Near-Entropy Hotlink Assignments

Karim Douïeb[⋆] and Stefan Langerman[⋆⋆]

Département d'Informatique, Université Libre de Bruxelles, Belgique.
`{kdouieb,stefan.langerman}@ulb.ac.be`

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 weight w_i representative of the frequency with which it is visited. By adding hotlinks — shortcuts from a node to one of its descendents — 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. Finally, we present an algorithm implementing these methods in $O(n \log n)$ time, an improvement over the previous $O(n^2)$ time algorithms.

1 Introduction

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 assignment*. The problem of the *hotlink assignment* was originally introduced by Perkowitz and Etzioni [13] to improve the search in Web sites.

The *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).

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

[⋆] Boursier FRIA

[⋆⋆] Chercheur qualifié du FNRS

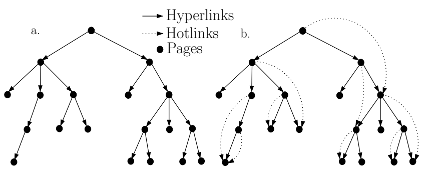

Fig. 1. a. Modeled site web, b. Example of hotlink assignment.

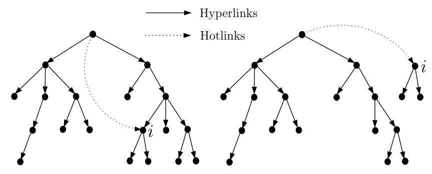

Fig. 2. Consequence due to the *greedy user* model assumption.

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 [9], 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).

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 $p = \langle p_i = w_i/W : i = 1, \ldots, n \rangle$ 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 [2] using information theory [12]. Let $H(p)$ be the entropy of the probability distribution p, defined by $H(p) = \sum_{i=1}^{n} p_i \log(1/p_i)$, 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 Perkowitz and Etzioni [13] to improve the search in Web sites (seen as DAGs). Later Bose et al. [2] 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 [11] 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. [8]. Recently, Gerstel et al.[9], and A.A. Pessoa et al. [14] independently discovered a polynomial time dynamic programming algorithms 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 are given in [5], and a software tool to structure websites efficiently by automatic assignment of hotlinks has been developed [10].

The concept of hotlinks can be applied to other problems than that of web structuring. For instance, Bose et al.[3] 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 [4] 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 [15]. A deterministic version of the randomized jumplist was developed by Elmasry [7] and by Douïeb and Langerman [6], independently.

Using this deterministic jumplist, we recently introduced a linear time algorithm [6] to allow the assignment of one hotlink per node in such a way that the number of steps to reach a node i from the root of a tree is bounded by the entropy, namely by $(3+\epsilon)H(p)$ for any $\epsilon > 0$. 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 [9,14] 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 [11], 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). The running time of this algorithm is $O(n^2)$ where n is the number of elements in the tree.

After showing some preliminary lemmas in the next section, a new top-down method for assigning one hotlink per node is presented in Section 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, 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.* [3] for assigning k hotlinks per node of 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.

Finally in the Section 5 we develop a fast algorithm for the methods seen in the preceding section; it uses an enhanced version of the link-cut trees of Sleator and Tarjan [16] and performs the hotlink assignment in $O(n \log n)$ time for all our methods and the KKS method [11]. This is an improvement over the previous $O(n^2)$ algorithms. Omitted proofs appear in the full version.

2 Top-Down Methods

Before giving some hotlinks assignment methods and their analysis we present a useful Lemma concerning entropy. Consider a probability distribution $p = \langle p_1, p_2, \ldots, p_n \rangle$ and a partition $A_1, A_2, \ldots, A_k$ of the index set $\{1, 2, \ldots, n\}$ into k non-empty subsets. Define $S_i = \sum_{j \in A_i} p_j$ for $i = 1, 2, \ldots, k$. Consider the new distributions: $p^{(i)} = \langle p_j^{(i)} := \frac{p_j}{S_i} : j \in A_i \rangle$ for $i = 1, 2, \ldots, k$. Kranakis, Krizanc and Shende [11] proved the following lemma:

Lemma 1. *For any partition $A_1, A_2, \ldots, A_k$ 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 $T_1^{\mathcal{A}}, T_2^{\mathcal{A}}, \ldots, T_k^{\mathcal{A}}$ with corresponding weights $S_1, S_2, \ldots, S_k$. 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 in the subtrees $T_1^{\mathcal{A}}, T_2^{\mathcal{A}}, \ldots, T_k^{\mathcal{A}}$ of weights $S_1, S_2, \ldots, S_k$ 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 $E[T^{\mathcal{A}}, p] \leq aH(p) + 1$.*

Finally we generalize Lemma 5 of [11]:

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

3 Single hotlink assignment: *h/ph* method

The KKS method [11] assigns one hotlink per node for trees with a 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 $\frac{W(T)}{(d+1)} \leq W(T_h) \leq \frac{dW(T)}{(d+1)}$.

The running time of this algorithm is quadratic in the number of vertices of the tree and assigns for any probability distribution $p = \langle p_1, p_2, \ldots, p_n \rangle$ on the n leaves of a tree one hotlink per node such a way that the expected number of steps to reach a leaf of the tree from the root is at most $\frac{H(p)}{\log(d+1) - (d/(d+1)) \log d} + \frac{d+1}{d}$ (see [11]). This bound is asymptotically tight for the KKS method, and is achieved by a caterpillar with uniform distribution on its leaves.

The problem of this method is that its 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. The h/p_h *Method* :

The idea of this hotlink assignment method remains the same, 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 4. *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.*

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_hmethod.

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 $H(p) \frac{2}{\log 1/\alpha} + 1 \approx 1.141 H(p) + 1$ if $d > 2$.*

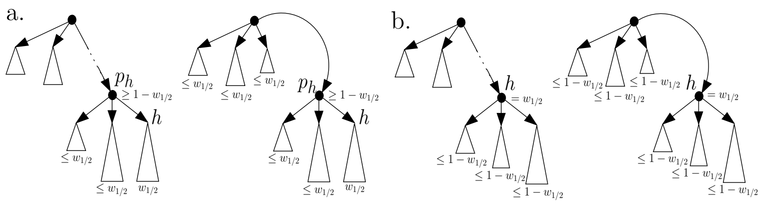
Fig. 3. Before and after assigning the hotlink of the root to the node p_h (a.) or h (b.)

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

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. 3.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 $\{c_1, c_2, \ldots, c_k\}$ 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 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. 3.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.* [8] 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 [9,14]. The KKS method [11] has been generalized by Bose *et al.* [3] 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.* [3] 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 [3]. The maximum average access time to the leaves of T^A is at most $\frac{H(p)}{\log(k+d)-\log d}$.*

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.*

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* [16]. 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 an node, the root of an 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. 5.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) [1] (or splay trees [17]) whose external nodes correspond to the vertices of the solid paths and internal nodes correspond to subpaths (see Fig. 5.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 [16] 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 [1], 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 4 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

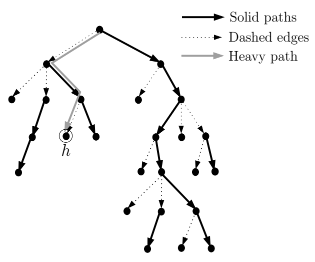

Fig. 4. The heavy path could intersect several solid paths.

the heavy path could traverse several solid paths
(see Fig. 4). 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 $rigth[r]$ 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. 5.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 $\log \frac{Size(T)}{Size(x_1)} + \log \frac{Size(x_1)}{Size(x_2)} + \cdots + \log \frac{Size(x_k)}{Size(h)} \leq \log \frac{Size(T)}{Size(h)} \leq \log n$, where $x_1, x_2, \ldots, x_k$ 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 cuting 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 [16].

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 5. *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. 5.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 hotinks 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. 5.c), thus we obtain at most $d + k$ subtrees for which we iterate the same process recursively (Fig. 5.d).

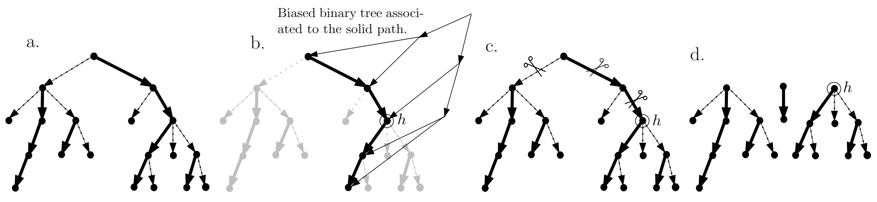

Fig. 5. a. Decomposition of a tree T in solid paths. b. A search in the biased binary tree representing the solid path. c. Cut of the node h and the children of the root. d. Resulting trees.

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 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. $\square$

References

1. S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased seach trees. *SIAM J. Comput.*, 14 , number 3:545–568, 1985.
2. 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.
3. 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.
4. 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.
5. 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.
6. K. Douïeb and S. Langerman. Dynamic hotlinks. In *Proc. of the Workshop on Algorithms and Data Structures (WADS 2005)*, volume 3608 of LNCS, pages 271–280, 2005.
7. A. Elmasry. Deterministic jumplists. *Nordic Journal of Computing*, 12:27–39, 2005.
8. S. Fuhrmann, S. O. Krumke, and H.-C. Wirth. Multiple hotlink assignment. In *27th Int. Workshop on Graph-Theoric Concepts in Computer Science*, volume 2204 of LNCS, pages 189–200, 2001.
9. 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.
10. 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.
11. 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.
12. N.Abramson. Information theory and coding. *McGraw Hill*, 1963.
13. M. Perkowitz and O. Etzioni. Towards adaptive Web sites: conceptual framework and case study. *Computer Networks*, 31(11-16):1245–1258, 1999.
14. 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.
15. 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.
16. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–381, 1983.
17. D. D. Sleator and R. E. Tarjan. Self-adjusting binary trees. *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pages 235–245, 1983.