An Experimental Study of Recent Hotlink Assignment Algorithms

TOBIAS JACOBS
National Institute of Informatics, Tokyo

The concept of *hotlink assignment* aims at enhancing the structure of Web sites such that the user's expected navigation effort is minimized. We concentrate on sites that are representable by trees and assume that each leaf carries a weight representing its popularity.

The problem of optimally adding at most one additional outgoing edge ("hotlink") to each inner node has been widely studied. A considerable number of approximation algorithms have been proposed and worst-case bounds for the quality of the computed solutions have been given. However, only little is known about the practical behavior of most of these algorithms.

This article contributes to closing this gap by evaluating all recently proposed strategies experimentally. Our experiments are based on trees extracted from real Web sites, as well as on synthetic instances. The latter are generated by a new method that simulates the growth of a Web site over time. Finally, we present a new heuristic that is easy to implement and exhibits excellent behavior in practice.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph Algorithms*; E.1 [**Data Structures**]: Trees

General Terms: Algorithms, Design, Measurement, Performance, Experimentation

Additional Key Words and Phrases: Search tree, approximation, hotlink

ACM Reference Format:

Jacobs, T. 2010. An experimental study of recent hotlink assignment algorithms. ACM J. Exp. Algor. 15, Article 1.1 (January 2010), 18 pages.
DOI = 10.1145/1671970.1671971 http://doi.acm.org/10.1145/1671970.1671971

1. INTRODUCTION

The design of Web sites typically aims at making a large amount of information conveniently accessible. Web designers cannot arbitrarily distribute the contents among the pages, as this would make information retrieval too complex

Work supported by the Deutsche Forschungsgemeinschaft, project AL 464/5-1.
Author's address: T. Jacobs, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan; email: jacobs@nii.ac.jp.

DOI 10.1145/1671970.1671971 http://doi.acm.org/10.1145/1671970.1671971

for the users. The site's structure must somehow represent structural properties of the information. On the other hand, the interests of users on most Web sites are highly correlated. Typically, the most popular 20% of the pages are queried by about 80% of all Web server accesses (see Pitkow [1999]). Moreover, the popularity of pages is likely to change over time.

By automatically moving popular pages closer to the homepage, one can both reduce Web traffic and increase user friendliness. However, restructuring the whole Web site is not practical for reasons mentioned in the preceding paragraph. In contrast, the concept of adding additional hyperlinks called *hotlinks* to the pages is a nondestructive approach, which preserves the original site's structure.

We concentrate on Web sites that are representable by a rooted tree, where the root is the homepage, inner nodes represent navigation pages, and the information is contained in the leaves, each having a certain popularity. This model has been widely studied in literature (see the related work section for details), and a number of algorithms with provably good worst-case behavior have been proposed. Our main purpose is to evaluate these algorithms experimentally and to make a statement about which of them are recommendable in practice.

1.1 Problem Definition

A weighted tree T is a triple (V, E, ω), where (V, E) is a tree rooted at a node $r \in V$. Let $L \subseteq V$ be the set of leaves. The weight function $\omega : L \to \mathbb{R}_0^+$ assigns a nonnegative weight to each leaf.

For nodes $u \in V$, we also write $u \in T$. The set of ancestors/descendants of u, not including u itself, is denoted $\mathrm{anc}(u)/\mathrm{desc}(u)$, $\mathrm{ch}(u)$ is the set of u's direct children, and $\mathrm{par}(u)$ denotes the parent of u.

A *hotlink* (u, v) is an additional directed edge between nodes in T. We say that u is the *hotparent* of v and v is the *hotchild* of u. We further say that the hotlink *starts* in u and *ends* in v. A set $A \subset V \times V$ of hotlinks is called a *hotlink assignment (HLA)*.

In this work, we assume that a user only knows about hotlinks that start in nodes she has already visited, and she always uses any hotlink taking her closer to her destination leaf. This natural behavior is called the *greedy user model* or *obvious navigation assumption*. In contrast, in the clairvoyant user model, users always know their shortest path in $(V, E \cup A)$. In our opinion, the greedy user model is more realistic. To our best knowledge, all papers about hotlinks that have appeared since its introduction in Pessoa et al. [2004b] and Gerstel et al. [2003] consider that model.

Referring to Pessoa et al. [2004b], a hotlink assignment A is called *feasible* if it satisfies the following properties:

(i) $v \in \mathrm{desc}(u)$ for any $(u, v) \in A$.

(ii) If $(u, v) \in A$, then there is no $(u', v') \in A$ with $u' \in \mathrm{desc}(u) \cap \mathrm{anc}(v)$ and $v' \in \mathrm{desc}(v)$.

(iii) For any node $u \in T$, there is at most one $(u, v) \in A$.

Properties (i) and (ii) exclude hotlinks that would never be taken by a greedy user. Property (iii) reflects the requirement that the number of hotlinks on a

concise Web page must be somehow limited. However, many algorithms naturally generalize to relaxations of that property. In the remainder of this article, when talking about hotlink assignments, we always mean feasible HLAs.

A hotlink assignment A is optimal if the path length

$$p(A) = \sum_{l \in L} \omega(l) \mathrm{dist}^A(r, l)$$

is minimal among all hotlink assignments for the tree under consideration, where $\mathrm{dist}^A(r, l)$ denotes the length of the greedy user's path from r to l in the graph $(V, E \cup A)$. In assignments that are feasible according to conditions (i) and (ii), the greedy users always takes the shortest path. We abbreviate $\mathrm{dist}^{\emptyset}(u, v)$ by $\mathrm{dist}(u, v)$. The task of finding an optimal hotlink assignment for a given tree is called the *hotlink assignment problem*.

Equivalently, one can also maximize the *gain*

$$g(A) = p(\emptyset) - p(A).$$

Despite equivalence with respect to optimal solutions, the two optimization terms are not equivalent when considering approximation ratios. Most known polynomial time algorithms for the hotlink assignment problem approximate only one of these terms up to a good ratio (see Table II). On the other hand, it is possible to combine the approximation properties of several algorithms by subsequently running each of them and then choosing the best among the computed hotlink assignments.

1.2 Related Work

The hotlink assignment problem has first been formulated by Bose et al. [2000]. They have shown that it is NP-hard when considering general DAGs. The result applies to the clairvoyant user model. Note that it is not clear how to extend the greedy user model to DAGs, as in that kind of graphs shortest paths are not necessarily unique, and so the user's behavior is not determined by the graph structure. We have recently proven that in the greedy user model the problem is NP-hard for trees [Jacobs 2008].

An optimal algorithm whose runtime and space requirements are exponential in the depth of the tree (and thus polynomial for trees of logarithmic depth) has been independently discovered by Gerstel et al. [2007] and Pessoa et al. [2004b]. An implementation of this algorithm has been confirmed in Pessoa et al. [2004b] and Gerstel et al. [2007] to be able to find optimal solutions for trees having hundreds of thousands of nodes and a depth up to 14.

The first approximation algorithm for the hotlink assignment problem has been presented in Kranakis et al. [2004]. Since in Czyzowicz et al. [2003], Pessoa et al. [2004b], and Gerstel et al. [2007] that algorithm has already been shown to be outperformed by greedy-like strategies, we do not include it in our experiments.

Algorithm GREEDY was first formulated in Czyzowicz et al. [2001]. In Jacobs [2007], we proved that it achieves at least half of the gain of an optimal solution in the greedy user model. Another 2-approximation in terms of the gain was presented by Matichin and Peleg [2007]. Their algorithm assigns only hotlinks

that bypass exactly one node. In Jacobs [2007], we generalized this approach by showing that restricting hotlinks to some length h leads to a guaranteed approximation ratio of $\frac{h}{h-1}$ in terms of the gain. Using a modified version (called LPATH) of the optimal algorithm mentioned earlier in the text, such hotlink assignments can be computed in polynomial time for constant h, so LPATH is a PTAS. The previously mentioned algorithm of Matichin and Peleg is implicitly covered by the evaluation of LPATH for different values of h, including $h = 2$. By assigning the tree's depth to h, we also obtain an efficient method for computing optimal solutions. This is crucial as we are interested in approximation ratios that occur in practice.

Another family of recent algorithms is more tailored to approximating the resulting path length. A fundamental lower bound for that optimization term has been given in Bose et al. [2000]: Let Δ be the outdegree of the tree under consideration. Then, no hotlink assignment can achieve a path length better than $H(\omega)/\log(\Delta + 1)$, where $H(\omega) = \sum_{l \in L} \omega(l) \log(1/\omega(l))$ is the entropy of the probability distribution among the leaves (assuming that the weights are normalized to sum up to 1). In Douïeb and Langerman [2005, 2006], they have presented two algorithms that guarantee a path length of $O(H(\omega))$ and are thus constant factor approximations for trees of constant degree. In Jacobs [2007], we have introduced a 2-approximation algorithm in terms of the path length for arbitrary trees.

1.3 Our Contribution

Generally speaking, the intention of this work is to close the gap between theory and practice concerning the hotlink assignment problem. What is already known from previous experimental studies [Czyzowicz et al. 2003; Gerstel et al. 2007] is that that greedy-like algorithms outperform the approximation method described in Kranakis et al. [2004], and that optimal solutions to fairly large instances can be computed on a standard PC [Gerstel et al. 2007].

The main purpose of the present study is to evaluate the behavior of the numerous approximation algorithms that have been proposed during the last few years. These algorithms guarantee approximation factors in terms of different measures (gain, path length, entropy), and we seek to find out which of these approaches leads to good solutions in practice. We have implemented all algorithms mentioned earlier in the text and evaluated their performance by applying them to two different sets of trees. The first set contains instances that have already been used for the tests in Gerstel et al. [2007]. We have extended this set by extracting 20 additional trees from German university Web sites. These new instances have a depth of up to 179, so they are typically harder to handle by optimal algorithms than the instances from the original set. The trees contained in the second set have been generated by a new random construction method that is based on a model by Barabási and Albert [1999].

A second purpose of this work is to investigate to what extent the approximation algorithms from theory are beaten by hand-tuned heuristics. We present a new heuristic for the hotlink assignment problem, which assigns hotlinks such that the estimated path length is minimized. Any method for estimating the

minimal possible path length can be plugged into that heuristic. It turns out that plugging in a slightly modified version of the lower bound $p_{\min}$, given in Jacobs [2007], produces excellent results. We will see that these are at most 5% away from the optimal solution for almost all instances, which is only beaten by the PTAS. However, the gap between the solution quality obtained by the heuristic and the quality achieved by some of the algorithms with known approximation guarantees is not too high, and latter algorithms are often faster.

The article is organized as follows: Some further notation is given in Section 2. That section also introduces the hotlink assignment algorithms we have evaluated experimentally. Section 3 describes the acquisition of our test instances and the experimental set-up. The results of our study are presented in Section 4. Section 5 concludes the article.

2. FURTHER NOTATION AND ALGORITHMS

In this section, we introduce some further notations for our optimization problem. We then describe the hotlink assignment algorithms we have evaluated in our experiments and discuss implementation issues.

2.1 Notation

For $u \in T$, we denote by T_u the maximal subtree of T rooted at u. For any set V' of nodes, $T \setminus V'$ is the maximal subtree of T that is obtained from T by removing all subtrees rooted at an element of $V' \cap V$. Furthermore, for any hotlink assignment A, let $T_{u,A} = T_u \setminus \{v \mid \exists (u', v) \in A, u' \notin T_u\}$ be the subtree obtained from T_u by omitting all subtrees rooted at hotchildren of u's ancestors.

We now define the weights of inner nodes. Interpreting weights as access frequencies, a natural way for extending ω to inner nodes is to assign to them the frequency with which they are visited by the user that navigates through the tree. This frequency possibly changes when hotlinks are assigned or modified. Formally,

$$\omega^A(u) = \sum_{l \in L \cap T_{u,A}} \omega(l).$$

We abbreviate $\omega^{\emptyset}$ with ω.

Next, we introduce an order among the children of a node. The relation "$\succ$" is defined as a total order among siblings such that $\omega(u) > \omega(u') \Rightarrow u \succ u'$ for any pair u, u' of siblings. Ties are broken arbitrarily. The direct successor sibling of u with respect to that order is denoted by $\mathrm{succ}(u)$. The first child of an inner node is its unique child having no predecessor.

Finally, we define the heavy path of a node u, referring to Sleator and Tarjan [1983]. If u is a leaf, then its heavy path only consists of u. Otherwise, it is the path obtained by appending the heavy path of u's first child to u.

2.2 Algorithms

We begin with a number of simple top-down methods. The latter term has been introduced in Douïeb and Langerman [2006] and denotes hotlink assignment

algorithms that assign a hotlink (r, v) and then recursively apply themselves to T_v and all $T_u \setminus \{v\}, u \in \text{ch}(r)$. Thus, any top-down method is fully characterized by the choice of v.

GREEDY: The algorithm GREEDY was first studied in Czyzowicz et al. [2001] (where it is called "recursive"). It is a top-down method where the root's hotchild v is chosen such that $g(\{(r, v)\})$ is maximized.

GREEDY is a 2-approximation in terms of the gain [Jacobs 2007]. It has exhibited the best performance among the approximation algorithms studied experimentally so far [Czyzowicz et al. 2001, 2003; Pessoa et al. 2004b; Gerstel et al. 2007].

H/PH: The H/PH-strategy is also a top-down method. Let h be the node whose weight is closest to $\omega(r)/2$. If $\omega(h) > \alpha\omega(r)$, then h is chosen as the hotchild of r. Otherwise, the parent p_h of h becomes r's hotchild. The threshold α is given as the solution of $(\frac{\alpha}{1-\alpha})^{2(1-\alpha)} = \alpha$, that is, $\alpha \approx 0.2965$.

The H/PH-algorithm has been proposed in Douïeb and Langerman [2006], where it is proved to guarantee a path length of at most $1.141H(\omega)+1$. Thus, it is asymptotically a $(1.141 \log(\Delta+1))$-approximation in terms of the path length, where Δ is the outdegree of the tree (see Section 1).

In Douïeb and Langerman [2006], the authors propose an implementation of H/PH that runs in worst-case time $O(n \log n)$. They observe that it suffices to traverse the root's heavy path when looking for h. As in the worst-case, that path can have a length of $O(n)$, an involved tree representation is employed for finding h in time $O(\log n)$. We do not adopt the tree representation because in typical tree instances, including our test set, the depth is rather small.

PMIN: Assume that there was an oracle telling the exact minimum path length $p_{\text{opt}}(T)$ a hotlink assignment can achieve for a given tree T. Then, an optimal assignment could be easily computed by a top-down method that chooses the hotchild v of r such that

$$p_{\text{opt}}(T_v) + \sum_{u \in \text{ch}(r)} p_{\text{opt}}(T_u \setminus \{v\}) \tag{1}$$

is minimized. Due to Jacobs [2008], such an oracle needs exponential time unless P = NP.

Our algorithmic idea is to guess that optimal path length p_{opt} by some fast estimation method. After experimenting with a number of such methods, it turned out that the usage of

$$p_{\min}(T) = \sum_{u \in V \setminus (L \cup \{r\})} \Big(\omega(u) - \max_{v \in \text{ch}(u)} \omega(v)\Big) + \sum_{l \in L} \omega(l)$$

produces excellent results and is very robust. The term $p_{\min}$ comes from the fact that this estimate is closely related to a lower bound for the path length given in Jacobs [2007]. Intuitively, it is the result of subtracting the weight of each heaviest child, except the root's one, from the path length of the original tree.

For an efficient implementation, instead of explicitly calculating $p_{\min}$, we consider the improvement to $p_{\min}(T)$ caused by the hotlink (r, v). For determining that improvement, it suffices to traverse the path between v and r.

HEAVY PATH: The algorithm HEAVY PATH was proposed in Douïeb and Langerman [2005]. It works as follows: First, split the tree into the set of heavy paths. This can be done in linear time by recursively computing the set of heavy paths of the subtrees rooted at the children of r, uniting these sets and appending r to the path containing the first child of r. Then interpret each of these paths as a list of weighted elements. The weight $W(u)$ of each such element u is $\omega(u) - \omega(v_f)$, where v_f is the first child of u in the tree. If u is a leaf, then $W(u) = \omega(u)$. Then a HLA for each such list $u_1 \dots u_n$ is computed as follows: Assign a hotlink (u_1, u_i) such that $\sum_{1 \le j < i} W(u_j)$ and $\sum_{i < j \le n} W(u_j)$ are both at most $\frac{1}{2} \sum_{1 \le j \le n} W(u_j)$ and recursively apply the algorithm to the sublists $u_2 \dots u_{i-1}$ and $u_i \dots u_n$.

The whole algorithm can be implemented such that it takes linear time. Thus, the heavy path strategy has the lowest worst-case runtime among all strategies evaluated in this work. It guarantees a maximum path length of $3H(\omega)$, making it a $(3 \log(\Delta + 1))$-approximation in term of the path length.

LPATH : The length of a hotlink (u, v) is defined as $\text{dist}(u, v)$. The best assignment of hotlinks that have a maximum length of h is a $\frac{h}{h-1}$-approximation in terms of the gain (see Jacobs [2007]). Algorithm LPATH , which for any given h computes a HLA that is at least as good as the best length h assignment in worst-case time $O(|V|3^h)$, is thus a PTAS in terms of the gain. In our experiments, we also use LPATH as an optimal algorithm by setting h to the tree's depth.

LPATH is a dynamic programming algorithm. Subproblems are determined by a subtree together with a number of hotlinks starting outside that subtree that have already been decided to end in it. More formally, subproblems are defined by a triple (q, a, u), where $q = q_1 \dots q_n$ is a directed path, $a = a_1 \dots a_n a_{n+1} b \in \{0, 1\}^{n+2}$ is a binary vector and u is a node in T. It represents the tree qT^u obtained by appending the tree T^u to q_n. $T^u = T_{\text{par}(u)} \setminus \{u' \mid u' \succ u\}$ (not to be confused with T_u) is the tree obtained by deleting all subtrees from $T_{\text{par}(u)}$ that are rooted at a sibling $u' \succ u$. The vector a represents a number of restrictions concerning HLAs for qT^u. Path nodes q_i are only allowed to have a hotchild if $a_i = 1$. They are not allowed to be hotchildren. The node $r = \text{par}(u)$ is only allowed to be a hotparent if $a_{n+1} = 1$ and may only be a hotchild if $b = 1$.

For a given subproblem (q, a, u), LPATH deletes the first $|q| - h$ components of q and a in case of $|q| > h$. This only excludes hotlinks that would have a length more than h. Then all possible configurations of hotlinks starting in nodes from q are compared. A configuration is determined either by one hotlink from q pointing at r, or by a selection of hotlinks starting at nodes in q that end somewhere in T^u. In the latter case, the remaining hotlinks end in $T^{\text{succ}(u)}$.

Each such configuration results in new subproblems, see Jacobs [2007] for details. As the solution of (q, a, u) only depends on a and u, the number of subproblems is limited by $|V|2^{h+2}$. Summing up the number of configurations that have to be compared for all subproblems results in a runtime of $O(|V|3^h)$ (see Pessoa et al. [2004b]).

Next, we discuss a number of practical improvements to LPATH. The dynamic programming approach for the algorithm has been adopted from the PATH algorithm of Pessoa, Laber, and Souza [2004a] (see also Gerstel et al. [2007]). The only difference to LPATH is that whenever the path q of a subproblem becomes longer than h, PATH gives up. Pessoa et al. [2004b] have proposed two improvements to their algorithm. The first is to increase the number of considered hotlink assignments by always cutting the first component from q and a until $a_1 = 1$. That improvement is already included in the original definition of LPATH, as the latter strategy always cuts the first components q and a when they become too long. The second improvement is a consequence of the observation, that for any subproblem (q, a, u) the total number of hotlinks starting from path nodes q_i is limited by the number of leaves in T^u. Let $\hat{a} = \sum_{1 \le i \le |a|} a_i$ and let l_u be the number of leaves in T^u. We adopt the second improvement by inverting the $\hat{a} - l_u$ components of a having the highest possible indices from 1 to 0 whenever the $\hat{a} > l_u$.

In our experiments, we have observed that the memory requirements of LPATH are by far more critical than the runtime is. For all tree instances, the algorithm either terminated in reasonable time (2 to 3 minutes or less) or the memory requirements exceeded our hardware limit. So, the purpose of our main improvement is to reduce space consumption.

The total number of subproblems considered by LPATH is $\Theta(|V|2^h)$. Fortunately, we do not need to store all of them simultaneously. For any fixed node u, let S_u be the set of solutions for all possible values of (a, u). Let u_f be the first child of u. Then, $S_{\text{succ}(u)}$ and S_{u_f} suffice to compute any element of S_u. Furthermore, $S_{\text{succ}(u)}$ and S_{u_f} are not required for any further computation, so these sets can be removed from memory after computing S_u.

Let $u_1, \ldots, u_n$ be the unique postorder sequence of $V - \{r\}$, where each node appears before its successor with respect to "$\succ$". We successively compute $S_{u_1}, \ldots, S_{u_n}$ and delete any set as soon as it is not required anymore. It is easy to observe that with this policy, any solution of a subproblem is guaranteed to be available when it is required. Note that the solution of the original problem is contained in S_{u_n}. The following lemma bounds the number of solution sets that are simultaneously stored.

LEMMA 2.1. *Let d be the depth of the tree. At any time during the execution of* LPATH*, for $1 \le x \le d$, there is at most one node u with* $\text{dist}(r, u) = x$ *whose solution set S_u is currently stored and not currently computed.*

PROOF. Let u and u' be two nodes that have the same distance to the root and let u' occur after u in the postorder sequence. Let $S_{u'}$ be currently stored and completely computed. If u and u' are siblings, then S_u has been used for the computation of $S_{u''}$, where u'' is either u' or another sibling between u and u', so S_u has already been deleted. If u and u' are not siblings, then the parent v of u occurs in the postorder sequence before u'. In that case, S_v has already been computed, which implies that S_u has been used and thus deleted. □

As only one solution set is computed at a time, it follows directly from Lemma 2.1 that at most $d + 1$ solution sets are simultaneously stored and

Table I. Characteristics of the Set of Real Tree Instances

	Min	Max	Average
size	32	512,484	32,652
depth	3	179	16

thus the memory requirements of our implementation are $O(d2^h)$. The improvement is significant in practice, as the depth is typically small compared to the number of nodes.

CENTIPEDE: In a centipede hotlink assignment, only the first child of each inner node is allowed to be bypassed by hotlinks. The best centipede HLA is a 2-approximation in terms of the path length [Jacobs 2007].

From the centipede restriction follows that for each nonheaviest inner node u of a tree T the partial hotlink assignment for T_u is independent of that for T', where T' is obtained from $T \setminus \{u\}$ by appending a leaf of weight $\omega(u)$ to the the parent of u. So, when computing the best centipede hotlink assignment for T, the trees T_u and T' can be considered separately.

By applying this observation to each nonheaviest inner node, the algorithm CENTIPEDE splits the tree into a set of *centipede trees*, which are trees whose inner nodes have at most one nonleaf child. Optimal hotlink assignments to those trees are then computed in polynomial time by a dynamic programming algorithm. That algorithm exploits special properties of optimal assignments for centipede trees and has a worst-case runtime of $O(n^4)$.

3. EXPERIMENTAL SETUP

3.1 Real Instances

Our set of real instances consists of 104 trees. Eighty-four of them represent the structure of Brazilian and U.S. university sites and have been made available by the authors of Gerstel et al. [2007]. Access patterns measured over the time horizon of 1 week are available for one of those instances, namely *puc-rio.br*. We have extended the set by 20 instances representing Web sites of German universities. All trees have been extracted from the corresponding sites using breadth-first search, which implies that for any page v in the original structure, a shortest path from the home page to v becomes the unique path from r to v in the resulting tree.

We note that the German university instances, unlike the others, partially have a large depth up to 179. These kinds of trees are especially hard to handle by optimal algorithms. Typically, very deep subtrees are online tutorials that do not have an index page. Table I shows the main characteristics of our test set.

As further access patterns are not available to us, we have randomly assigned weights to the leaves. We did so using the Zipf distribution, that is, the ith heaviest leaf, which is chosen at random, is assigned a weight of $\frac{1}{iH_m}$, where H_m is the mth harmonic number and m is the number of leaves in the tree. The Zipf distribution is considered as the typical access distribution, see, for example,

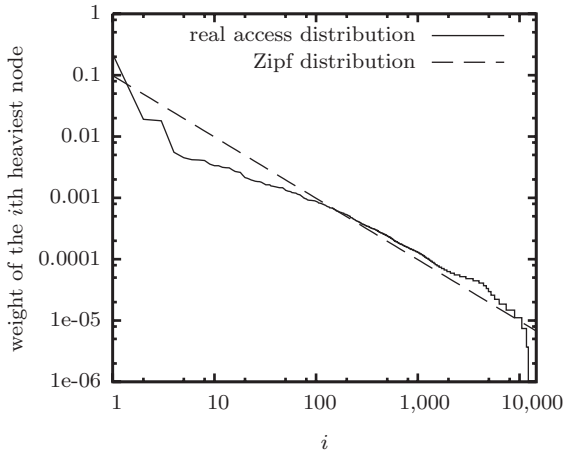

Fig. 1. Real and synthetic access frequency distribution.

Pitkow [1999]. The same approach has also been employed in Czyzowicz et al. [2001], Pessoa et al. [2004b], and Gerstel et al. [2007]. Figure 1 shows the validity of that approach by comparing the access frequency distribution of *puc-rio.br* with the Zipf distribution.

3.2 Synthetic Instances

In order to make reliable statements about the algorithms' behavior for different input sizes, we need a large number of test instances. The available set of real trees is not sufficient for that purpose. Pessoa et al. [2004b] increase the number of instances by considering each subtree of minimum depth 3 that is rooted at a node in one of their original trees. This approach causes strong dependencies in the data set and is, therefore, problematic in our opinion.

Instead, we randomly generate a large number of synthetic trees. To this extent, we have developed a new algorithm that is based on a model of Barabási and Albert [1999]. In their model, a graph, initially containing a small number of m_0 nodes, is built by in each iteration step i adding a new node v_i adjacent to m existing nodes. The probability for any node v to be chosen as one of v_i's neighbors is proportional to the number of nodes already adjacent to v. The authors show that for large i the number of neighbors of a node converges to a power-law distribution. In our case, $m_0 = m = 1$ so that the resulting graph is a tree. Unlike the random constriction method used in a previous experiment [Czyzowicz et al. 2003], our algorithm generates trees with leaves at all levels.

The data set generated for our experiments consists of trees having the sizes 1,000,2,000, ... ,100,000. For each of these size values, 10 instances were generated, so their total number is 1, 000. Like in previous experiments [Czyzowicz et al. 2003; Gerstel et al. 2007], the weights of the leaves have been generated using the Zipf distribution.

In these synthetic trees, the distribution of the nodes' outdegree is similar to the typical distribution in real instances, as Figure 2 shows. Figure 3 displays

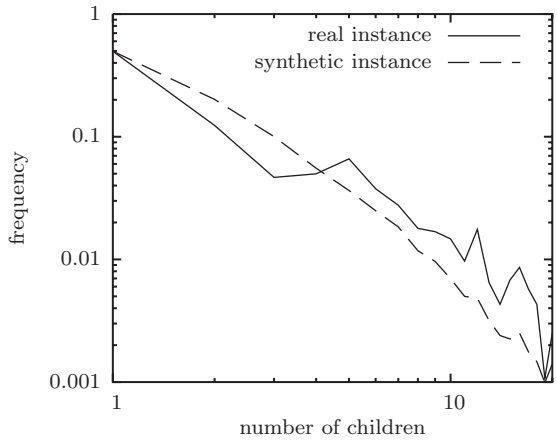

Fig. 2. Distribution of the number of children in real and synthetic instance.

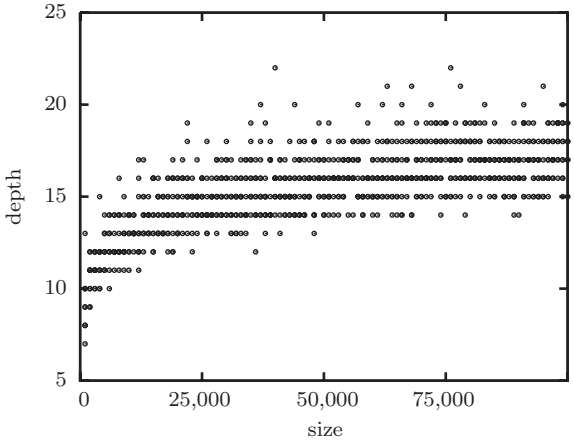

Fig. 3. Relation between size and depth in the synthetic trees.

the depth of the generated trees. It turns out that the depth tends to grow only very slowly with the size.

3.3 Test Environment

We have run all algorithms described in Section 2 on both the real and the synthetic tree instances. In the case of LPATH , we have applied each configuration of $h = 2 \ldots 15$. Computations that required more than 1 hour or exceeded a RAM limit of 500MB were aborted.

We have measured the path length of the resulting hotlink assignments, as well as the runtime of the algorithms. Based on the path length, we have calculated a number of additional values. As defined in Section 1, the gain $g(A)$ of an assignment A is $p(\emptyset) - p(A)$. The main focus of our study lies on the approximation ratios that occur in practice. Therefore, for all instances where

Table II. Theoretical Worst-Case Approximation Ratios and Experimentally Observed Ratios

	Approximation ratio (gain)					
Algorithm	Worst case	Min	Max	Average	Deviation	puc-rio.br
GREEDY	2	1.000	1.351	1.029	0.066	1.283
PMIN	?	1.000	1.094	1.008	0.018	1.000
H/PH	∞	1.052	∞	∞	∞	2.966
HEAVY PATH	∞	1.052	∞	∞	∞	2.977
CENTIPEDE	∞	1.000	2.041	1.138	0.193	1.064
LPATH , $h = 2$	2	1.000	1.530	1.128	0.136	1.113
LPATH , $h = 3$	1.5	1.000	1.096	1.013	0.023	1.000
LPATH , $h = 4$	1.333	1.000	1.047	1.001	0.006	1.000
	Approximation ratio (path length)					
Algorithm	Worst case	Min	Max	Average	Deviation	puc-rio.br
GREEDY	∞	1.000	1.062	1.009	0.013	1.051
PMIN	?	1.000	1.052	1.004	0.009	1.000
H/PH	$1.141 \log(\Delta + 1)$	1.018	1.460	1.167	0.078	1.154
HEAVY PATH	$3 \log(\Delta + 1)$	1.018	1.488	1.195	0.077	1.155
CENTIPEDE	2	1.000	1.148	1.036	0.022	1.014
LPATH , $h = 2$	∞	1.000	1.372	1.069	0.090	1.024
LPATH , $h = 3$	∞	1.000	1.090	1.010	0.019	1.000
LPATH , $h = 4$	∞	1.000	1.030	1.001	0.004	1.000

an optimal solution OPT could be computed by our implementation of LPATH, we have calculated $p(A)/p(OPT)$, as well as $g(OPT)/g(A)$.

4. RESULTS

Our implementation of LPATH terminated for all but 16 of the 104 real instances. The remaining trees have a depth of between 14 and 179 and a size of between 74,042 and 512,484. Gerstel et al. [2007] report about two hard instances that require 488MB and >1GB of RAM. Our experience with those instances was nearly the same, so we suppose that the implementation of OPT in that work uses similar optimization techniques as ours. The other algorithms always terminated.

Due to their smaller depth (see Figure 3), optimal solutions for all but one synthetic instances could be computed within our resource bounds.

4.1 Solution Quality

Table II gives an overview of the approximation ratios the algorithms have achieved on the real instances[1] and compares these ratios to the theoretical upper bounds.

The results achieved on instance *puc-rio.br* can be found in the right-most column. The relative performance ranking of the algorithms equals the ranking observed from the instances with artificial weights. With exception of GREEDY, the solution quality on *puc-rio.br* is closer to the optimum than the average quality achieved on the other instances, but without having more real access data, it is hard to tell whether or not this is something typical.

[1]Of course, only those instances are considered in the table where an optimal hotlink assignment is available.

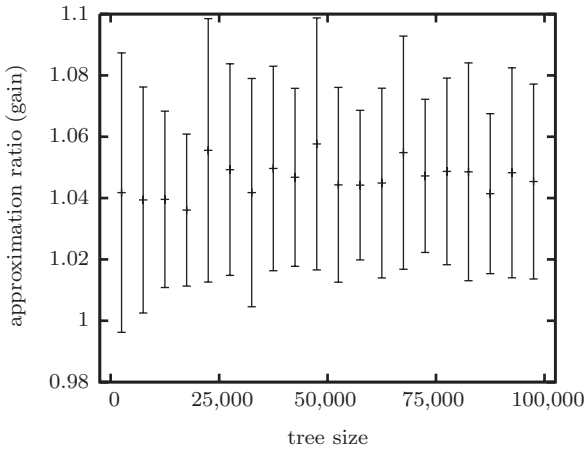

Fig. 4. Ratio (gain) of GREEDY and PMIN for different tree sizes.

For algorithms H/PH and HEAVY PATH, no bound in terms of the gain had been given yet. However, trees having a depth of 2 where no hotlink at all is assigned by these strategies are easy to construct, so their ratios in terms of the gain do not exist. As shown in Table II, this scenario even occurs in practice. Concerning the path length, H/PH and HEAVY PATH achieve approximation ratios less than 1.5 on all instances. However, the ratios of all other algorithms are better for both measures.

With exception of the PTAS, the best results are achieved by PMIN, which achieves better ratios for the path length than all other algorithms. The differences in terms of the gain are even more pronounced. PMIN is only beaten by LPATH for $h \geq 4$. As observed in previous experiments on hotlink assignment, algorithm GREEDY also achieves comparatively good results. It is, by far, better than LPATH for $h = 2$. The latter algorithm has the same worst-case approximation ratios as GREEDY and simulates the behavior of the 2-approximation given in Matichin and Peleg [2007]. Algorithm CENTIPEDE computes solutions whose quality is still better than those of H/PH and HEAVY PATH, but not that good as GREEDY or PMIN.

The experiments based on the random instances allow for some more detailed findings. The average gain and the standard deviation of algorithm GREEDY for varying tree size is visualized in Figure 4, which exemplarily shows that the algorithms' solution quality is basically independent of the tree size. This is the case for all algorithms and types of ratios.

The relative performance ranking observed from Table II is also confirmed by the histograms in Figures 5, 6, and 7. For almost all instances, PMIN is less than 5% from the optimal solution's path length (Figure 6). Algorithm GREEDY deviates by less than 10% from the optimum for most instances. The same holds for CENTIPEDE, but the majority of its solutions are clearly worse than the solutions computed by GREEDY. Algorithms H/PH and HEAVY PATH are far-off from the other algorithms, their ratios are between 1.15 and 1.35.

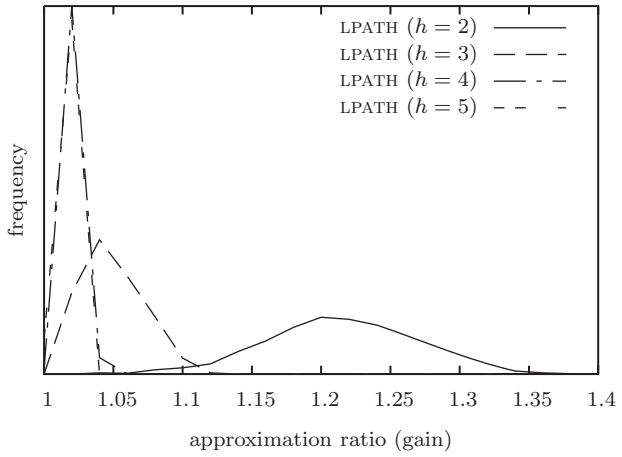

Fig. 5. Histograms of the approximation ratio in terms of the gain for LPATH.

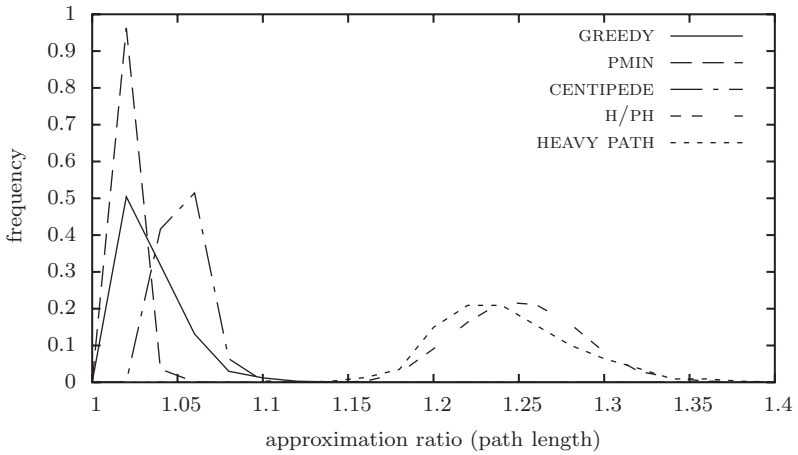

Fig. 6. Histogram of the approximation ratio in terms of the path length.

All these phenomena can also be observed for the gain in Figure 7, although all ratios are typically higher here. The difference between PMIN and GREEDY is even more pronounced, and the ratios for the gain of H/PH and HEAVY PATH are mainly distributed between 1.3 and 2.2.

Figure 5 visualizes the behavior of LPATH for different values of h. For $h = 3$, the results are comparable to those of GREEDY. For $h > 3$, LPATH is very close to the optimal solution. This can also be observed in Figure 8. On each of the four selected instances of real trees, the approximation ratio converges to 1 much faster than in theory. However, it seems that the speed of convergence is lower for trees having a greater depth d. Recall that for $h \geq d$ a ratio of 1 is guaranteed.

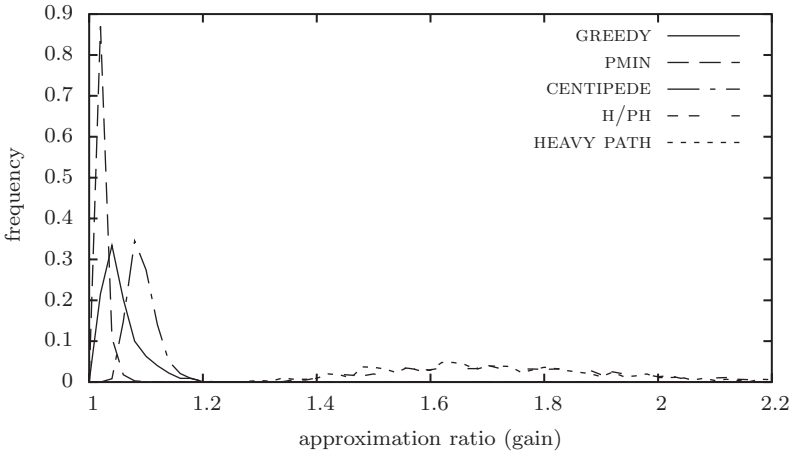

Fig. 7. Histogram of the approximation ratio in terms of the gain.

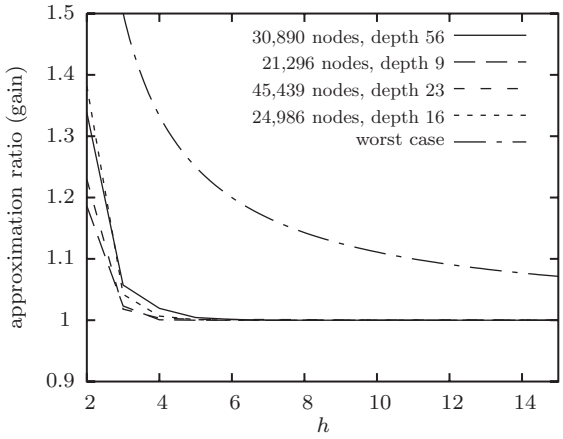

Fig. 8. Approximation ratio resulted from running LPATH on tree instances for different values of h.

4.2 Runtime

The runtime of LPATH , as expected, grows exponentially with h, up to some characteristic value (Figure 9). For h larger than that value, the runtime remains constant. The characteristic value depends on the tree instance. It is typically slightly smaller than the tree's depth, when there are only few possibilities for long hotlinks.

The runtimes for processing the synthetic instances by the other algorithms are depicted in Figures 10 and 11. Apparently, for each algorithm, the runtime values form a point cloud that is quite compact, that is, the runtimes are reliable. The only exception is GREEDY, having many outliers above its typical runtime. CENTIPEDE is by far the slowest algorithm, and it is also the only method whose runtime is clearly superlinear in practice. It is followed by algorithms PMIN, H/PH, and GREEDY. The fastest method is HEAVY PATH.

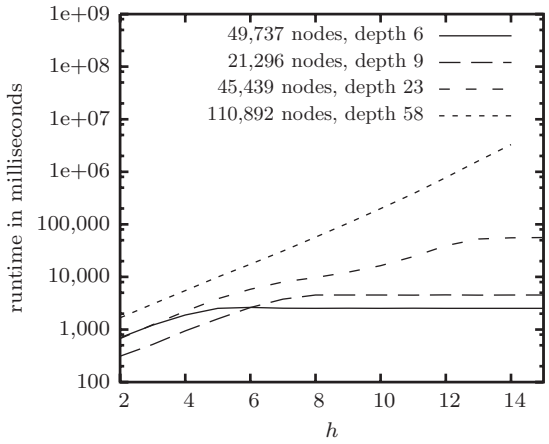

Fig. 9. Runtime resulted from running LPATH on tree instances for different values of h.

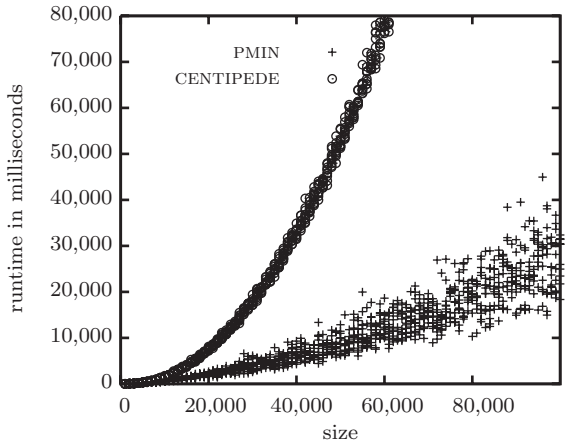

Fig. 10. Runtime of the approximation algorithms.

5. SUMMARY AND CONCLUSION

In this article, we have studied the practical behavior of six different hotlink assignment algorithms for which different kinds of approximation guarantees are known. Our experiments were conducted using both synthetic instances and real Web site trees. We summarize the most important findings of our study.

Algorithm PMIN performs excellent on all instances. It is also easy to implement, but its running time is quite high compared to other strategies.

The GREEDY strategy also exhibits a very good performance. It is slightly worse than PMIN in practice, but the guaranteed approximation ratio of 2 concerning the gain makes this algorithm a good choice. It is also easy to implement and runs faster than PMIN.

Although the path length seems to be the more natural optimization term, strategies tailored to approximate it are not the first choice in practice. The

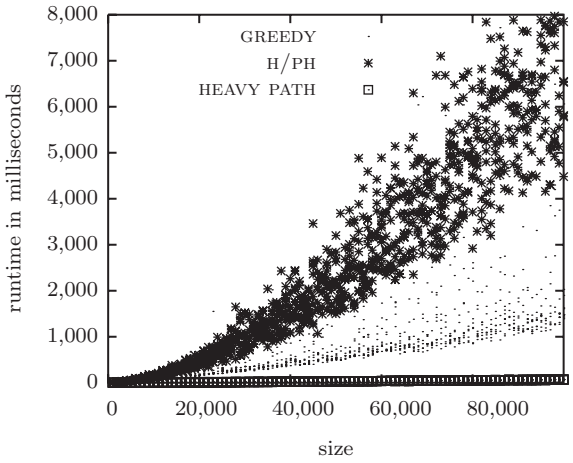

Fig. 11. Runtime of the approximation algorithms.

CENTIPEDE algorithm has a very high running time. As the quality of its solutions is only moderate, this algorithm is only advisable in practice if one wants a guaranteed approximation factor of 2 for the path length.

Algorithms H/PH and HEAVY PATH exhibit the worst performance among all algorithms studied in this work regarding solution quality. It sometimes happens that these strategies do not assign any hotlink at all, so in applications, one would probably implement a variant that always assigns some hotlinks. Anyhow, HEAVY PATH is extremely fast in practice and has the best worst-case runtime, making it interesting for situations where efficiency is more important than solution quality.

For small values of h, the assignments computed by LPATH are also not as good as the results of GREEDY and PMIN. This PTAS is only recommendable in situations where the solution quality is much more important than the resource requirements are. Parameter h should then be set to a value larger than 3.

Note that it is always possible to combine the properties of several algorithms. For example, by running PMIN, GREEDY, and CENTIPEDE and then choosing the best among the three resulting assignments, one would obtain a solution having the quality of PMIN with a guaranteed approximation ratio of 2 in terms of the path length and the gain.

ACKNOWLEDGMENTS

The author wishes to thank Críston de Souza and all authors of Gerstel et al. [2007] for making their test data set available. The numerous helpful comments provided by each of the anonymous referees of the ACM Journal on Experimental Algorithmics also deserve sincere thanks.

REFERENCES

BARABÁSI, A. L. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Science 286*, 5439, 509–512.

Bose, P., Kranakis, E., Krizanc, D., Martin, M. V., Czyzowicz, J., Pelc, A., and Gasieniec, L. 2000. Strategies for hotlink assignments. In *Proceedings of the International Symposium on Algorithms and Computation*. Springer, Berlin, 23–34.

Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., and Martin, M. 2001. Evaluation of hotlink assignment heuristics for improving Web access. In *Proceedings of the 2nd International Conference on Internet Computing*. CSREA Press, 793–799.

Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., and Martin, M. V. 2003. Enhancing hyperlink structure for improving Web performance. *J. Web Eng. 1*, 2, 93–127.

Douïeb, K. and Langerman, S. 2005. Dynamic hotlinks. In *Proceedings of the 9th Workshop on Algorithms and Data Structures*. Springer, Berlin, 182–194.

Douïeb, K. and Langerman, S. 2006. Near-entropy hotlink assignments. In *Proceedings of the 14th Annual European Symposium on Algorithms*. Springer, Berlin, 292–303.

Gerstel, O., Kutten, S., Laber, E. S., Matichin, R., Peleg, D., Pessoa, A. A., and Souza, C. 2007. Reducing human interactions in Web directory searches. *ACM Trans. Inf. Syst. 25*, 4, 20.

Gerstel, O. O., Kutten, S., Matichin, R., and Peleg, D. 2003. Hotlink enhancement algorithms for Web directories: Extended abstract. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation*. Springer, Berlin, 68–77.

Jacobs, T. 2007. Constant factor approximations for the hotlink assignment problem. In *Proceedings of the 10th Workshop on Algorithms and Data Structures*. Springer, Berlin, 188–200.

Jacobs, T. 2008. On the complexity of optimal hotlink assignment. In *Proceedings of the 16th European Symposium on Algorithms*. Springer, Berlin, 540–552.

Kranakis, E., Krizanc, D., and Shende, S. 2004. Approximate hotlink assignment. *Inf. Process. Lett. 90*, 3, 121–128.

Matichin, R. and Peleg, D. 2007. Approximation algorithm for hotlink assignment in the greedy model. *Theor. Comput. Sci. 383*, 1, 102–110.

Pessoa, A. A., Laber, E. S., and de Souza, C. 2004a. Efficient algorithms for the hotlink assignment problem: The worst-case search. In *Proceedings of the 15th Annual International Symposium on Algorithms and Computation*. Springer, Berlin, 778–792.

Pessoa, A. A., Laber, E. S., and de Souza, C. 2004b. Efficient implementation of hotlink assignment algorithm for Web sites. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments and the 1st Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALC'04)*. SIAM, Philadelphia, 79–87.

Pitkow, J. E. 1999. Summary of www characterizations. *World Wide Web 2*, 1-2, 3–13.

Sleator, D. D. and Tarjan, R. E. 1983. A data structure for dynamic trees. *J. Comput. Syst. Sci. 26*, 3, 362–391.

Received August 2008; revised November 2009; accepted November 2009