into two trees by deleting the edge from a selected node to its parent. The cut operation with an enhanced Link-Cut tree is done as in the original structure excepted that the extra structures (vertex sets and fields wt) have to be updated.

Cutting a subtree rooted at a node h consists first in making an *expose*

operation on the node h . That operation creates a single solid path, ending in h and beginning at the root of the original tree T , by converting *dashed edges* (connecting two distinct solid paths) to *solid* (connecting two vertices of the same solid path) along the tree path from h to the root of the original tree T and converting solid edges incident to this path to dashed.

Those kinds of edge conversions may change the vertex sets associated to several vertices of the original tree T . The dashed edges converted in solid must be deleted from the corresponding vertex set and respectively the solid edges converted in dashed must be inserted in their corresponding vertex set. The value wt of the nodes is also affected by those changes and have to be updated following the Link-Cut tree methods.

After the expose of the node h , we have to cut the subtree rooted at h and update the values of some nodes in the associated BBT of the solid path containing h , i.e. all nodes i where we walk to their right child during a search for h in the associated BBT have to update their value wt_i to $wt_i - W(T_h)$. Once this is done, we restructure the associated BBT and we repair the damage caused by the expose. Namely after the cut, the decomposition into solid paths could have changed and we must update the structure by an operation which can be seen as an expose running backwards. This operation is fully described in [18].

Thus the cut in a enhanced Link-Cut tree has the same asymptotic running time than in the original structure, i.e. each cut operation takes $O(\log n)$ time, where n is the number of nodes in the initial tree.

Lemma 8 *The hotlink assignment of a tree T according to the methods described in the previous sections can be done in $O(n \log n)$ time using the enhanced Link-Cut trees data structure seen above.*

Proof: Consider that we use an enhanced Link-Cut tree data structure as described above for a tree T . The hotlink assignment consists in finding a node h for one hotlink of the root according to the desired method. We have seen above that this search is performed in $O(\log n)$ time (Fig. 7.b). Once h is found, we cut the edge between h and its parent. This cut takes $O(\log n)$ time using the enhanced Link-Cut trees. For the multiple assignment methods we carry out the same operation as long as necessary. Once all the hotlinks of the root has been assigned we cut all the edges connecting the root to its children, those cuts are done in $O(\log n)$ (Fig. 7.c), thus we obtain at most $d + k$ subtrees for which we iterate the same process recursively (Fig. 7.d).

Although each node could have up to k hotlinks, the total number of hotlinks assigned is smaller than n because there cannot be more than one

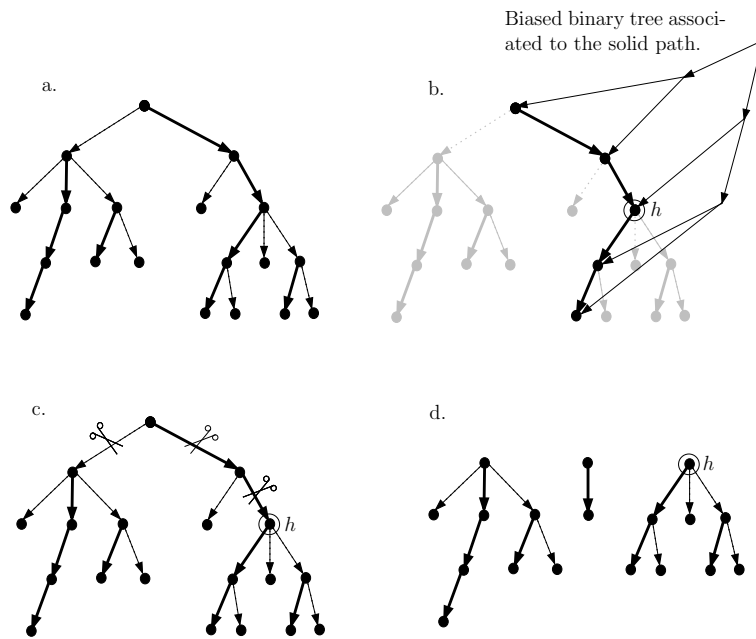


Figure 7: a. Decomposition of a tree T in solid paths. b. A search in the biased binary tree representing the solid path. c. Cut of the node h and the children of the root. d. Resulting trees.

hotlink pointing to each node. Thus the enhanced link-cut trees allow to perform a hotlink assignment for a node in $O(\log n)$ time, this must be done at most n times which implies that the entire hotlink assignment takes $O(n \log n)$ time. \square

6 Lower bound on running time of approximation methods

We consider here hotlink assignment methods for trees which guarantee a maximum average access time to their leaves at most $\alpha H(p)$ with $\alpha < 2$. They must be *strong* in the sense that they guarantee the same bound on the access time for all subtrees composing the augmented tree. Namely, if $p^{(i)}$ is defined to be the relative access distribution of the leaves inside the subtree T_i (see section 2) then the methods should guarantee an average access time of at most $\alpha H(p^{(i)})$.

Theorem 4 *Strong hotlink assignment methods guaranteeing a average access time at most $\alpha H(p)$ with $\alpha < 2$ cannot have a running time less than $\Omega(n \log n)$ in the computation tree model.*

Proof: We prove the above theorem by a reduction from the following sorting problem:

INPUT: A permutation S of integers going from 1 to n , with $|S| = n$.

OUTPUT: Sort S in an ordered sequence.

This problem is clearly known to have a lower bound of $\Omega(n \log n)$ in the computation tree model. Given an instance $S = \{s_1, s_2, \dots, s_n\}$ of the sorting problem we construct an instance T of the hotlink assignment problem. The tree T is defined as follows: We build a simple list of nodes $\{x_0, x_1, \dots, x_{n-1}\}$, at the end of which we attach a node a connected to n leaves (see Fig. 8). We set the weight of each of those leaves l_i to $\frac{k-1}{k^{n-s_i+1}}$ if $s_i \geq 2$, $\frac{1}{k^{n-1}}$ otherwise.

We remark that for any $k \geq 2$ the sum of the weights of all the leaves is equal to 1 and the weight of the i^{th} heaviest leaf corresponds to $(k-1)$ times the sum of the weights of leaves of smaller weight, excepted for the last one.

The optimum assignment of the constructed tree T consists in assigning the heaviest leaf as hotlink of its root, and do the same recursively with the resulting subtree. This leads to a situation where the heaviest leaf is reached

in one step of search, the second heaviest in two steps and so on. Thus the nodes $\{x_0, x_1, \dots, x_{n-1}\}$ will point to the leaves in sorted order of their weights, and implicitly sort S . The average access time opt_i of the optimum assignment on T_{x_i} can be expressed as

$$\begin{aligned} opt_i &= \frac{n-i}{k^{n-1}} + \sum_{j=i+1}^{n-1} (j-i) \left(\frac{k-1}{k^j} \right) \\ &= \frac{n-i}{k^{n-1}} + \frac{k^{n-i} - (n-i)k + (n-1-i)}{k^n - k^{n-1}}. \end{aligned}$$

Note that opt_0 is the largest of all opt_i where $0 \leq i \leq n-1$. Now we show that any non optimal hotlink assignment of a strong method gives an average access time greater than $2 - \frac{(k-1)}{k^{i+2}}$ for some T_{x_i} . Let x_i be the first node not pointing to the heaviest leaf among the ones which have not already been adopted by the ancestors of x_i . Namely x_i does not point to the node of weight $\frac{k-1}{k^{i+1}}$ if $0 \leq i < n-1$ or $\frac{1}{k^{n-1}}$ if $i = n-1$. Now consider the different possible assignments for the hotlink of x_i :

- If the hotlink points to a node of the list $\{x_i, x_{i+1}, \dots, x_{n-1}\}$ or to the node a which is connected to every leaf, then we are in a situation where all the leaves are at a distance at least 2 from the root of T_{x_i} . That gives an average access time at least 2.

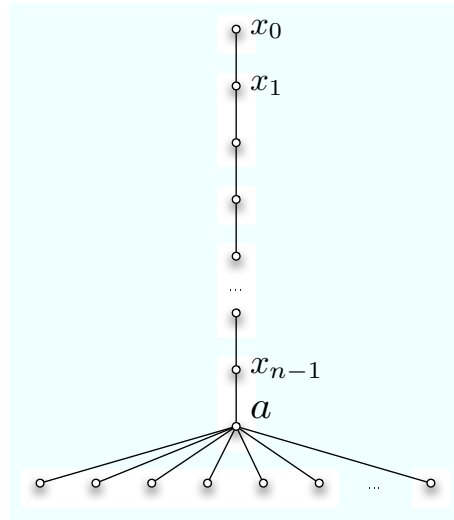


Figure 8: The tree T .

- If we choose a leaf of weight smaller or equal to $\frac{(k-1)}{k^{i+2}}$ for the hotlink of the root of T_{x_i} where $0 \leq i < n - 1$, then we are in a situation where a leaf of weight smaller than $\frac{(k-1)}{k^{i+2}}$ is at distance 1 from the root and every other leaf is at distance at least 2 from it, the average access time in this situation is greater than $2 - \frac{(k-1)}{k^{i+2}}$.

The approximation ration for node x_i is $\frac{2 - \frac{(k-1)}{k^{i+2}}}{opt_i}$. The value k is chosen so that this ratio exceeds α for all x_i , i.e.

$$\begin{aligned} \alpha &\leq \min_{0 \leq i < n} \left(\frac{2 - \frac{(k-1)}{k^{i+2}}}{opt_i} \right) \\ &= \frac{2 - \frac{(k-1)}{k^2}}{\frac{n}{k^{n-1}} + \frac{k^n - nk + (n-1)}{k^n - k^{n-1}}}. \end{aligned} \quad (3)$$

Remark that the limit of this ratio is 2 as k approaches infinity. Thus any strong α -approximation method of hotlink assignment which guarantees an average access time at most αopt_i for every i , gives the optimum assignment for T if $\alpha opt_i \leq 2 - \frac{(k-1)}{k^{i+2}}$, which is always true because the constant k was fixed to satisfy (3). This implicitly shows that any α -approximation method could be used to sort S , any of those has then a running time of $\Omega(n \log n)$. This holds for strong methods guaranteeing an average access time at most $\alpha H(p)$ because $\alpha H(p^{(i)}) \leq \alpha opt_i$ in this specific reduction. Indeed the optimal average access time for T results in a binary tree T^A with an average access time of opt_0 , which is greater than the entropy lower bound $H(p)$ on the average access time of a binary tree. \square

7 Conclusion

We develop hotlink assignment methods, either for assigning one or k hotlinks per element of an arbitrary tree. These methods run in $O(n \log n)$ time, which we show is the best possible (in the computational tree model) for strong methods guaranteeing an average access time to the leaves of at most $\alpha H(p)$ for $\alpha < 2$. Among the previous methods which run in $O(n^2)$ time or smaller, our methods achieve the best performances so far.

This contrasts with our previous work [7] where we presented a linear time method for assigning one hotlink per element of a tree. This method guarantees an average access time to the leaves of $3H(p)$ although we believe that the exact constant for that algorithm lies strictly between 2 and 3. To

determine if linear time strong methods can guarantee an average access time of $2H(p)$ or if non-strong linear time methods can achieve better bounds are still open problems.

References

- [1] <http://news.netcraft.com>.
- [2] S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased search trees. *SIAM J. Comput.*, 14, number 3:545–568, 1985.
- [3] P. Bose, E. Kranakis, D. Krizanc, M. V. Martin, J. Czyzowicz, A. Pelc, and L. Gasieniec. Strategies for hotlink assignments. In *Proc. 11th Ann. Int. Symp. on Algorithms and Computation*, volume 1969 of LNCS, pages 23–34, 2000.
- [4] P. Bose, D. Krizanc, S. Langerman, and P. Morin. Asymmetric communication protocols via hotlink assignments. In *Proc. 9th Int. Coll. on Structural Information and Communication Complexity (SIROCCO 2002)*, pages 33–40, 2002.
- [5] H. Brönnimann, F. Cazals, and M. Durand. Randomized jumplists : A jump-and-walk dictionary data structure. *Proc. 20th Ann. Symp. on Theoretical Aspects of Computer Science (STACS 2003)*, 2607 of LNCS, 2003.
- [6] J. Czyzowicz, E. Kranakis, D. Krizanc, A. Pelc, and M. Martin. Evaluation of hotlink assignment heuristics for improving web access. In *Proc. 2nd Int. Conf. on Internet Computing (IC'2001)*, pages 793–799, 2001.
- [7] K. Douïeb and S. Langerman. Dynamic hotlinks. In *Algorithmica, to appear. Special issue of selected papers from the 9th Workshop on Algorithms and Data Structures (WADS 2005)*.
- [8] M. Drott. Using web server logs to improving site design. In *Proc. ACM Conf. on Internet Computer Documentation*, pages 43–50, 1998.
- [9] A. Elmasry. Deterministic jumplists. *Nordic Journal of Computing*, 12:27–39, 2005.
- [10] S. Fuhrmann, S. O. Krumke, and H.-C. Wirth. Multiple hotlink assignment. In *27th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2204 of LNCS, pages 189–200, 2001.

- [11] O. Gerstel, S. Kutten, R. Matichin, and D. Peleg. Hotlink enhancement algorithms for web directories. In *Proc. 14th Ann. Int. Symp. on Algorithms and Computation*, volume 2906 of LNCS, pages 68–77, 2003.
- [12] E. Kranakis, D. Krizanc, and M. V. Martin. The hotlink optimizer. In *Proc. 3rd Int. Conf. on Internet Computing (IC'2002)*, pages 33–40, 2002.
- [13] E. Kranakis, D. Krizanc, and S. Shende. Approximate hotlink assignment. In *Proc. 12th Ann. Int. Symp. on Algorithms and Computation*, volume 2223 of LNCS, pages 756–767, 2001.
- [14] N. Abramson. Information theory and coding. *McGraw Hill*, 1963.
- [15] M. Perkowitz and O. Etzioni. Towards adaptive Web sites: conceptual framework and case study. *Computer Networks*, 31(11-16):1245–1258, 1999.
- [16] A. Pessoa, E. Laber, and C. de Souza. Efficient algorithms for the hotlink assignment problem: The worst case search. In *Proc. 15th Ann. Int. Symp. on Algorithms and Computation*, volume 3341 of LNCS, page 778, 2004.
- [17] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Proc. Workshop on Algorithms and Data Structures*, volume 382 of LNCS, pages 437–449, 1989.
- [18] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–381, 1983.
- [19] D. D. Sleator and R. E. Tarjan. Self-adjusting binary trees. *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pages 235–245, 1983.