# Efficient Implementation of a Hotlink Assignment Algorithm for Web Sites

Eduardo Sany Laber
e-mail: laber@inf.puc-rio.br

Artur Alves Pessoa
e-mail: artur@inf.puc-rio.br

Críston Pereira de Souza
e-mail: criston@inf.puc-rio.br

PUC-RioInf.MCC 41/03 Outubro 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, we define hotlinks as additional hyperlinks added to web pages in order to reduce the number of accessed pages per search. Given a search probability for web page, we address the problem of inserting at most one hotlink per page, minimizing the expected number of accesses in a search. In a previous work, we proposed a dynamic programming polynomial time algorithm for this problem, assuming that the height of $T$ is logarithmic. In this paper, we present an efficient implementation for the previous algorithm that leads to optimal or quasi-optimal hotlinks assignments. We also describe experiments with 1,914 trees extracted from 21 actual web sites and randomly generated access probabilities. Our implementation has found optimal solutions to all but two generated instances in up to five minutes each, using a standard computer. The largest solved instance has 57,877 nodes.

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

Resumo: Seja $T$ uma árvore direcionada enraizada onde os nós representam as páginas 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 a informação $i$. Neste contexto, definimos hotlinks como hyperlinks adicionados em páginas web de modo a reduzir o número de páginas acessadas por busca. Dada uma probabilidades de busca para cada páginas web, consideramos o problema de inserir no máximo um hotlink por página, minimizando o número esperado de acessos em uma busca. Em um trabalho anterior, proposemos um algoritmo de tempo polinomial, baseado em programação dinâmica, que obtêm uma atribuição de hotlinks ótima ou quase ótima. Descrevemos também experimentos com 1.914 árvores extraídas de 21 web sites reais, e distribuição de probabilidades gerada aleatoriamente. Em apenas duas instâncias nossa implementação não encontrou uma solução ótima, executando em um computador convencional cada instância em até cinco minutos. A maior instância resolvida tinha 57.877 nós.

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 root 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, we define hotlinks 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. Given a search probability for each web page, we address the problem of inserting at most one hotlink per page, minimizing the expected number of accesses in a search. We call this problem the Average Case Hotlink Search (ACHS) problem. We assume that 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 also assume that the user always follows a hotlink $(u, v)$ from node $u$ when searching for a leaf in the subtree rooted by $v$. This assumption was introduced by Czyzowicz et al. [CKK$^+$03], and called the "obvious navigation" assumption.

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 $(z, w)$.

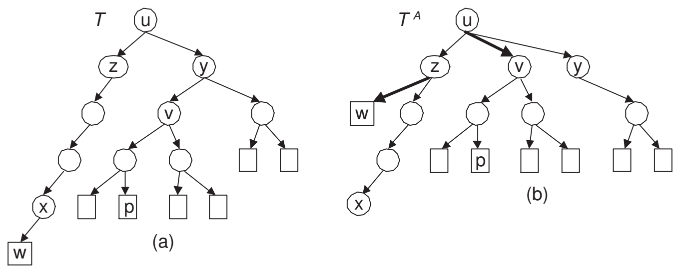


Figure 1: (a) the tree $T$. (b) the tree $T^A$, where $A = \{(u, v), (z, w)\}$.

## 1.1 Problem Definition

An instance of the ACHS problem is a pair $(T, \mathbf{p})$, where $T = (V, E)$ is a directed tree rooted at a node $r \in V$ and $\mathbf{p} = [p_1, p_2, \ldots, p_m]$ is a probability distribution for the leaves of $T$.

In order to explain the problem, let us recall some basic definitions. The level $d(u)$ of a node $u$ in a tree $T$ 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 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 of $V$ 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 objective for the ACHS problem. 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 $X = \{(u, v) \in E \mid (y, v) \in A$ 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 optimal 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. The objective of the ACHS problem is to find an optimal hotlink assignment to $T$.

We use $E^*[T, \mathbf{p}]$ to denote $E[T^{A^*}, \mathbf{p}]$. 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 $T'$. 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 $H(T^A) = 4$.

## 1.2 Statement of the Results

In this paper, we present EX-PATH (experimental PATH) an efficient implementation for the PATH algorithm [PLdS03]. PATH is an exact algorithm for ACHS that has the following property: given a parameter $D$, 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 and requires $O(2^D)$ memory space. Since $H(T^{A^*}) \leq H$ it suffices to execute PATH with $D = H$ in order to solve ACHS. It is worth mentioning that PATH runs in polynomial time, under the assumption that $H(T) = O(\log n)$.

The main motivation for this implementation is our intuition that $H(T)$ typically grows very slowly as $n$ increases in actual web sites. This is based on the belief that a web designer avoids to construct sites that obligates an user to traverse a long path in order to locate its target information. In fact, our intuition is confirmed by the experiments reported in this paper.

We describe experiments with 1914 trees extracted from 21 actual web sites (we found 16 as a maximum value of $H$). Motivated by the research of [Gla94], the access probabilities were generated according to the Zipf distribution. EX-PATH has found optimal hotlinks assignments to 1912 generated instances in up to five minutes each, using a standard computer. Furthermore, approximately 98% of them completed in up to 0.3 seconds. The largest solved instance has 57877 nodes. EX-PATH has the nice property of producing quasi-optimal solutions when the parameter $D$ is set slightly smaller than $H$. In this case, its performance is greatly improved. Furthermore, EX-PATH introduces two practical improvements:

(i) it discards some subproblems generated by PATH without affecting its optimality;

(ii) it uses a weaker restriction as a function of the parameter $D$.

The first improvement reduces both the execution time and the memory usage of EX-PATH. As a result of the second improvement, EX-PATH explores a larger number of feasible solutions in order to find the best one, for the same value of $D$. As a consequence, the quality of the obtained solution is improved without increasing the memory usage and with negligible changes on the execution times. Later, we detail the current $D$-restriction of EX-PATH.

Finally, we also compare the solutions obtained by our implementation with three other implementations for ACHS algorithms: greedyBFS [CKK+03], *approximateHotlinkAssignment* algorithm [KKS01] and MIP. The latter is the implementation of a Mixed Integer Programming Model introduced in this paper. As far as we know, we are the first to show how the solutions of these algorithms compare to the optimal solutions.

### 1.3 Related Work

The idea of adding hotlinks to web sites were firstly proposed by Perkowitz and Etzioni [PE97]. After that, some authors have considered the ACHS problem and some variations [BKK+00, KKS01, FKW01].

Bose et al. [BKK+00] considered a variation of the ACHS problem where the input tree is replaced by an arbitrary DAG (Directed Acyclic Graph). Even for an uniform distribution, they showed that this problem is $\mathcal{NP}$-complete. Furthermore, for general trees, they give a lower bound on optimal value of $\frac{E(p)}{\log(d+1)}$, where $E(\mathbf{p})$ is the entropy [Abr63] of the access probability distribution $\mathbf{p}$ and $d$ is the maximum node outdegree in the tree. This lower bound also holds for the ACHS problem. In [FKW01], Fuhrmann et al. considered another variation of ACHS where multiple hotlinks can be assigned from a single node. For $k$-regular complete trees and general distributions, they proved upper and lower bounds on the optimal values. The previous related works do not use the obvious navigation assumption.

In [CKK+03], Czyzowicz et al. gives a detailed discussion on the ACHS problem and some variants, including practical aspects. The authors also propose a greedy heuristic to the ACHS problem called *greedyBFS*, which is implemented in a software tool. The greedyBFS heuristic was tested with both random and actual instances, where the latter were extracted from the web sites of canadian universities. In [KKS01], Kranakis et al. present an $O(n^2)$ time algorithm to this problem that provides an upper bound on the optimal value of $\frac{E(p)}{\log(d+1)-(d/(d+1))\log d} + \frac{d+1}{d}$.

### 1.4 Paper Organization

This paper is organized as follows. In Section 3, we give a detailed description of the PATH algorithm as well as a brief description of the other algorithms used in our experiments. In Section 4, we introduce the EX-PATH implementation by adding two practical improvements to the PATH algorithm. In Section 5, we describe our experiments and comment our experimental results.

## 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 $T_u$. We use $S_u$ to denote the set of all children of node $u$ in $T$.

## 3 Algorithms

In this section, we explain the algorithms that will be used in our experiments.

### 3.1 Approximation Algorithms

Algorithm *greedyBFS* was designed by Czyzowicz et al. [CKK+03]. The authors defined the *gain* of a hotlink $(u, v)$ as $g(u, v) = p(T_v)(d(v) - d(u) - 1)$, where $p(T_v)$ is the sum of the probabilities of the leaves in $T_v$. First, greedyBFS assigns a hotlink from the root $r$ of the input tree to the node $u$ that maximizes the gain $g(r, u)$. This generates a new tree $T^{(r,u)}$. Then, the algorithm recursively assigns hotlinks to the subtrees rooted at the children of $r$ in $T^{(r,u)}$.

Algorithm *approximateHotlinkAssignment* (AHA for shortness) was proposed by Kranakis et al [KKS01]. The only difference between this algorithm and the previous one is the criterion used to choose the node $u$. AHA chooses a node such that $\frac{p_r}{d+1} \leq p_u \leq \frac{d.p_r}{d+1}$. If such a node does not exist or is a child of $r$, then AHA chooses the grandchild of $r$ with the greatest probability.

### 3.2 The Mixed Integer Programming Algorithm

In order to model ACHS as a MIP problem, we consider the $m$ paths $(\lambda_1, \ldots, \lambda_m)$ in the input tree $T$ from a hotleaf to the root $r$. We denote by $c_k$ and $p_k$ the length of the path $\lambda_k$ and the probability of the only hotleaf in this path, respectively.

In our MIP model, we use the binary variable $x_{i,j}$ to indicate whether the hotlink $(i, j)$ belongs to current solution $(x_{i,j} = 1)$ or not $(x_{i,j} = 0)$.

We say that two hotlinks $(i, j)$ and $(a, b)$ are *nested* when there is at least one path that contains $i, j, a, b$, $i \neq a \vee b \neq j$, and $d(i) \leq d(a) < d(b) \leq d(j)$. Hence, we also use a continuous variable $y^k_{i,j}$, that represents the length reduction on the path $\lambda_k$ due to the addition of the hotlink $(i, j)$. Clearly, if two hotlinks are nested, then only the path length reduction due to the most external hotlink should be non-zero. Moreover, any path length reduction due to the addition of the hotlink $(i, j)$ is not greater than $d(i, j) = (d(j) - d(i) - 1)$. Hence, we have the following two restrictions in our model:

$y^k_{a,b} \leq (1 - x_{i,j})\ d(a, b)$, if $(i, j)$ and $(a, b)$ are nested;
$y^k_{a,b} \leq d(a, b)\ x_{a,b}$, for all $(a, b)$.

We say that two hotlinks $(i, j)$ and $(a, b)$ cross when there is at least one path that contains $i, j, a, b$, and $d(i) < d(a) < d(j) < d(b)$. Since the obvious navigation assumption does not allow crossing hotlinks, we add the following additional restriction to our model for all pair of crossing hotlinks: $x_{i,j} + x_{a,b} \leq 1$.

Finally, since we can use at most one hotlink from each node, we have the following restriction: $\sum_{j=1}^{n} x_{i,j} \leq 1$, for each node $i$.

The objective of ACHS is modeled as follows: $\displaystyle\sum_{k=1}^{m} p_k \left( c_i - \sum_{i,j \in \lambda_k} y^k_{i,j} \right)$.

### 3.3 The PATH Algorithm

In this section, we present the PATH algorithm [PLdS03], an exact dynamic programming for solving 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 $E(T^{A^*_D}, \mathbf{p}) \leq E(T^A, \mathbf{p})$.

Observe that the ACHS 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-ACHS problem (P-ACHS for shortness).

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 six conditions:

a) $A$ is feasible in the sense of the ACHS 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) $H(T_\mathbf{q}^A) \leq D$.

**Objective:** Minimize the expected level of a hotleaf in the resulting improved tree $T_\mathbf{q}^A$.

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

### 3.3.1 Solving P-ACHS

Figure 2 is used throughout this section to illustrate some ideas. Figure 2.(a) presents an instance of P-ACHS 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 $i$th 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.

Let $E^*(\mathbf{q}, \mathbf{a}, T^s)$ denote the cost of an optimal solution of a P-ACHS 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 sets $E^*(\mathbf{q}, \mathbf{a}, T^s) = \infty$. 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 2.(a), then PATH generates the subproblem of Figure 2.(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 ACHS problem definition (see Section 1.1) assures that

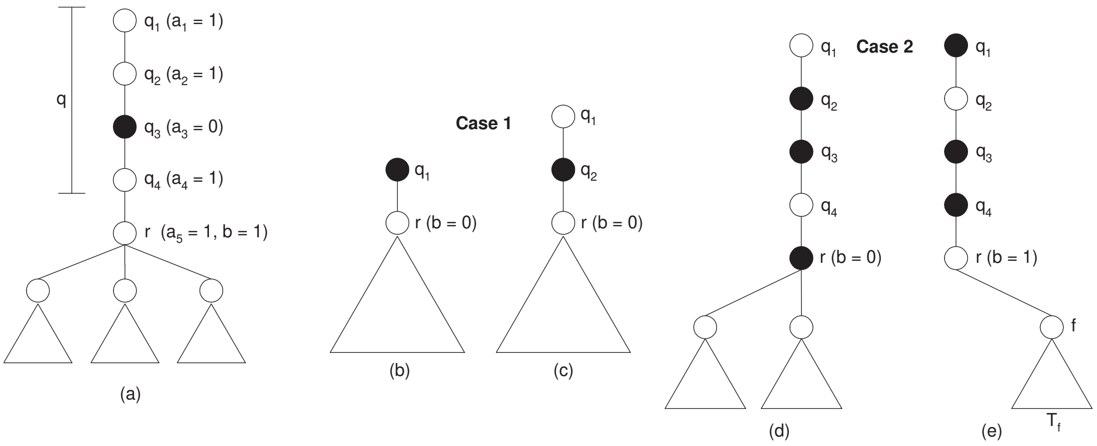


Figure 2: (a) an instance of the A-ACHS problem. (b) and (c) two possible subproblems generated in the case 1. (d) and (e) the decomposition in the case 2 when $c = (0, 1, 0, 0, 1)$.

no node can receives two hotlinks. When $(q_2, r)$ is assigned, instead of $(q_1, r)$, then PATH generates the subproblem of Figure 2.(c). In general, if some hotlink points to $r$ in the optimal solution, we have that

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

**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 2.(e)). As a consequence, only $q_1$ and $q_4$ will be allowed to point to nodes in $T^s - T_f$ (Figure 2.(d)). Figure 2.(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 2.(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 2.(d) and 2.(e) is the cost of the optimal solution for the problem of Figure 2.(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 $C = \{(c_1, \ldots, c_{k+1}) | c_i \le a_i \text{ for } i = 1, \ldots, k+1\}$. 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

$$E^*(\mathbf{q}, \mathbf{a}, T^s) = \min_{\mathbf{c} \in C} \{E^*(\mathbf{q} \to r, \mathbf{c}11, T_f) + E^*(\mathbf{q}, \overline{\mathbf{c}}0, T^s - T_f)\}, \tag{2}$$

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 (1) and (2). Thus,

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

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

$$E^*(\mathbf{q},\mathbf{a},T^s) = \begin{cases} p_x \min\{i | 1 \le i \le k \text{ and } a_i = 1\} \text{ if } \mathbf{q} \text{ has some available node} \\ p_x k, \text{ otherwise} \end{cases} \quad (3)$$

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

# 4 The EX-PATH Implementation

In this section, we discuss two ideas that we implemented on the EX-PATH code in order to improve both its performance and its memory consumption with respect to the PATH algorithm.

## 4.1 Reducing the number of subproblems

Let $(\mathbf{q},\mathbf{a},T^s)$ be a subproblem generated by EX-PATH. The tree $T^s$ is obtained from some subtree $T_r$ of $T$ by removing the last $i$ children of $r$ (following some arbitrary order). Hence, we use the last non-removed child of $r$ to identify $T^s$ in the set of all generated trees (except for the single hotleaves). We denote this last child by $f$. For example, in the subproblem of Figure 2.(a), $f$ is the third child of $r$ (from left to right). On the other hand, $f$ is the second child of $r$ for the subproblem of Figure 2.(d). Moreover, let $\alpha$ be the binary value of the vector $\mathbf{a}$, given by $\sum_{i=1}^{k+1} a_i\ 2^{i-1} + b\ 2^{k+1}$. We also use this value to identify the vector $\mathbf{a}$ in the set of all generated vectors. By means of the dynamic programming technique, our implementation maintains a table that contains the optimum values for all generated subproblems. Hence, an element of this table can be identified by the pair $(\alpha, f)$.

When EX-PATH decomposes a subproblem according to the case 2, it checks $2^{k'}$ possibilities, where $k' = \sum_{i=1}^{k+1} a_i$ is the number of available hotlinks in both the path $\mathbf{q}$ and the node $r$. The following lemma, whose proof we defer for the extended version of this paper, allows us to reduce this number to $2^m$ when $k' \ge m$.

**Lemma 1** *Let $I = (\mathbf{q},\mathbf{a},T^s)$ be an instance of P-ACHS generated as a subproblem for an input tree $T$, where $T^s$ has $m$ leaves and $\mathbf{q}$ has $k' \ge m$ available nodes. Then, there is an optimal assignment for $T$ does not use $I$.*

By using Lemma 1, we discard many elements in this table. To efficiently store and retrieve the optimum values for the non-discarded subproblems, we propose a new indexing method to identify the vector $\mathbf{a}$. Let $\beta$ be the number of non-discarded vectors whose binary values are not greater than $\alpha$. We use the pair $(\beta, f)$ to identify an element in our improved table. In this case, newPosition$(\mathbf{a}, k+1, m)$ gives the value of $\beta$ for a subproblem identified by the vector $\mathbf{a}$ of size $k+2$ and a tree $T^s$ with $m$ hotleaves. A pseudocode for the function newPosition is presented in the Algorithm 1. The idea behind this function is that, for each element $a_{k+2-j} = 1$, all non-discarded vectors with $a_{k+2-j} = 0$ are skipped. In this case, the number of skipped vectors is given by $\sum_{i=0}^{m} \binom{j-1}{i}$.

**Algorithm 1:** newPosition ($\mathbf{a}$, $j$, $m$)
(1) If $m = 0$ then return 0;
(2) If $m \ge j$ then return $j$;
(3) If $a_{k+2-j} = 1$ then return $\sum_{i=0}^{m} \binom{j-1}{i}$ + newPosition$(\mathbf{a}, j-1, m-1)$;
(4) else, return newPosition$(\mathbf{a}, j-1, m)$.

To improve the performance of this function, we obtain the value of $\sum_{i=0}^{m} \binom{j-1}{i}$ from another table constructed in a preprocessing phase, for $j = 1, \ldots, D$ and $m = 1, \ldots, j-1$.

## 4.2 Weakening the $D$-restriction

Let $(\mathbf{q}, \mathbf{a}, T^s)$ be an instance of P-ACHS. Let also $z = \min\{i \mid a_i = 1\} - 1$. We observe that $E^*(\mathbf{q}, \mathbf{a}, T_r) = z\, p(T^s) + E^*(\mathbf{q}', \mathbf{a}', T_r)$, where $\mathbf{a}'$ ($\mathbf{q}'$) is obtained from $\mathbf{a}$ ($\mathbf{q}$) by removing its first $z$ elements (nodes), and $p(T^s)$ is the sum of the probabilities of all hotleaves in $T^s$. For example, if we have $\mathbf{a} = (0, 0, 0, 1, 0, 1, 1)$, then $z = 3$ and $\mathbf{a}' = (1, 0, 1, 1)$. In this case, since no hotlink can be assigned from $q_1, q_2$ or $q_3$, these nodes only cause a constant to be added to the optimum value. Since we can use the value of $E^*(\mathbf{q}', \mathbf{a}', T_r)$ to calculate $E^*(\mathbf{q}, \mathbf{a}, T_r)$, we need only to store $E^*(\mathbf{q}, \mathbf{a}, T_r)$ in the table.

The immediate consequence for the previous improvement is to reduce both the time and space needed by our implementation. Now, we show a further consequence of this improvement. Observe that we do not need to restrict the size of $\mathbf{q}$ to be not greater than $D$ as it does not directly affect the size of our table. Instead, we restrict only the size of $\mathbf{q}'$. Due to this improvement, our implementation obtains the best possible solution $T^{A_D^*}$ under a restriction that is weaker than $H(T^{A_D^*}) \leq D$. In fact, we only impose that no hotlink assigned in our solution traverses a path (already compressed by other hotlinks) whose length is greater than $D - 1$. Moreover, this restriction does not include hotlinks directly assigned to hotleaves as they are considered as base cases. Hence, our implementation obtains optimal solutions whenever $D = H$.

# 5 Experimental Results

In this section, we describe our experiments. All of them were executed in a Xeon 2.4 GHz machine with 1GB of RAM Memory. The MIP algorithm was implemented on XPRESS-MP optimization package [Opt].

## 5.1 Instances

In order to obtain our instances, we adopted the same approach employed by Czyzowicz [CKK+03]. First, we obtained the directed graphs associated to 21 Brazilian Universities sites. Let $G^1, \ldots, G^{21}$ be these graphs and let $s_i$, for $i = 1, \ldots, 21$, be the node that models the home page of the $i$th site. Next, for $i = 1, \ldots, 21$, we executed a breadth first search in $G^i$, starting at $s_i$, to extract a tree $T^i$. This process has generated 21 main trees. Then, for each tree $T^i = (V^i, E^i)$ we generated $|V^i|$ additional trees by considering each subtree rooted at a node in $T^i$. Finally, we discard the trees with height smaller than 3 so as to account a total of 1914 trees, including the main ones.

Table 1 indicates some parameters of the main trees. All of them have been defined before but $d$, the maximum node degree in a tree.

In order to assign probabilities to the leaves, we employed the Zipf's distribution, where the probability of the $i$th most probable leaf is given by $p_i = \frac{1}{iH_m}$. Here, $H_m$ is the harmonic number $H_m = \sum_{j=1}^{m} \frac{1}{j}$. The usage of such a distribution is motivated by the work of Glassman [Gla94], which experimentally proves that the popularity of Web pages can actually be modeled with Zipf's popularity law. We note that the popularity order among the leaves was randomly selected.

## 5.2 Results

For each of the 1914 instances, we executed the four algorithms: greedyBFS [CKK+03], AHA [KKS01], EX-PATH and MIP. We executed EX-PATH with $D = 14$. This value is the maximum one for which EX-PATH can execute all instances without memory overflow. Since $H > 14$ for only two instances, then EX-PATH has found the optimal solution for at least 1912 instances.

The gain of an algorithm $Alg$ for an instance $(T, \mathbf{p})$ is given by $(E[T, \mathbf{p}] - E[T^A, \mathbf{p}])/E[T, \mathbf{p}]$, where $A$ is the set of hotlinks that $Alg$ adds to the tree $T$. The leftmost chart in Figure 3 shows the average gain of each algorithm as a function of the input height. On the other hand, the rightmost chart shows how the solution cost of each algorithm compares to that of EX-PATH. Recall that all solutions of EX-PATH

| Site | $n$ | $m$ | $d$ | $H$ | $E[T,p]$ |
|---|---|---|---|---|---|
| www.ufs.br | 542 | 440 | 47 | 6 | 3,61 |
| www.ufrn.br | 320 | 254 | 28 | 7 | 5,06 |
| www.pucminas.br | 1016 | 921 | 81 | 4 | 3,00 |
| www.ufmg.br | 2275 | 1863 | 153 | 6 | 4,79 |
| www.pucsp.br | 1151 | 938 | 43 | 7 | 5,13 |
| www.unicamp.br | 57877 | 45559 | 358 | 10 | 7,69 |
| www.puc-rio.br | 13315 | 11618 | 513 | 8 | 5,45 |
| www.ucb.br | 1158 | 933 | 52 | 7 | 5,41 |
| www.ucsal.br | 369 | 219 | 43 | 5 | 3,97 |
| www.ufg.br | 556 | 477 | 78 | 4 | 2,74 |
| www.ucg.br | 2317 | 2016 | 76 | 6 | 4,47 |
| www.ufmt.br | 746 | 583 | 84 | 9 | 5,62 |
| www.ufms.br | 1743 | 1534 | 390 | 3 | 2,61 |
| www.ufpr.br | 646 | 554 | 80 | 5 | 3,53 |
| www.pucrs.br | 10484 | 8225 | 426 | 16 | 11,63 |
| www.ufsc.br | 443 | 341 | 40 | 10 | 6,89 |
| www.ufba.br | 1892 | 1446 | 148 | 9 | 5,68 |
| www.ufpe.br | 842 | 641 | 30 | 10 | 6,95 |
| www.unicap.br | 48 | 35 | 21 | 6 | 4,86 |
| www.ufpb.br | 117 | 96 | 20 | 4 | 3,32 |
| www.ufpi.br | 1100 | 650 | 48 | 8 | 5,57 |

Table 1: The main instances obtained from Brazilian Universities sites.

are optimal for $H \leq 14$. We observe the solutions of both the greedyBFS and AHA become farther from the optimal ones as $H$ increases. This fact is less clear for $H > 10$ because we have few instances in this range. We also observe that, as in [CKK$^+$03], the solutions given by greedyBFS are better than that of AHA. However, we give the additional information that the greedyBFS solutions are at most 5% from the optimal ones (in the average), for $H \leq 10$.

In terms of speed, both greedyBFS and AHA solved all instances in less than 0.3 seconds. XPRESS exactly solved 1681 instances spending up to 1 minute, 18 of the instances took 1 to 2 minutes time and 215 instances did not finish in less than 1 hour. EX-PATH exactly solved 1880 instances in up to 0.3 seconds, 28 in up to 10 seconds and the remaining 6 (for 2 of them we do not have an optimality certificate) under 27 minutes. For these 6 instances we gradually reduced the value of $D$ so as to observe both the performance gain and the solution quality loss. Tables 2 and 3 illustrate the performance of EX-PATH for these instances.

Table 2 shows the gain for each value of $D$, as well as the corresponding execution times. The bold printed values are either non-optimal gains or gains for which we do not have an optimality proof as for the first two rows. We remark that, without weakening the $D$-restriction, EX-PATH only finds a solution $A$ when $H(T^A) \leq D$. After weakening the $D$-restriction, however, this is not necessarily true. For example, for the instance 4, the solution found for $D = H - 4 = 9$ leads to an optimal tree whose height is 12 ($H(T^{A^*}) = 12$). As a second example, for the fifth instance, the solution found for $D = H - 6 = 4$ leads to a tree whose height is 10. Despite of the very restrictive value of $D$, this last tree has a gain of $39,69\%$ against an optimum gain of $40,96\%$. Although, greedyBFS has found good solutions for all instances, only the last gain of the fifth row is worse than that of greedyBFS.

Finally, Table 3 shows the allocated memory and the relative reduction on the memory usage due to the first improvement of section 4. Roughly speaking, we observe that the memory usage divides by two whenever the value of $D$ decreases by one. Moreover, the first improvement provided a reduction on the memory usage of about 50% for large values of $D$. This reduction deteriorates as the value of $D$ decreases.

# References

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

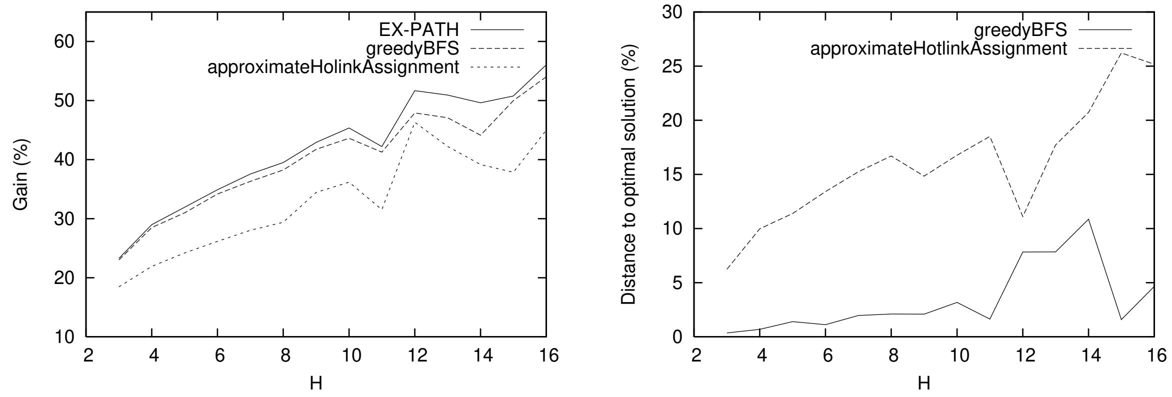


Figure 3: Average gain (%) due to each algorithm as a function of $H$ (leftmost), and relative difference between the solution cost of each algorithm and that of EX-PATH (rightmost).

| | | D | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $H$ | | $H-1$ | | $H-2$ | | $H-3$ | | $H-4$ | | $H-5$ | | $H-6$ | | $H-7$ | |
| Instance | $H$ | % | sec | % | sec | % | sec | % | sec | % | sec | % | sec | % | sec | % | sec |
| 1 | 16 | - | - | - | - | **56,06** | 1589 | **56,06** | 786 | **56,06** | 387 | **56,06** | 178 | **56,06** | 68 | **56,06** | 26 |
| 2 | 15 | - | - | **50,78** | 903 | **50,78** | 539 | **50,78** | 269 | **50,78** | 132 | **50,78** | 61 | **50,78** | 24 | **50,78** | 9 |
| 3 | 14 | 49,62 | 313 | 49,62 | 309 | 49,62 | 186 | 49,62 | 92 | 49,62 | 45 | 49,62 | 20 | 49,62 | 8 | 49,62 | 3 |
| 4 | 13 | 50,94 | 108 | 50,94 | 108 | 50,94 | 63 | 50,94 | 32 | 50,94 | 16 | 50,94 | 7 | **50,93** | 3 | **50,85** | 1 |
| 5 | 10 | 40,96 | 50 | 40,96 | 44 | 40,96 | 32 | 40,96 | 14 | 40,96 | 6 | **40,91** | 3 | **39,69** | 1 | **32,22** | $<1$ |
| 6 | 12 | 51,69 | 37 | 51,69 | 37 | 51,69 | 21 | 51,69 | 11 | 51,69 | 5 | 51,69 | 2 | 51,69 | $<1$ | **50,82** | $<1$ |

Table 2: Gain (%) due to the EX-PATH algorithm, and execution times (seconds), for $D$ ranging from $H$ to $H-7$.


[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.

[Gla94] Steve Glassman. A caching relay for the world wide web. In *First International World-Wide Web Conference*, pages 69–76, May 1994.


| | | D | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $H$ | $H-1$ | | $H-2$ | | $H-3$ | | $H-4$ | | $H-5$ | | $H-6$ | | $H-7$ | | |
| Instance | $H$ | MB | % | MB | % | MB | % | MB | % | MB | % | MB | % | MB | % | MB | % |
| 1 | 16 | - | - | - | - | 637 | 51 | 355 | 46 | 197 | 40 | 108 | 34 | 59 | 28 | 32 | 23 |
| 2 | 15 | - | - | 600 | 54 | 332 | 49 | 185 | 43 | 103 | 37 | 57 | 31 | 31 | 25 | 17 | 19 |
| 3 | 14 | 561 | 57 | 314 | 52 | 173 | 47 | 97 | 41 | 54 | 34 | 30 | 27 | 16 | 21 | 9 | 15 |
| 4 | 13 | 295 | 55 | 165 | 49 | 91 | 44 | 51 | 38 | 28 | 30 | 16 | 23 | 8 | 17 | 5 | 10 |
| 5 | 10 | 233 | 48 | 138 | 39 | 84 | 26 | 46 | 18 | 25 | 11 | 13 | 5 | 7 | 0 | 4 | 0 |
| 6 | 12 | 130 | 53 | 74 | 46 | 41 | 40 | 23 | 33 | 13 | 25 | 7 | 17 | 4 | 10 | 2 | 3 |

Table 3: Allocated memory (MB) by the EX-PATH algorithm, and the relative reduction obtained through the first improvement, for $D$ ranging from $H$ to $H-7$.

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

[Opt] DASH Optimization. XPRESS-MP software. www.dashoptimization.com.

[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.

[PLdS03] Artur A. Pessoa, Eduardo Sany Laber, and Críston de Souza. On the worst case search in trees with hotlinks. Technical Report 23, Departamento de Informática, PUC-RJ, Rio de Janeiro, Brasil, August 2003.