Efficient Algorithms for Hotlinks Assignment

Artur Alves Pessoa
e-mail: artur@inf.puc-rio.br
Eduardo Sany Laber
e-mail: laber@inf.puc-rio.br
Críston Pereira de Souza
e-mail: criston@inf.puc-rio.br
PUC-RioInf.MCC23/03 Julho 2003

Abstract: Let T be a rooted directed tree where nodes represent web pages of a web site and arcs represent hyperlinks. In this case, when a user searches for an information i, it traverses a directed path in T, from the root node to the node that contains i. In this context, hotlinks are defined as additional hyperlinks added to web pages in order to reduce the number of accessed pages per search. In this paper, we address the problem of inserting at most one hotlink in each web page. We consider two objectives: minimizing the number of accesses in a worst case search and minimizing the average number of accesses for a given probability distribution on the leaves. For the worst case problem, we present a (14/3)-approximate algorithm that runs in a $O(n \log m)$ time and requires a linear space, where n and m are the number of nodes (internal and external) and the number of leaves in T, respectively. We also introduce an exact dynamic programming algorithm for solving both problems. For the worst case search problem, it runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space. For the average case search problem it runs in $O(n3^H)$ time and uses $O(n2^H)$ space, where H is the height of T. We point out that our algorithm runs in polynomial time, $O(n^{1+\log_k 3})$, for k-regular trees with arbitrary probability distribution on its leaves, which improves the results of Kranakis et. al. [KKS01].

Keywords: Web Algorithms, Hotlinks Assignment, Approximation Algorithms, Dynamic Programming.

Resumo: Seja T uma árvore direcionada enraizada, onde os nós representam páginas web de um web site, e os arcos representam hyperlinks. Neste caso, quando um usuário busca uma informação i, ele percorre um caminho direcionado em T, partindo do nó raiz até o nó que contêm i. Neste contexto, definimos hotlinks como hyperlinks adicionados às páginas web, de modo a reduzir o número de páginas acessadas nas buscas. Neste trabalho, tratamos o problema de inserir no máximo um hotlinks por página web. Consideramos dois objetivos: minimização do número de acessos no pior caso e minimização do número de acessos no caso médio. Para o problema do pior caso, introduzimos um algoritmo (14/3)-aproximado para o mesmo problema, que executa em tempo $O(n \log m)$ e requer espaço linear, onde n e m são os número de nós (internos e externos) e o número de folhas em T, respectivamente. Também introduzimos um algoritmo exato baseado em programação dinâmica para resolver os dois problemas. Para o problema do pior caso este executa em tempo $O(n(nm)^{2.882})$ e utiliza $O(n(nm)^{1.441})$ bytes de memória. Para o problema do caso médio este executa em $O(n3^H)$ e requer espaço $O(n2^H)$, onde H é a altura da árvorwe T. Observamos que o nosso algoritmo executa em tempo $O(n^{1+\log_k 3})$ para árvores k-regulares com distribuição de probabilidades arbitrárias, melhorando os resultados de Kranakis et. al. [KKS01].

Palavras-chave: Algoritmos para a Web, Atribuição de Hotlinks, Algoritmos Aproximativos, Programação Dinâmica.

1 Introduction

Due the expansion of the Internet at unprecedented rates, continuing efforts are being made in order to improve its performance. An important approach is improving the design of web sites [Dro98, PE99]. A web site can be viewed as a directed graph where nodes represent web pages and arcs represent hyperlinks. In this case, the node that corresponds to the home page is a source node. Hence, when a user searches for an information i in a web site, it traverses a directed path in the corresponding graph, from the source node to the node that contains i. Here, we assume that the user always knows which link leads to the desired information. In this context, hotlinks are defined as additional hyperlinks added to web pages in order to reduce the number of accessed pages per search [PE99].

Since a "nice" web page cannot contain much information, the number of hotlinks inserted in each page should be limited. This scenario motivates the problem of inserting at most one hotlink in each web page, so as to minimize the number of accesses required to locate an information.

Here, we consider the version of the problem in which the given web site is represented by a rooted directed tree T, where only the leaves contain information to be searched by the user. We consider two objectives: minimizing the number of accesses in a worst case search and minimizing the average number of accesses for a given probability distribution on its leaves. We denote these problems by Worst Case Hotlink Search (WCHS) problem and the Average Case Hotlink Search (ACHS) problem, respectively. Following the "obvious navigation assumption" of [CKK+03], we assume that the user always follows a hotlink (u, v) from node u when searching for a leaf in the subtree rooted by v. Due to this assumption, we consider that the insertion of a hyperlink (u, v) must be followed by the deletion of any other arc that ends in v. As a result, the graph obtained after inserting a hotlink in a tree is also a tree. The tree of Figure 1.(b) is obtained from that of Figure 1.(a) through the addition of the hotlink (u, v) and then the addition of hotlink (z, w).

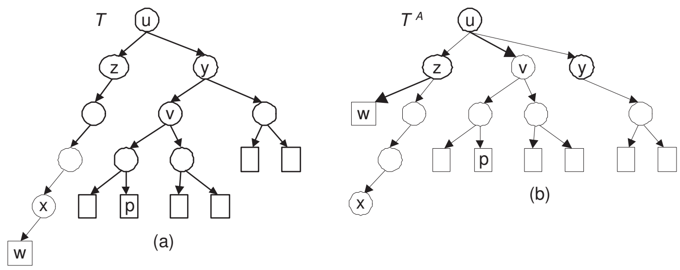

Figure 1: (a) the tree T. (b) the tree T^A, where $A = \{(u, v), (z, w)\}$.

1.1 Problem Definition

An instance of the WCHS problem is defined by a directed tree $T = (V, E)$, rooted at a node $r \in V$. For the ACHS problem, we are also given a probability distribution $\mathbf{p} = [p_1, p_2, \ldots, p_m]$ for the leaves of T. Thus, an instance for ACHS is defined by the pair $(T, \mathbf{p})$.

In order to define the problem, let us recall some basic definitions. The level of a node in a tree is its distance to the root of T. We say that a node v is a descendant of another node u in T when the only path in T that connects r to v contains u. In this case, we also have that u is ancestor of v. A node u is a proper descendant (ancestor) of v if u is a descendant (ancestor) of v and $u \neq v$.

Given $T = (V, E)$, a solution to the ACHS (WCHS) problem is a *hotlink assignment*, defined as a set $A \subset V \times V$. A hotlink assignment A is *feasible* if and only if it satisfies the following three conditions:

(i) for every arc $(u,v) \in A$, v is descendant of u in T;

(ii) Let u,v,a,b be nodes from V such that u is a proper ancestor of v and v is a proper ancestor of a. If $(u,a),(v,b) \in A$, then a is not an ancestor of b;

(iii) for every node $u \in V$, there is at most one arc $(u,v) \in A$.

The condition (ii) is related to the "obvious navigation assumption" and its motivation can be better understood by examining Figure 1. The condition prevents, as an example, that both (u,v) and (y,p) belong to A simultaneously. Recall that we assume that the user always follows a hotlink (u,v) from node u when searching for a leaf in the subtree rooted at v. Since p is a node in such a subtree, we conclude that the hotlink (y,b) (if it exists) will never be followed by the user.

We remark that the definition of a feasible hotlink assignment allows self loops. Although this is not necessary, it helps the description of the algorithms proposed in this paper. For practical purposes, however, we never add a self loop.

Now, we formalize the objectives for both the ACHS and the WCHS problems. For that, we define the concept of an *improved tree.*

Definition 1 *Given $T=(V,E)$, and a feasible hotlink assignment A, the improved tree obtained from T through A is defined as $T^A=(V,(E-X)\cup A)$, where $X=\{(u,v)\in E\ |(y,v)\in A$ for some $y\in V\}$.*

In the definition above, X is the set of arcs of E whose heads receive some hotlink from A. As an example, Figure 1.(a) shows a tree T and Figure 1.(b) shows the tree $T^{\{(u,v),(z,w)\}}$. The set X in this case is $(x,w),(y,v)$.

Given an improved tree T^A rooted at r, let $d_A(u)$ be the level of u in T^A. Moreover, let L be the subset of V that contains all leaves of T. Given $\mathbf{p}=[p_1,p_2,\ldots,p_m]$, an AC-optimal (where AC means average case) hotlink assignment to $(T,\mathbf{p})$ is a feasible hotlink assignment A^* that minimizes

$$E[T^{A^*},\mathbf{p}]=\sum_{x\in L} p_x d_{A^*}(x)$$

over all possible feasible assignments. On the other hand, a WC-optimal (where WC means worst case) hotlink assignment to T is a feasible hotlink assignment A^* that minimizes

$$h(T^{A^*})=\max_{x\in L} d_{A^*}(x),$$

over all possible feasible assignments. The objective of the ACHS (WCHS) problem is to find an AC-optimal (WC-optimal) hotlink assignment to T.

We use $h^*(T)$ and $E^*[T,\mathbf{p}]$, respectively, to denote $h(T^{A^*})$ and $E[T^{A^*},\mathbf{p}]$. Observe that some internal nodes of T may become leaves in T^A (e.g. the node x in Figure 1.(b)). By definition, these nodes do not belong to L. Hence, we refer to the nodes of L as *hotleaves.* The *height* of a tree T', denoted by $H(T')$, is the maximum distance from the root of T' to a leaf in T'. The height takes into account all leaves including those that are not hotleaves. This measure plays an important role in the complexity of our exact algorithm. Throughout this paper, we may use H to denote the height $H(T)$ of the input tree T. In Figure 1.(b), we have $H(T^A)=4$ while $h(T^A)=3$.

1.2 Statement of the Results

Let $n=|V|$ and $m=|L|$. For the WCHS problem, we present a (14/3)-approximation algorithm that runs in a $O(n\log m)$ time and requires a linear space. Maybe the first idea to design an approximation algorithm is to find a node u for which the addition of (r,u) leads to subproblems with approximately the same number of hotleaves. Although this idea is useful to prove an upper bound on $h(T^{A^*})$, it does not lead to a constant factor approximation algorithm. In the same spirit, our algorithm finds a node u for which the addition of (r,u) leads to "balanced" subproblems. However, it uses a lower bound on $h^*(T)$ as a balancing criterion. This lower bound is based on the following two simple ideas:

1. the length of the worst case search is not smaller than $\log_{d+1} m$, where d is the maximum node out-degree in T;
2. removing subtrees from T never increases the length of the worst case search but may increase the previous lower bound, once the node degrees are decreased.

We also introduce a dynamic programming algorithm for the WCHS problem that runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space. This algorithm can be easily adapted to exactly solve ACHS in $O(n3^H)$ time with $O(n2^H)$ space requirement, where H is the height of T.

It is worth mentioning that a straightforward application of the dynamic programming technique to WCHS (ACHS) leads to an exponential time algorithm on the size $|V|$ of the input tree. Here, by using a special decomposition technique we devise an algorithm with the following property: given an integer D, it produces the best possible hotlink assignment A^*_D among the hotlink assignments A such that $H(T^A) \leq D$. This algorithm runs in $O(n3^D)$ time. For exactly solving the WCHS problem, we execute this algorithm with $D = O(\log n)$ since we prove in Section 3 that there is a WC-optimal hotlink assignment A^* such that $H(T^{A^*}) = O(\log n)$. On the other hand, for the ACHS problem this algorithm is executed with $D = H$ since we do not know a better upper bound on $H(T^{A^*_D})$. However, under the assumption that H is logarithmic on n, $k-$regular complete trees as an example, it implies on a polynomial algorithm for ACHS.

1.3 Related Work

Hotlinks were firstly studied by Perkowitz and Etzioni [PE97]. After that, some authors have considered the ACHS problem and its variants [BKK+00, FKW01, KKS01]. The first and the second references address a variation of the problem where the "obvious navigation assumption" is not considered.

In [BKK+00], Bose et. al, give a lower bound of $\frac{Ent(\mathbf{p})}{\log(d+1)}$ on $E^*[T,\mathbf{p}]$, where $Ent(\mathbf{p})$ is the entropy [Abr63] of the access probability distribution $\mathbf{p}$ and d is the maximum node outdegree in the tree T. In [KKS01], Kranakis et al. present an $O(n^2)$ time greedy algorithm whose analysis provides an upper bound of $\frac{Ent(\mathbf{p})}{\log(d+1)-(d/(d+1))\log d} + \frac{d+1}{d}$ on $E^*[T,\mathbf{p}]$. They also show that their algorithm performs close to the optimal for k-regular complete trees.

We note that despite of the research described in these papers, no exact algorithm was previously known even for k-regular trees with arbitrary probability distribution. Since $H = \log n$ in this case, our exact algorithm for ACHS problem runs in $O(n^{1+\log_k 3}) = O(n^{2.59})$ time.

In [FKW01], Fuhrmann et al. considered the version of ACHS where multiple hotlinks can be assigned from a single node. For k-regular complete trees, they proved upper bounds on $E^*[T,\mathbf{p}]$ for general distributions and lower bounds on $E^*[T,\mathbf{p}]$ for uniform distributions.

Bose et al. [BKK+00] considered a variant of the ACHS problem where the input tree is replaced by an arbitrary DAG (Directed Acyclic Graphs). They showed that this problem is $\mathcal{NP}$-complete even for an uniform distribution. For a more detailed discussion of ACHS problem and its variants, including practical aspects, we refer to [CKK+03].

1.4 Paper Organization

This paper is organized as follows. In Section 2, we introduce some additional notations and definitions. In the following section, we prove some properties of a WC-optimal solution for WCHS. Then, in Section 4, we present the (14/3)-approximation algorithm for the WCHS. Finally, in Section 5, we describe an exact polynomial algorithm for both the ACHS and the WCHS.

2 Notations and Definitions

We use $T_u = (V_u, E_u)$ to denote the subgraph of T induced by the descendants of u, that is, $V_u = \{v \in V \mid v \text{ is descendant of } u\}$, and $E_u = \{(v,w) \in E \mid v, w \in V_u\}$. $T - T_u$ is used to denote the subgraph of T induced by $V - V_u$. Throughout this paper, we refer to T_u as the *subtree of T rooted at u*. Furthermore, we use L_u to denote the subset of V_u that contains all leaves of T_u. If $u \in L$ then we say that T_u is a *trivial subtree* of T. Finally, we use S_u to denote the set of all children of node u in T.

3 Structural Properties

In this section, we prove some properties for an WC-optimal hotlink assignment. Throughout this paper, we use $\log x$ to denote $\log_2 x$.

3.1 Upper Bounds

Here, we prove upper bounds on both $h^*(T)$ and $H(T^{A^*})$. For that, we introduce the LOG algorithm for the WCHS problem, which is a quite simple recursive algorithm that starts with the tree T and during its execution it obtains a sequence of improved trees due to the addition of hotlinks. At the end, the tree T^A is obtained, where A is the set of hotlinks added by LOG.

In order to decide the next hotlink to be assigned, LOG maintains a vector $\mathbf{w}$, where $w(v)$, $\forall v \in V$, gives the number of hotleaves that are descendents of v in the current tree. First, LOG assigns a hotlinks to a node u which splits the tree T into "balanced" subtrees. By balanced we mean that none of them has more than $\max\{t, (1-t)\}|L|$ hotleaves, where t is a parameter whose value will be set in the analysis. Then, it updates the vector $\mathbf{w}$ with respect to the improved tree and recursively handles each of these balanced trees. A pseudo-code for LOG is presented in Figure 2. Figure 3 shows the trees T and $T^{(r,u)}$. In $T^{(r,u)}$, the subtrees $T_1 - T_u$, T_2, ..., T_k have at most $(1-t)|L|$ hotleaves while the subtrees rooted at the nodes that are children of u have at most $t|L|$ hotleaves.

```
LOG(T: directed tree)
    If T has exactly one hotleaf
        Add a hotlink from r to this leaf
    Else If T has more than one hotleaf
        Find a node u in V with the property that w(u) ≥ tw(r) and w(v) < tw(r) for every v ∈ S_u. (*)
        Add the hotlink (r,u) to T to obtain an improved tree T^{(r,u)}.
        Update the value of w(v) with respect to T^{(r,u)}, for every ancestor v of u in T.
        Let K = {v|v is a child of r in T^{(r,u)}} and M = {v|v is a child of u in T^{(r,u)}}.
        For every v ∈ (K ∪ M) − {u} do LOG(T_v)
```

Figure 2: The LOG Algorithm

The next lemma, whose proof we defer for the extended version of the paper, guarantees that it is always possible to find a node u as in the line (*) of the pseudo-code.

Lemma 1 *Let T be a tree rooted at r and let t be a real number with $1/|L| \leq t \leq 1$. Then, there is a node $u \in T$ such that $w(u) \geq tw(r)$ and $w(v) < t(w(r)$, for every child v of u.*

For this algorithm, we have the following theorem.

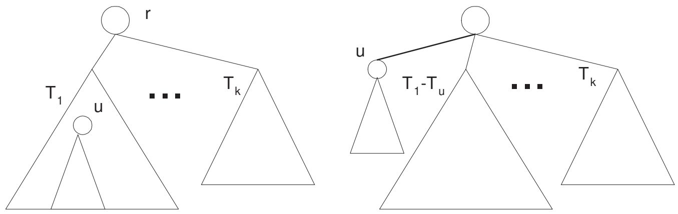

Figure 3: The improved tree obtained by LOG due to the addition of (r, u)

Theorem 1 *Let A be a hotlink assignment obtained by the LOG algorithm, for the input tree T rooted at r. In this case, we always have $h(T^A) \leq c \log w(r) + 1$, where*

$$c = \max\left\{ \frac{2}{\log\left(\frac{1}{t}\right)}, \frac{1}{\log\left(\frac{1}{1-t}\right)} \right\}.$$

Proof: We prove it by induction on the value of $w(r)$. If $w(r) = 1$ then $h(T^A) = 1 = c \log w(r) + 1$. Now, let us assume that the result holds for the case where $w(r) < k$. Under this assumption, we shall prove that it also holds for the case where $w(r) = k$. By the choice of u, we have in $T^{\{(r,u)\}}$: $w(u) \geq tw(r)$, $w(v) < tw(r)$ for every $v \in M$, and $w(v) \leq (1-t)w(r)$ for every child $v \in K - \{u\}$. By applying the inductive hypothesis on the subtrees rooted at the nodes in $(K \cup M) - \{u\}$, we obtain that

$$\begin{aligned} h(T^A) &= \max\{\max\{1 + h(T_v^A) \mid v \in K - \{u\}\}, \max\{2 + h(T_v^A) \mid v \in M\}\} \\ &\leq \max\{1 + c\log((1-t)w(r)) + 1, 2 + c\log(tw(r)) + 1\} \\ &= c\log w(r) + 1 + \max\left\{1 - c\log\left(\frac{1}{1-t}\right), 2 - c\log\left(\frac{1}{t}\right)\right\}. \qquad (1) \end{aligned}$$

We establish this theorem by replacing the second and the third occurrence of c in the righthand side of the previous equality respectively by $1/\log(1/(1-t))$ and $2/\log(1/t)$. ■

From elementary calculus, we obtain that the best upper bound given by the previous theorem is achieved with $t = \frac{3-\sqrt{5}}{2}$. In this case, we have $c = \frac{2}{1-\log(3-\sqrt{5})} < 1.441$, which leads to the following corollary.

Corollary 1 $h^*(T) \leq h(T^A) \leq 1.441 \log m + 1$

Now, let us assume that the LOG algorithm uses the vector $\mathbf{w}$ to maintain the number of nodes in the subtree rooted at each node rather than the number of hotleaves. We refer to this algorithm as the Modified LOG algorithm. In this case, we claim that it constructs a hotlink assignment A such that the height of T^A is not larger than $1.441 \log n + 1$. The proof for this claim is analogous to that of Theorem 1. The next theorem uses this claim to give an upper bound on the minimum height of a tree improved by a WC-optimal hotlink assignment.

Theorem 2 *There is a WC-optimal hotlink assignment A^* such $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$.*

Proof: Let A' be an WC-optimal hotlink assignment for T. We prove this theorem constructing another WC-optimal hotlink assignment A^* from A' such that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$. First, we insert in A^* every hotlink $(u,v) \in A'$ such that u is at a level smaller than $h^*(T)$ in $T^{A'}$. At this point, we already have $h(T^{A^*}) = h^*(T)$. Hence, by corollary 1, we obtain that any non-trivial subtree of T^{A^*} rooted at a level not smaller than $\lfloor 1.441 \log m + 1 \rfloor$ has no hotleaf and no hotlink assigned from. Since any change in such subtrees does not affect the optimality of A^*, we apply the Modified LOG algorithm on every subtree $T_v^{A^*}$ of T^{A^*} such that v is at level $\lfloor 1.441 \log m + 1 \rfloor$. Recall that the Modified LOG algorithm always construct improved trees with heights not greater than $1.441 \log n + 1$. Thus, by inserting the obtained hotlinks in A^*, we obtain that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$, and the proof is done. ■

3.2 Lower Bounds

Here, we prove a simple lower bound on $h^*(T)$. First, observe that the node outdegree in T^{A^*} is at most $d+1$, where d is the maximum node outdegree of T. Then, we have the following lower bound.

Lemma 2 *Let T be a tree with maximum out degree d. Then,* $h^*(T) \geq \log_{d+1} m$.

We say that T_F is a *binary subtree* of T when it is a connected subgraph of T with a maximum out-degree of two. Let $T_F = (V_F, E_F)$ be a binary subtree of T and let m_F be the number of hotleaves in T_F. In this case, we have the following theorem.

Theorem 3 $h^*(T) \geq \log_3 m_F$.

Proof: Applying Lemma 2 to T_F, we get that $h^*(T_F) \geq \log_3 m_F$. Thus, it remains to show that $h^*(T) \geq h^*(T_F)$. Let A be a WC-optimal hotlink assignment for T and let $A_F = \{(u,v) \in A \mid u, v \in V_F\}$. For every node $i \in V_F$, we have that $d_{A_F}(i)$ in $T_F^{A_F}$ is the same as $d_A(i)$ in T^A. Hence, we obtain that $h(T^A) \geq h(T^{A_F})$, which completes our proof. ■

4 An Approximation Algorithm

In this section, we describe a (14/3)-approximation algorithm for the WCHS problem that executes in $O(n \log m)$. Let us refer to this algorithm as the ALOG (Approximation LOG) algorithm. ALOG is similar to LOG except for the information stored by vector $\mathbf{w}$. Here, $w(v)$ stores the number of hotleaves of the binary subtree rooted at v which contains the largest number of hotleaves. Note that if v has only one child a, then $w(v) = w(a)$. Otherwise, $w(v) = w(a) + w(b)$ where a and b are the children of v with maximum $\mathbf{w}$'s value, that is, $w(v) = \max_{\{(a,b) \in S_v \times S_v \mid a \neq b\}} \{w(a) + w(b)\}$.

An efficient way to update the vector $\mathbf{w}$ when we improve a tree is discussed in the proof of Theorem 5.

Now, we analyze the approximation ratio of ALOG. For that, we use $w(v)$ to denote the value of $w(v)$ with respect to T. As a consequence of Theorem 3 and the definition of $\mathbf{w}$, we have that

$$h^*(T) \geq \log_3 w(r). \tag{2}$$

Now, we develop an upper bound on $h(T^A)$. First, we must understand how $\mathbf{w}$ evolves when the tree T is improved to become the tree $T^{\{(r,u)\}}$. The next lemma address this issue. For every $v \in V$, let $w^{T^{\{(r,u)\}}}(v)$ be the value of $w(v)$ with respect to $T^{\{(r,u)\}}$.

Lemma 3 *Let K and M as in Step 4 of LOG pseudo-code, and let s_1 be the node of K that is a proper ancestor of u in T (if it exists). Let also $t \geq 0.5$. Then, we have*

(i) $w^{T^{\{(r,u)\}}}(v) \leq (1-t)w(r), \forall v \in K - \{u, s_1\}$

(ii) $w^{T^{\{(r,u)\}}}(v) < tw(r), \forall v \in M$

(iii) $w^{T^{\{(r,u)\}}}(s_1) \leq 2(1-t)w(r)$

Proof: See appendix A ■

The following lemma gives an upper bound on the $h(T^A)$.

Lemma 4 *Let A be a hotlink assignment obtained by the ALOG algorithm, for the input tree T rooted at r. In this case, we always have $h(T^A) \leq c \log w(r) + 1$, where*

$$c = \max\left\{ \frac{2}{\log\left(\frac{1}{t}\right)}, \frac{1}{\log\left(\frac{1}{2(1-t)}\right)} \right\}.$$

Proof: The proof is analogous to that of Theorem 1. The only difference is that we use the upper bound given by Lemma 4 for the child of r that is ancestor of u, when the hotlink (r, u) is assigned. ■

Again, we obtain from elementary calculus that the optimal value of t is $\frac{9-\sqrt{17}}{8}$ (observe that $t > 0.5$ as required in Lemma 3). In this case, we have $c = \frac{2}{3-\log(9-\sqrt{17})} < 2.801$, which leads to the following theorem.

Theorem 4 *The ALOG algorithm is $(14/3)$-approximate.*

Proof: If $w(r) = 1$ then ALOG always constructs an optimal hotlink assignment. Hence, let us assume that $w(r) > 1$. It follows from equation (2) and Lemma 4 that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{c \log w(r) + 1}{\log_3 w(r)}. \tag{3}$$

However, since the objective function for the WCHS problem is always integer, we also have that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{\lfloor c \log w(r) + 1 \rfloor}{\lceil \log_3 w(r) \rceil}. \tag{4}$$

We establish this theorem by using (4) whenever $w(r) \leq 125$ and (3) otherwise. In the first case, it can be verified that this theorem holds for all possible values of $w(r)$. In the second case, we have that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{c \log w(r)}{\log_3 w(r)} + \frac{1}{\log_3 126} < 14/3.$$

■

Now, we analyze the running time of the ALOG algorithm.

Theorem 5 *The ALOG algorithm runs in $O(n \log m)$ time.*

Proof: See appendix B. ■

5 Exact Algorithms

In this section, we introduce the PATH algorithm, an exact dynamic programming based algorithm for solving hotlinks assignment problems. It can be adapted to either solve WCHS or ACHS. For the sake of simplicity, we describe only the variation of PATH algorithm that solves the WCHS problem. At the end of this section, we briefly discuss how to modify this algorithm to solve ACHS. Given a parameter D selected by the user, PATH finds a hotlink assignment A^*_D with the following properties: $H(T^{A^*_D}) \leq D$ and for every feasible hotlink assignment A, with $H(T^A) \leq D$, we have $h(T^{A^*_D}) \leq h(T^A)$.

Observe that the WCHS problem could be exactly solved as follows. For each possible hotlink assignment (r, u) from r, obtain the improved tree $T^{\{(r,u)\}}$. Then, for each child v of r in $T^{\{(r,u)\}}$, recursively solve the subproblem where the input tree is the subtree rooted at v. At the end, return the best solution found. This approach, however, generates an exponential number of subproblems.

In order to avoid this undesirable behavior, our approach consists in determining which child v of r is an ancestor of u (the node which receives the hotlink in the optimal solution) rather than deciding which node is u. In other words, the hotlink from r is available only to one subtree T_v, for $v \in S_r$.

In order to generalize the previous idea, we define the PATH-WCHS problem (P-WCHS for shortness).

Input:

i) a directed path $\mathbf{q} = (V_\mathbf{q}, E_\mathbf{q})$ where $V_\mathbf{q} = \{q_1, \ldots, q_k\}$ and $E_\mathbf{q} = \{(q_i, q_{i+1}) | 1 \leq i \leq k-1\}$

ii) a vector $\mathbf{a} = (a_1, ..., a_k, a_{k+1}, b) \in \{0, 1\}^{k+2}$;

iii) a tree $T^s = (V^s, E^s)$ rooted at r.

iv) an integer D

Output: A hotlink assignment A to the tree $T_\mathbf{q} = (V_\mathbf{q} \cup V^s,\ E_\mathbf{q} \cup E^s \cup \{(q_k, r)\})$, satisfying the following five conditions:

(a) A is feasible in the sense of the WCHS problem;

(b) No hotlink can point to a node in V_p;

(c) If $a_i = 0$, then no hotlink can leave q_i;

(d) If $a_{k+1} = 0$, then no hotlink can leave r;

(e) if $b = 0$, then no hotlink can point to r.

(f) $H(T^A_\mathbf{q}) \leq D$

Objective: Minimize the maximum level of a hotleaf in the resulting improved tree $T^A_\mathbf{q}$.

Observe that the WCHS problem is a particular case of the P-WCHS problem when $\mathbf{q}$ is empty, $b = a_1 = 1$, and $D = H$. Thus, an exact algorithm for P-WCHS is also an exact algorithm for WCHS.

5.1 Solving P-WCHS

Figure 4 is used throughout this section to illustrate some ideas. Figure 4.(a) presents an instance of P-WCHS where the path $\mathbf{q}$ consists of four nodes q_1, q_2, q_3 and q_4. If a node $q_i \in \mathbf{q}$ is such that $a_i = 1$, then q_i is said to be *available*. The only non-available node, q_3, is black colored. The subtree T^s is rooted at a node r which has three children. Since $a_5 = b = 1$, hotlinks can be assigned from and to the node r.

We need to introduce the following conventions: given a binary vector $\mathbf{c}$ and a binary string d, we use $\mathbf{c}d$ to indicate the vector obtained by concatenating $\mathbf{c}$ and d. For example, if $\mathbf{c} = (0,1)$ and $d = 100$, then $\mathbf{c}d = (0,1,1,0,0)$ and $d\mathbf{c} = (1,0,0,0,1)$. We use $\mathbf{c}_i$ to denote the vector obtained by truncating the ith first components of $\mathbf{c}$. At the previous example, $\mathbf{c}_1 = (0)$ and $\mathbf{c}_2 = (0,1)$. For a directed path $\mathbf{q}$ and a node u, we use $\mathbf{q} \to u$ to indicate the path obtained by inserting u at the end of $\mathbf{q}$. We use $\mathbf{q}_i$ to denote the subpath of $\mathbf{q}$ formed by its i-th first nodes.

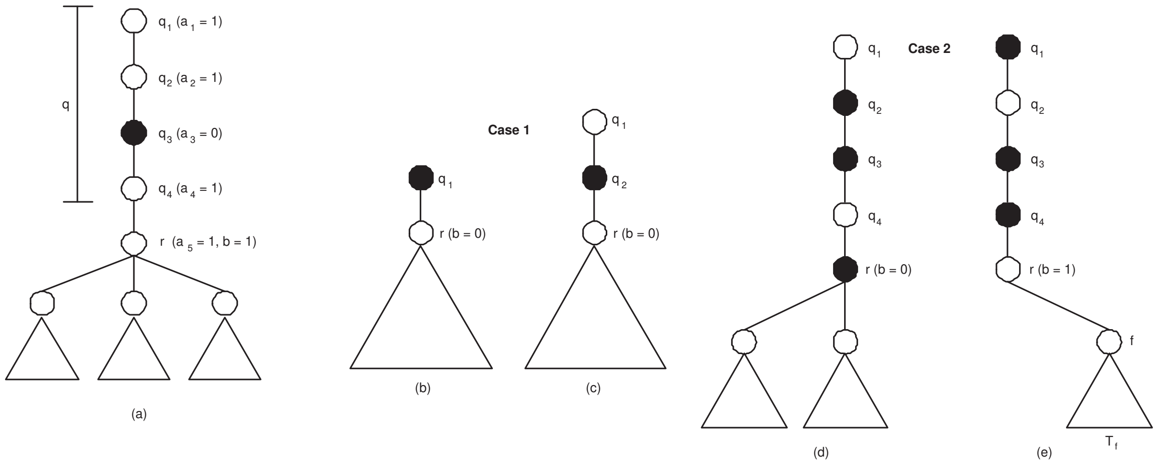

Figure 4: (a) an instance of the P-WCHS problem. (b) and (c) two possible subproblems generated in the case 1. (d) and (e) the decomposition in the case 2 when $c = (0,1,0,0,1)$.

Let $h^*(\mathbf{q}, \mathbf{a}, T^s)$ denote the cost of an optimal solution of a P-WCHS instance defined by a binary vector $\mathbf{a}$, a directed tree T^s, a path $\mathbf{q}$ with $|\mathbf{a}| - 2$ nodes and an integer D. If $|\mathbf{q}| > D$, PATH set $h^*(\mathbf{q}, \mathbf{a}, T^s) = \infty$. Hence, let us assume that $|\mathbf{q}| \leq D$. In order to solve this instance we must consider the following cases:

Case 1: some hotlink is assigned from a node of $\mathbf{q}$ to r in the optimal solution;

Case 2: no hotlink is assigned from a node of $\mathbf{q}$ to r in the optimal solution;

Case 1 This case is only considered when $b = 1$. In this case, we must add a hotlink from some available node to r. Thus, we have $\sum_{i=1}^{k} a_i$ possibilities. If (q_1, r) is assigned to the tree of Figure 4.(a), then PATH generates the subproblem of Figure 4.(b). In fact, the addition of hotlink (q_1, r) creates an improved tree where q_1 has two children: T^s and the path (q_2, q_3, q_4). However, since q_2, q_3 and q_4 are not ancestors of hotleaves, they can be removed without modifying the cost of the solution. Observe that b is set to 0 since the condition (ii) of the WCHS problem definition (see Section 1.1) assures that no node can receives two hotlinks. When (q_2, r) is assigned, instead of (q_1, r), then PATH generates the subproblem of Figure 4.(c). In general, if some hotlink points to r in the optimal solution, we have that

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \min_{i \in \{1,2,3,\ldots,k\} \ | \ a_i = 1} \{h^*(\mathbf{q}_i, \mathbf{a}_{i-1} 0 a_{k+1} 0), T^s)\} \qquad (5)$$

Case 2 In this case all the available nodes of $\mathbf{q}$ may only point to some node in $V_r - \{r\}$. Thus, PATH must decide which of the available nodes are allowed to point to the nodes of T_f, the subtree rooted at the last child f of r (assuming any order). Let $k' = \sum_{i=1}^{k+1} a_i$ be the number of available nodes.

Then, PATH has $2^{k'}$ possibilities to take such a decision. Since it is not clear which one is the best, then all of them are considered. In order to clarify this idea, let us consider the possibility where q_2 and r remain available for T_f (see Figure 4.(e)). As a consequence, only q_1 and q_4 will be allowed to point to nodes in $T^s - T_f$ (Figure 4.(d)). Figure 4.(d) defines a new subproblem ($\mathbf{q}$,$\mathbf{a}'$,$T^s - T_f$), where $\mathbf{a}' = (1, 0, 0, 1, 0, 0)$. Note that b is set to 0 since we are in case 2. On the other hand, Figure 4.(e) defines a new subproblem ($\mathbf{q} \to r$,$\mathbf{a}''$,T_f), where $\mathbf{a}'' = (0, 1, 0, 0, 1, 1, 1)$.

Thus, the maximum between the cost of the optimal solutions for the subproblems defined by Figures 4.(d) and 4.(e) is the cost of the optimal solution for the problem of Figure 4.(a) under the assumptions that no hotlink can be assigned to r (Case 2), the nodes q_2 and r cannot point to nodes in $T^s - T_f$, and the nodes q_1 and q_4 cannot point to nodes in T_f.

In general, let C be a binary vector defined by $C = \{(c_1, \ldots, c_{k+1}) | c_i \leq a_i \text{ for } i = 1, \ldots, k+1\}$. Each $\mathbf{c} \in C$ corresponds to one of the $2^{k'}$ possibilities for selecting the nodes that will remain available to point to nodes in T_f. Furthermore, let $\overline{\mathbf{c}} = \mathbf{a} - \mathbf{c}$. This vector defines which nodes from $\mathbf{q}$ will remain available to point to nodes in $T^s - T_f$. Then, by considering all choices for c, we have that

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \min_{\mathbf{c} \in C} \{\max\{h^*(\mathbf{q} \to r, \mathbf{c}11, T_f), h^*(\mathbf{q}, \overline{\mathbf{c}}0, T^s - T_f)\}\}, \qquad (6)$$

under the assumption that no node from $\mathbf{q}$ points to r.

Cases 1 and 2 together: Let P and Q be respectively, the righthand side of equations (5) and (6). Thus,

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \begin{cases} \text{Q if } b = 0 \\ \min\{P, Q\} \text{ if } b = 1 \end{cases}$$

Stop Conditions: If T^s has only one node and this node is a hotleaf, say l, then the best thing to do is to assign a hotlink from the first available node in $\mathbf{q}$ to l. Thus,

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \begin{cases} \min\{i | 1 \leq i \leq k \text{ and } a_i = 1\} \text{ if } \mathbf{q} \text{ has some available node} \\ k, \text{ otherwise} \end{cases} \qquad (7)$$

If T^s has no hotleaves then $h^*(\mathbf{q}, \mathbf{a}, T^s) = 0$.

5.2 Analysis

In order to prove that the algorithm is correct, we must argue that if A is a feasible hotlink assignment with $H(T^A) \leq D$, then A is among the assignments considered by the PATH algorithm. Since PATH only discards subproblems for which $|\mathbf{q}| > D$, we must argue that none of the subproblems generated by PATH to produce the assignment A has this property. Let $\mathbf{I}' = (\mathbf{q}', \mathbf{a}', T')$ be one of these subproblems. By observing that $\mathbf{q}'$ is also a path in T^A, we conclude that $|\mathbf{q}| \leq D$.

Now, we analyze both the time and space complexities of the PATH algorithm.

Theorem 6 *The PATH algorithm runs in $O(n3^D)$ time and uses $O(n2^D)$ space.*

Proof: First, let us analyze the number of generated subproblems. By means of the dynamic programming technique, this number has the same order as the PATH space usage. For each node $u \in V$ and each integer $q \in \{0, 1, \ldots, |S_u| - 1\}$, PATH generates a subtree T^s by removing from T_u all the subtrees rooted at the last q children of u. Now, let r be the root of T. Observe that PATH generates exactly m trivial subtrees. Moreover, each node $v \in V - \{r\}$ is the last (non-removed) child of the root in exactly one generated non-trivial subtree. Hence, PATH generates $m + n - 1 = O(n)$ subtrees. For each generated subtree, PATH generates all possible paths vectors $\mathbf{a}$. Since we have exactly 2^{i+2} possible vectors $\mathbf{a}$ corresponding to a path $\mathbf{q}$ with i nodes, the number of generated vectors is given by $\sum_{i=0}^{D} 2^{i+2} = O(2^D)$.

As a result, we have that PATH uses $O(n2^D)$ space. Finally, we obtain the time complexity of PATH by counting the number of subproblems checked to calculate each value of $h^*(\mathbf{q}, \mathbf{a}, T^s)$ (or another function with the same arguments, in the case of ACHS). Let r^s be the root of T^s. In the Case 1, PATH checks $O(D)$ subproblems, since we have $O(D)$ possible hotlinks from a node in $\mathbf{q}$ to r^s. On the other hand, in Case 2, the number of subproblems checked by PATH depends on the number of available hotlinks in both $\mathbf{q}$ and r^s. For j available hotlinks, $O(2^j)$ subproblems are checked, since this is the complexity of the number of possible distributions for these hotlinks between two subproblems. Moreover, we have $n\binom{D+2}{j}$ subproblems with j available hotlinks, for $j = 0, 1, \ldots, D + 2$. Hence, we have that the time complexity of PATH is given by

$$O(n2^D D) + \sum_{j=0}^{D+2} O\left(n\binom{D+2}{j}2^j\right) = O(n2^D D) + O(n(2+1)^{D+2}) = O(n3^D). \blacksquare$$

Corollary 2 *For the WCHS problem, the PATH algorithm runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space.*

Proof: Theorem 2 assures that there is an optimal solution for WCHS with height at most $1.441(\log m + \log n) + 2$. This assures that PATH can be executed with $D = 1.441(\log m + \log n) + 2$, which implies on the complexities above. ■

5.3 The ACHS Problem

In order to solve the ACHS problem, we must execute the PATH algorithm described above with the following modifications: replace $\max\{h^*(\mathbf{q} \to r, \mathbf{c}11, T_f), h^*(\mathbf{q}, \overline{\mathbf{c}}0, T^s - T_f)\}$ in expression (6) by $h^*(\mathbf{q} \to r, \mathbf{c}11, T_f) + h^*(\mathbf{q}, \overline{\mathbf{c}}0, T^s - T_f)$. Furthermore, the condition given by expression (7) must be modified to

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \begin{cases} \min\{i | 1 \le i \le k \text{ and } a_i = 1\} p_l, & \text{if } \mathbf{q} \text{ has some available node} \\ p_l k, & \text{otherwise} \end{cases}$$

where p_l is the probability of the leaf l in T_s.

References

[Abr63] N. Abramson. *Information Theory and Coding*. McGraw Hill, 1963.

[BKK+00] Prosenjit Bose, Evangelos Kranakis, Danny Krizanc, Miguel Vargas Martin, Jurek Czyzowicz, Andrzej Pelc, and Leszek Gasieniec. Strategies for hotlink assignments. In *International Symposium on Algorithms and Computation*, pages 23–34, 2000.

[CKK+03] J. Czyzowicz, E. Kranakis, D. Krizanc, A. Pelc, and M. Vargas Martin. Enhancing hyperlink structure for improving web performance. *Journal of Web Engineering*, 1(2):93–127, March 2003.

[Dro98] M. C. Drott. Using web server logs to improve site design. In *Proceedings of ACM Conference of Computer Documentation*, pages 43–50, 1998.

[FKW01] Sven Fuhrmann, Sven Oliver Krumke, and Hans-Christoph Wirth. Multiple hotlink assignment. In *Proceedings of the Twenty-Seventh International Workshop on Graph- Theoretic Concepts in Computer Science*, 2001.

[KKS01] Evangelos Kranakis, Danny Krizanc, and Sunil Shende. Approximate hotlink assignment. *Lecture Notes in Computer Science*, 2223:756–767, 2001.

[PE97] Mike Perkowitz and Oren Etzioni. Adaptive web sites: an AI challenge. In *IJCAI (1)*, pages 16–23, 1997.

[PE99] Mike Perkowitz and Oren Etzioni. Towards adaptive Web sites: conceptual framework and case study. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1245–1258, 1999.

A Proof of Lemma 3

First, we prove (i). Since s_1 is an ancestor of u in T, we have that $w(s_1) \geq w(u) \geq tw(r)$. Let $v \in K - \{s_1\}$. Since $w(r) \geq w(s_1) + w(v)$, it follows that $w(v) \leq (1-t)w(r)$. Once the assignment of the hotlink (r, u) does not modify T_v, we have $w^{T^{\{(r,u)\}}}(v) = w(v) \leq (1-t)w(r)$

Now, we prove (ii). First, note that $M = S_u$ since T_u is not modified due to the addition of hotlink (r, u). Let $v \in M$. By the choice of u, $w(v) < tw(r)$. The result follows from the fact that $w(v) = w^{T^{\{(r,u)\}}}(v)$.

Finally, we prove (iii). For that, let us consider the updates performed in the Step 3 of LOG pseudo-code. Figure 5 represents the current subtree T' before these updates. In this figure, we use s_i to denote the ancestor of node u at level i in T, for $i = 0, 1, \ldots, h$, where h is the level of u itself. We also use s'_i to denote a sibling of s_i in T that maximizes $\mathbf{w}$, for $i = 1, 2, \ldots, h$.

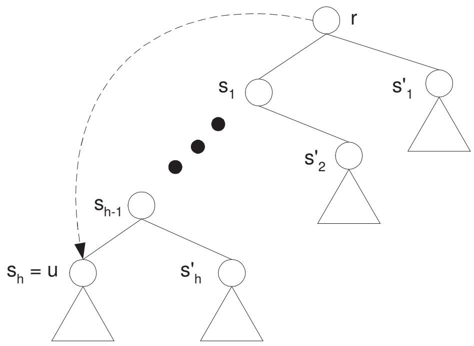

Figure 5: The current subtree before the updates that precede the hotlink assignment (r, u).

By the choice of $u = s_h$, we have that $w(s_h) \geq tw(r)$. As a result, we also have $w(s_i) \geq tw(r)$, for $i = 1, 2, \ldots, h-1$. Since $t \geq 0.5$, we obtain that $w(s_i)$ is maximum among all the children of s_{i-1} in T. Hence, we have $w(s_i) = w(s_{i+1}) + w(s'_{i+1})$, for $i = 0, 1, \ldots, h-1$.

By Figure 5, it is clear that $w(s_1) = \sum_{i=2}^{h} w(s'_i) + w(u)$.

Since $w(s_1) \leq w(r)$, and $w(u) \geq tw(r)$, we obtain that $\sum_{i=2}^{h} w(s'_i) \leq (1-t)w(r)$.

Due to the deletion of u, we may have $w^{T^{\{(r,u)\}}}(s_i) < w(s_i)$. Hence, $w^{T^{\{(r,u)\}}}(s_i)$ is not necessarily maximum among all the children of s_{i-1} in $T^{\{(r,u)\}}$. Let k be the smallest index greater than one such that $w^{T^{\{(r,u)\}}}(s'_k) > w^{T^{\{(r,u)\}}}(s_k)$. If no such index exists, then let $k = h$. In this case, we obtain that

$$\sum_{i=2}^{k} w^{T^{\{(r,u)\}}}(s'_i) = \sum_{i=2}^{k} w(s'_i) \leq (1-t)w(r).$$

As a result, we conclude that

$$w^{T^{\{(r,u)\}}}(s_1) \leq w^{T^{\{(r,u)\}}}(s'_k) + \sum_{i=2}^{k} w^{T^{\{(r,u)\}}}(s'_i) \leq 2(1-t)w(r).$$

B Proof of Theorem 5

Here, we analyze an implementation of ALOG that calculates $w(u)$ for every node u of the current tree, whenever executing Step 3 of the LOG pseudocode.

First, we point out that the ALOG calculates the initial value of $w(u)$ for all $u \in V$ in $O(n)$ time, through an DFS. Now, let us analyze the LOG pseudocode. For that, we consider the total time spent at each node $u \in V$. For a given call to the procedure IMPROVE, it can be shown that LOG spends $O(1)$ time at each node of the current tree. Hence, we obtain that LOG runs in $O(n \max\{q(u) \mid u \in V\})$ time, where $q(u)$ is the number of calls to IMPROVE such that u is a node of the current tree.

Next, we prove that $q(u) = O(\log m)$ for all $u \in V$. For that, let us use r_i to denote the root of current tree at the ith call to the procedure IMPROVE. Moreover, let $i_1, i_2, \ldots, i_{q(u)}$ be the indices such that the node u is descendant of r_{i_j}, for $j = 1, 2, \ldots, q(u)$. Given a node $u \in V$, we have the following three facts:

(i) $r_{i_1} \neq r_{i_2} \neq \cdots \neq r_{i_{q(u)}}$;

(ii) r_{i_j} is a proper ancestor of u in the final improved tree T^A, for $j = 1, 2, \ldots, q(u)$;

(iii) $w(r_{i_j}) > 0$, for $j = 1, 2, \ldots, q(u)$.

It is clear that Fact (i) is true, by definition of r_{i_j}. Fact (ii) is true because the hotlinks assigned after r_{i_j} is the root of the current tree does not change the set of descendants of r_{i_j}. Moreover, Fact (iii) is true because the procedure IMPROVE is not called when the $\mathbf{w}$'s value is zero for the root of the current tree. Due to Fact (iii), Corollary 1, and Theorem 4, we obtain that r_{i_j} is at a level not greater than $(14/3)(1.441 \log m + 1) = O(\log m)$, for $j = 1, 2, \ldots, q(u)$. Hence, we conclude that $q(u) = O(\log m)$. As a result, the total time spent by LOG is $O(n \log m)$, and the proof is done.