Journal of Web Engineering, Vol. 1, No. 2 (2003) 093–127
© Rinton Press

ENHANCING HYPERLINK STRUCTURE FOR IMPROVING WEB PERFORMANCE

JUREK CZYZOWICZ[1], EVANGELOS KRANAKIS[2], DANNY KRIZANC[3]
ANDRZEJ PELC[1], MIGUEL VARGAS MARTIN[2]

[1] *Département d'Informatique, Université du Québec en Outaouais*
Hull, Québec J8X 3X7, Canada
E-mail: {jurek.czyzowicz,pelc}@uqo.ca

[2] *School of Computer Science, Carleton University*
Ottawa, Ontario, K1S 5B6, Canada
E-mail: {kranakis,mvargas}@scs.carleton.ca

[3] *Department of Mathematics, Wesleyan University*
Middletown, Connecticut, 06459, USA
E-mail: dkrizanc@caucus.cs.wesleyan.edu

Received February 20, 2003
Revised March 18, 2003

In a Web site, each page v has a certain probability p_v of being requested by a user. The access cost of a Web site is the sum of $p_v \cdot c(r,v)$, of every page v, where $c(r,v)$ is the cost of the shortest path between the home page, r, and page v. The cost of a path is measured in two ways. One measure is in terms of its length, where the cost of the path is simply the number of hyperlinks in it. The other measure is in terms of the data transfer generated for traversing the path. This research work concerns the problem of minimizing the access cost of a Web site by adding hotlinks over its underlying structure. We propose an improvement on Web site access by making the most popular pages more accessible to users. We do this by assigning hotlinks to the existing structure of the Web site. The problem of finding an optimal assignment of hotlinks is known as the hotlink assignment problem. We present heuristic algorithms which are tested and compared by simulation on real and random Web sites. We develop The Hotlink Optimizer (HotOpt), a new software tool that finds an assignment of hotlinks reducing the access cost of a Web site. HotOpt is empowered by one of the algorithms presented in this paper.

Keywords: Web access optimization, hotlinks, data transfer
Communicated by: B White, R Baeza-Yates & C Watters

1 Introduction

There are many factors, from physical to logical, which affect the speed of information retrieval from the Internet. Examples include the efficiency of the underlying data transmission lines and the protocols that govern their usage; the physical location of the information and the efficiency of the Web browsers which locate it.

Continuing efforts are being made in order to improve the performance of the Internet. Some of the most important areas of research are Web site design, clustering, and caching. We believe that a well designed Web site contributes to the improvement of the Internet since

well structured sites lead to less traffic on the Web, as users are getting the information they want without having to traverse superfluous Web pages. In addition, well designed Web sites become more attractive to users since they offer rapid access to information.

1.1 Our Approach: Hotlink Optimization

Unfortunately, it is common that the users and designers of a Web site perceive the Web site in a different way. This discrepancy is reflected in users having to traverse "costly" paths in order to reach the pages they are interested in. We say that a path is costly either because it is "too" long or because the pages in it are "too" big (in bytes).

We endeavour to improve Web access by improving the design of Web sites. A well designed Web site will avoid some useless traffic, save time to users, and reduce the Web server work load. Our idea is conceptually simple, "bring the most popular pages closer to the home page."

We propose an improvement on Web site access by adding hotlinks that provide shortcuts to the most popular pages. Suppose that there is a page with many access hits; we want this page to be closer to the home page so users can reach it at a lower cost. Cost may be measured in terms of the expected number of steps or in terms of the expected data transfer. Figure 1 illustrates the idea.

It would be interesting to measure the cost of a Web site in terms of its *latency*. Latency is a measure of network performance that "corresponds to how long it takes a single bit to propagate from one end of a network to the other" ([1], page 23). The latency of a Web site is the average of the latencies of its Web pages. The latency of a Web page is the time elapsed from the time when the user requests a page on his or her navigator to the time when the page is completely displayed on the screen. From this definition of latency, we can say that the latency of page v is smaller for user A, who has a high speed Internet connection, than for user B, who has a slower one. The latency of a Web page experienced by users A and B also depends on other factors including their physical locations and the traffic on the transmission lines. Due to the fact that the latency of a Web page depends on a combination of many factors that vary from one user to other, it is extraordinarily complex to design a realistic model for optimizing the latency of a Web site. Nevertheless, by optimizing the expected number of steps (or the expected data transfer) of a Web site, we significantly decrease latency. See the example illustrated in Figure 1.

1.2 Related Work

Since its appearance, the Internet has been the subject of arduous study. Until the last few decades, when the amount of information on the Internet started to expand at unprecedented rates and with unknown patterns, it was never so urgent to organize the content. Perhaps, the unregulated growth of the Internet is a result of the different socio-economic interests that interact on it. As an entity that is not well-understood, consisting of a solid source of information and communication for the whole world, the Internet has become an attractive field of study to researchers.

Researchers have focused their attention on the optimal design of Web sites based on user access patterns. Some of them suggest analyzing user access patterns to help design better Web pages, sites, and browsers. Catledge et al. [2] and Drott et al. [3] analyze user access patterns to suggest improvements that help to design better Web pages. For example,

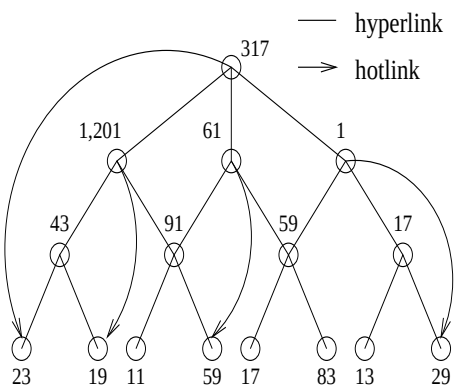

Fig. 1. Example of an assignment of one hotlink per page. The figure shows a small Web site modelled with a directed graph (edges going downward). The sizes of the Web pages are in KBytes. To see how a hotlink assignment reduces the latency of a Web site by reducing the average number of steps, suppose that a user at the home page (at the top) wishes to reach the Web page of 23KBytes (left most page). Without the hotlink, the user would have to download the two intermediate pages between this page and the home page.

Catledge et al. find that users rarely traverse a path of more than two hyperlinks before returning to the starting point. This observation would suggest to create dense Web sites. Pirolli et al. [4] propose to create aggregation of Web pages according to their importance or their content.

Perkowitz et al. [5] propose the design of adaptive Web sites by promoting and demoting pages, highlighting hyperlinks, adding hyperlinks and clustering related pages. Perkowitz et al. [6] present an algorithm that analyzes user access logs in order to identify candidate hyperlink clusters to be included in index pages. The performance of the algorithm is measured according to the quality of the clusters; specifically, they assess the quality of a cluster by answering the following question: Given a visit to a page of a cluster, what percentage of the pages in this cluster was visited by this user? They find that when the number of clusters is 1, the percentage of visits is approximately 72; but when the number of clusters is 10, the percentage of visits is around 20. They also compare clusters constructed by a human with clusters constructed by their algorithm and find that the algorithm constructs clusters with at least 15 more visits than the human-authored ones. Note that the approach of Perkowitz et al. finds clusters of related documents and creates index pages to those documents, however, this solution does not specify where the index pages are to be inserted in the Web site.

Spiliopoulou et al. [7] present a tool for detecting "interesting" commonly traversed paths. They suggest the use of this information to improve Web site design but do not suggest a specific mechanism.

Nakayama et al. [8] propose a technique to detect the gap between Web designer's expectations and users' behaviour. The former is assessed based on the content of Web pages, whereas the latter is assessed by analyzing user navigation patterns. The resulting gap suggests (without specifying) the possibility for improvements to the Web site based on the criteria of the Web designer. These improvements may be on the hyperlink topological structure or on the page layout. The statistical analyses used to assess Web designers' expectations and users'

behaviour are suitable for evaluating the improvements without involving actual users.

Fu et al. [9] propose an algorithm to reduce the number of steps to reach the most popular pages of a Web site. They classify the Web pages according to the number of hyperlinks in them into "index pages" and "content pages". Based on this classification and on the popularity of Web pages, the authors promote and demote pages to reduce the number of steps. Fu et al. test their approach in a particular Web site. They show experimentally that their approach actually "reduces" the number of steps required to reach popular pages. Their algorithm requires the empirical adjustment of parameters used in their classification of Web pages.

Srikant et al. [10] propose improving Web site design by finding the pages whose actual location is different from their expected locations, i.e. where visitors expect to find them. Their algorithm relies on the belief that when the user presses the back button of the navigator it is because the user did not find the page where he or she had expected to find it. The algorithm of Srikant et al. is tested only in a particular Web site. They find that "many" pages are wrongly placed, according to their criteria.

Some researchers, e.g. [11, 13, 14], have studied the assignment of shortcuts in full binary tree structures.

1.3 General Notation and Terminology

A *Web site* is a collection of Web pages administered by the same authority which are linked together to form a unified source of information. We say that two Web pages are connected by a *hyperlink*, which is a one-way linkage between two pages. The notion of a *home page* is needed to understand the organization of a Web site. The home page is considered to be the starting point of any Web site, and it is assumed that any Web page belonging to the Web site can be reached from the home page. Under this assumption, we can say that within a site, a Web page b is a *descendant* of a Web page a if there is a path of hyperlinks leading from a to b. Therefore, any Web page is a descendant of the home page. This research focuses on improving the design of Web sites by assigning hotlinks (shortcuts) to the collection of Web pages. A *hotlink* is defined as an added hyperlink that links a Web page to a descendant of that page.

In a Web site, each page v has a certain probability p_v of being requested by a user. The *access cost of page* v, is denoted $c(r, v)$, and corresponds to the cost of the shortest path between the home page, r, and page v. The *access cost of a Web site* is the sum of the access cost of all its pages. The cost of a path is measured in two ways. One measure is in terms of its length, where the cost of the path is simply the number of hyperlinks in it. The other measure is in terms of the data transfer generated by the path, i.e., the number of bytes that need to be transferred in order to traverse the path. The problem is to minimize the access cost of a Web site by adding hotlinks to its underlying structure. An immediate intuitive solution to the problem would be to add as many hotlinks as necessary to connect directly the home page with every other page of the Web site. However, from a practical point of view, this solution could produce a Web site without semantic structure and with a very dense home page, that would be difficult to visualize and understand by users. Therefore, we must restrict our problem to assigning at most k hotlinks per page. *The problem is to minimize the access cost of a Web site by adding at most k hotlinks per page.* This is a very difficult

problem, in fact, some instances of it have been proven to be NP-hard (see [11]).

Consider a part of the Web called a *Web site*, consisting of a collection $V = \{v_1, \ldots, v_N\}$ of Web pages connected by hyperlinks. These hyperlinks have been placed a priori by design in the initial construction of the Web pages. Assume there exists a directed path of hyperlinks from the *home page* r to any other page of the collection. We can view the Web site as a directed graph $G = (V, E)$, where each page is represented by a node, and each hyperlink is represented by an edge.[a] The number of hyperlinks in page v is called *outdegree*, and is denoted by δ_v. The *maximum degree* of all the pages of a Web site is denoted by δ.

Consider a tree $T = (V, E')$, where $E' \subseteq E$, with a distinguished node called the root, r. We define the *distance* from the root r to a node $v \in V$, denoted by $d(v)$, as the number of edges between them.

Suppose that the leaves of T have a probability distribution p over them. We assign popularities[b] to the internal nodes in a bottom-up fashion, in such a way that the weight of a node is equal to the sum of the probabilities of the leaves descendant to it. Observe that in this way, the root node will have a weight of 1. Thus, let us say that node v has weight p_v, then we define the *access cost of a Web site* T, as follow:

$$E[T] = \sum_{v \text{ is a leaf}} p_v \cdot c(r, v). \tag{1}$$

The *optimal k-hotlinks assignment problem* consists in minimizing Eq. 1 by adding at most k hotlinks from each node of the tree. If $k = 1$ we call it *optimal hotlink assignment problem.*

As mentioned before, the cost of a path can be measured in two ways. One measure is in terms of the number of steps, where the cost of the path is the number of hyperlinks in it. The other measure is in terms of the data transfer generated for traversing the path. We use a slightly different notation and terminology for these two measures.

Optimizing Number of Steps

Consider a *hotlink* $h = (s, t)$, added to the original tree.[c] We say that s is the *hyperparent* of t, and t is the *hyperson* of s. Consider a set of hotlinks, H, assigned to the tree T; the resulting graph is denoted by T^H, and the *gain* of H is defined by

$$\mathcal{G}(H) = E[T] - E[T^H]. \tag{2}$$

The gain of a single hotlink, $h = (s, t)$, is defined by

$$g(h) = p_t(d(t) - d(s) - 1). \tag{3}$$

A set of hotlinks H is optimal if $\mathcal{G}(H) \geq \mathcal{G}(H')$ for any hotlink set H'.

Optimizing Data Transfer

Define the *weight* (in bytes) of a page v, w_v, as its own size plus the size of its embedded files. The *access cost of a page* v, $w(v) = c(r, v)$, is equal to the sum of the weights of the pages contained in the shortest path between the home page r and v.

[a] Throughout this paper we use interchangeably the terms *root* and *home page*, *node* and *Web page*, *edge* and *hyperlink*.

[b] The terms *probability* and *popularity* will be used interchangeably throughout this paper.

[c] Literals s and t stand for *source* and *target* nodes.

The *gain* of a single hotlink, $h = (s, t)$, is now defined by

$$g(h) = p_t(w(t) - w(s) - w_t). \tag{4}$$

Define $\Pi^H(r, u)$ as the shortest path from r to u given the hotlink assignment H. We have the following:

$$w^H(u) = \sum_{v \in \Pi^H(r,u)} w_v. \tag{5}$$

$$E_w[T^H] = \sum_{i \text{ is a leaf}} w^H(i) \cdot p_i. \tag{6}$$

$$\mathcal{G}(H) = E_w[T] - E_w[T^H] = \sum_{i \text{ is a leaf}} (w(i) - w^H(i)) \cdot p_i. \tag{7}$$

A set of hotlinks H is optimal if $\mathcal{G}(H) \geq \mathcal{G}(H')$ for any hotlink set H'.

1.4 Contributions of the Paper

In this paper we make several contributions. We consider some variants of the problem, namely, one hotlink per page and multiple hotlinks per page. Hotlink assignment algorithms are presented, and tested with simulations. In particular, we use simulated Web sites with simulated access patterns, real Web sites with simulated access patterns, and a real Web site with real access patterns. We develop the Hotlink Optimizer, a hotlink assignment software tool. In this section we summarize our main contributions.

1.4.1 Hotlink Assignments to Web Sites

We present algorithms for one and multiple hotlinks per page, which were tested by simulations.

For the case of one hotlink per page, among the algorithms we have tested, algorithm *greedyBFS* achieves the best performance. The results of our experiments reveal that algorithm *greedyBFS* is capable of saving up to 27% the average number of steps on simulated Web sites with simulated access patterns, 35% on real Web sites with simulated access patterns, and 27% on a particular Web site with real access patterns. Regarding data transfer, algorithm *weighted-greedyBFS* is capable of saving up to 14, 30, and 22%, respectively.

For the case of multiple hotlinks per page, we test algorithms *k-greedyBFS* and *greedyBFS*^{k} on simulated Web sites with simulated access patterns, and find that algorithm *k-greedyBFS* outperforms algorithm *greedyBFS*^{k} for $k \leq 5$. For $k = 5$, the algorithms are capable of saving up to 43% on the expected number of steps of a Web site.

1.4.2 The Hotlink Optimizer

We have developed the Hotlink Optimizer (HotOpt), a powerful software tool that assists Web administrators and designers in re-structuring their Web sites according to the needs of users.

By analyzing the hyperlink structure of a Web site, and taking into consideration the access patterns of the users, HotOpt is able to suggest a set of hotlinks, H, and thus save a certain proportion of the access cost of the Web site. While the inputs are the *home page file*

and the *access log files*, the output is a set of hotlinks H along with $x\%$, the proportion of gain achieved by H. Figure 16 shows the user interface of HotOpt.

1.5 Outline of the Paper

In Section 2 we present heuristic hotlink assignment algorithms and the simulation process. In Section 3 we introduce the Hotlink Optimizer, a software tool that assists Web administrators and designers by suggesting an assignment of hotlinks. Finally, in Section 4 we discuss the conclusions and some possible extensions to our work.

2 Hotlink Assignment Algorithms and Simulations

In this section we present our hotlink assignment algorithms (we also describe an algorithm by Kranakis et al. [12] called *approximateHotlinkAssignment*) and their performance when tested on random and real Web sites.

Before presenting the algorithms we explain the process of the simulations. We evaluate the algorithms on three different kinds of structures: random Web sites, real Web sites with unknown access probability distribution, and an actual Web site with a known access probability distribution.

2.1 The Simulation Process

In order to generate realistic random Web sites, we need to know how the pages of a Web site are linked together. We also need to know how users access the Web sites in order to emulate the popularity of the Web pages. For each Web page, we generate an outdegree δ and an access probability p according to theoretical distributions. Even though the distributions of δ and p are precise, they do not provide any information on how they are to be distributed among the pages of the Web site. Given the huge universe of possible combinations of δ and p, we obtain 95% confidence intervals for all the results that we present. In order to refer to the proportion of gain achieved by an algorithm we use the term *average proportion of gain*, which is computed by averaging the proportion of gain obtained in each single sample of the simulations. Details on Web structure, such as the outdegree of home pages, are not taken into consideration in the simulations. This may affect the approximation of our random Web sites to the real ones, however, we validate the correctness of the simulations regarding topology and popularity by showing that the outdegree δ and the popularity p of the random Web sites actually follow the theoretical probability distributions.

2.1.1 Random Web Sites

Faloutsos et al. [15] point out that Internet topology can be modelled using power-law relationships. They compare three snapshots of the Internet with power-laws and observe a strong similarity despite the growth of the Internet between the snapshots. In particular, we are interested in the power-law that governs the outdegree δ of a Web page. The probability that a page has degree i is proportional to i^{-k}, for $k > 1$.

Broder et al. [16] have found that $k = 2.72$ is an accurate value for the outdegree. Their experiments are based on a sample of over 200 million pages and 1.5 billion hyperlinks.

Given a Web page v, the probability that v has outdegree i is determined by the formula

$$P[\delta_v = i] \sim i^{-2.72}. \tag{8}$$

2.1.2 Generation of Random Web Sites

There are many techniques for generating random graphs. Waxman [17] proposes a random graph generator for modelling geographical networks. Calvert et al. [18] discusses how graph-based models can be used to generate large graphs with specific parameters of locality and hierarchy. Zegura et al. [19, 20] survey the generation of commonly used graph models. Zegura et al. also study the difficulty of generating random graphs with a particular average degree and then propose a technique to generate such graphs. Aiello et al. [21] describe a random graph model for reproducing sparse gigantic graphs with particular degree sequences. The Aiello et al. model fails to reproduce some characteristics of real networks. Watts [22] fully explains the concept of "small world" graphs, which are clustered, sparse, and of small diameter. There is an extensive literature about small worlds, e.g., [23, 24]. Hayes [25, 26] analyzes gigantic graphs such as the Web, and maintains that the Web is a small world graph. Hayes points out how lattices and Erdös-Rényi graphs fail to reproduce small world graphs since lattices do not have small diameter and Erdös-Rényi graphs are not clustered.

We generate random Web site trees $T = (V, E)$ of size $s = |V|$. The outdegree of a page is taken from a sample of outdegrees generated by Formula 8.

The generation of a random Web site tree structure is described in Algorithm 1. In Section 2.1.4, we discuss the assignment of access probabilities to the random Web site trees.

Algorithm 1 hyperlinksStructure(*size*)

---

1. $V = E = \phi$
2. *let* q *be an empty queue*
3. v =*new Web page*
4. $V = V \cup v$
5. *enqueue* v *in* q
6. *while* q *is not empty*
   (a) *if* $|V|$ = size *then return* $T = (V, E)$
   (b) v =*dequeue* q
   (c) *assign a random outdegree* δ_v *to* v (see Section 2.1.1)
   (d) *for* $i = 1$ *to* δ_v
      i. w =*new Web page*
      ii. $V = V \cup w$, $E = E \cup (v, w)$ *(i.e., insert in page* v *a new hyperlink pointing to a new Web page* w*)*
      iii. *enqueue* w *in* q
      iv. *if* $|V|$ = size *then return* $T = (V, E)$

2.1.3 Actual Web Sites

In Section 2.1.2, we briefly survey the efforts to generate random graphs with particular characteristics [17, 18, 19, 20, 21, 25, 26]. We know of no precise technique for simulating "typical" Web sites.

Instead of simulating random Web sites, we could take a sample of Web sites from the Internet. Henzinger et al. [27] suggest performing random walks on the Internet in order to extract representative samples of Web pages. They point out that with this random walk, pages with big indegree will have more chances to be visited than pages with small indegree, which may not be true in reality.

In the absence of efficient techniques for generating Web sites, we test our algorithms with the hyperlink structure[d] of the Web sites of eleven universities in Ontario. Once we retrieve the hyperlink structure of a Web site, we convert it into a directed graph $G = (V, E)$, where each vertex $v \in V$ represents a Web page, and every edge $(v, w) \in E$ represents a hyperlink going from v to w. We then generate a tree $T = (V, E')$ by performing breadth first search on G, starting with the home page. In Section 2.1.4 we study the assignment of access probabilities to these Web sites.

2.1.4 Web Access Distribution

To perform the simulations, we need to assign access probabilities to the Web sites. This information can be extracted from the access log files. However, sometimes the log files are unavailable, either because the Web administrators prefer not to disclose them or because the Web site structure has been simulated with a random graph. We explain how to assign access probabilities to Web sites with and without the log files.

Modeling Access Distribution

While it is possible to get the hyperlink structure of any Web site, it is not easy to convince a Web site administrator to disclose log files in order to know the popularity of the Web pages. When access logs are not available, we simulate the popularity of a Web page using the Zipf's popularity law.

Given Zipf's observations of human behaviour [28], one could conjecture that the popularity of Web pages obeys Zipf's popularity law. Glassman [29] prove experimentally that the popularity of Web pages can actually be modelled with Zipf's popularity law. Glassman analyzes 100,000 requests from 300 different users of 40,000 Web pages. Zipf's popularity law is also found in the length of user's navigation. Levene et al. [30] show experimentally that Zipf's popularity law explains user's navigation, in the sense that longer navigations are less probable than shorter ones. Pitkow [31] provides a good survey on Web characterizations.

In Zipf's distribution, the probability of the *i-th* most probable item is $p_i = \frac{1}{iH_m}$, where H_m is the harmonic number, $H_m = \sum_{j=1}^{m} \frac{1}{j}$ [32]. Thus, given the *i-th* most popular Web page v of a Web site of m pages,

$$p_v = \frac{1}{iH_m}. \tag{9}$$

In order to associate popularities to the m leaves of a Web site T, we generate a popularity

[d] Note that the hyperlink structure of a Web site is different from its file structure.

ranking-list of m elements based on Eq. 9, where the *i-th* most popular leaf page has popularity $p_i = \frac{1}{iH_m}$. Each of these popularities is randomly associated with a different leaf of T. This popularity represents the access probability of a page. The access probability of an intermediate (i.e., non-leaf) page is the sum of the access probabilities of the leaves descendant to it, in such a way that the home page has access probability of 1. The assignment of access probabilities to the Web pages of a Web site T is described in Algorithm 2.

Algorithm 2 accessProbabilities($T = (V, E)$)

1. *create a popularity ranking-list* rankingList *of length m, where m is the number of leaf pages of* T (see section 2.1.4)
2. *for each leaf page* v *of* T
   (a) *take a random element from* rankingList, p *(not taken before), and make* $p_v = p$
3. *for each intermediate page (i.e., non-leaf)* v
   (a) p_v *is equal to the sum of the leaf pages descendant to* v

2.1.5 The Distribution of File Sizes

Arlitt et al. [33] carefully analyze the traffic of a Web server. They prove experimentally that the file size distribution is *heavy tailed* (see [34] for a background of heavy tail models). The assertion of Arlitt et al. is confirmed by Crovella et al. [35], who show experimentally that file sizes greater than about $1,000$ bytes can be well modelled with a Pareto distribution. Barford et al. [36] show experimentally that file sizes can be modelled with a hybrid distribution. Files of "small" size can be well modelled with a lognormal distribution, whereas "big" files can be modelled with Pareto distribution. According to Barford et al., the body of the distribution is modelled with lognormal distribution for values smaller than 133KBytes. Downey [37] creates a model that suggests that file sizes are modelled only by a lognormal distribution. Mitzenmacher [38] points out why the arguments of Downey yield only a lognormal distribution. Mitzenmacher corroborates the results of Barford et al. and suggests a double Pareto distribution for the body of the curve and a Pareto distribution for the tail.

Assuming that the model proposed by Barford et al. [36] is correct, we create a file size distribution as follows:

$$P[w_v = x] = \begin{cases} \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2/2\sigma^2} & \text{if } x < \text{cutoff point} \\ \alpha k^\alpha x^{-(\alpha+1)} & \text{otherwise,} \end{cases} \tag{10}$$

where w_v is the size in bytes of page v and its embedded files. The values of μ, σ and α are taken from the observations of Barford et al. [39] on the requests of over $40,000$ files from over 500 users in 1998. The values of k and the *cutoff point* are calculated to fit the results of Barford et al. [39] who observe that 83% of the files fall in the body of the distribution. See Figure 14. The values used in Formula 10 are displayed in Table 1.

Table 1. Parameters used in Formula 10.

parameter	value
μ	7.796
σ	1.625
k	8,863
α	1.470
cutoff point	10,790

For the case of random Web sites, we assign weights to the pages in a random fashion, according to the distribution of Formula 10.

The straightforward solution for getting the actual weights of real Web sites would be to download the entire Web sites, however this action would require enormous storage capacity. One alternative would be to process the information contained in the log files, however the log files are usually undisclosed to the public. Therefore we use random file sizes. We search the number of embedded files on every page and for each of them we withdraw a size from the distribution given by Formula 10.

2.2 Algorithm simpleBFS

Algorithm *simpleBFS* iteratively assigns hotlinks in breadth first search order, starting with the home page. Consider a hotlink (s,t). In each iteration of the algorithm, s corresponds to the next node in the breadth first search order and t corresponds to a descendant of s that offers the biggest gain. The algorithm stops when there are no more possible hotlinks to assign. Algorithm *simpleBFS* uses the function *next_in_BFS_order*, which returns the next node of the tree in breadth first search order, starting from the home page.

Algorithm 3 simpleBFS(T)

1. $H = \phi$
2. *while(* $(\ s = next\_in\_BFS\_order\ ) \neq \phi$ *)*
   (a) $t = v$: *v maximizes Eq. 3; and v is descendant of s; and v does not have a hyperparent*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s,t)\}$

In a variant of algorithm *simpleBFS,* called *greedyBFS,* the target node can not be a descendant of a node that already has a hyperparent. The only modification of *simpleBFS* occurs in step 2a.

2.3 Algorithm greedyBFS

Algorithm *greedyBFS* assigns hotlinks iteratively in breadth first search order starting from the home page. Consider a hotlink (s,t). In each iteration of the algorithm, s corresponds to the next node in breadth first search order, and t corresponds to the descendant of s that maximizes the gain, but is not a descendant of a node x, which is at a higher level than s and

already has an incoming hotlink[e] See Figure 2. The algorithm stops when there are no more possible hotlinks to assign. We also have a recursive version of algorithm *greedyBFS*, which we call *recursive*. This recursive version will help us to validate the simulations, as we will see later.

Algorithm 4 greedyBFS(T)

1. $H = \phi$
2. *while(* $(\ s = next\_in\_BFS\_order\ ) \neq \phi$ *)*
   (a) $t = v$: *v maximizes Eq. 3; and v is a descendant of s; and v does not have a hyperparent; and v is not a descendant of a node x, which is at a higher level than s and already has a hyperparent.*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s,t)\}$

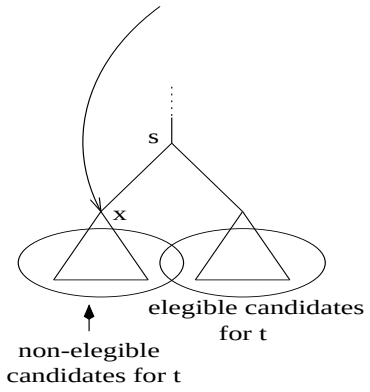

Fig. 2. In algorithm *greedyBFS*, two hotlinks are not allowed to cross each other. Consider a hotlink (s,t) to be assigned in an iteration of the algorithm. t must be a descendant of s that minimizes the cost but is not a descendant of a node x, which is at a higher level than s and already has an incoming hotlink.

By comparing these two algorithms we will see that in practice, *greedyBFS* offers the same or better performance than *simpleBFS*. This claim comes from the assumption of "obvious navigation". Obvious navigation consists in taking always the hyperlink or hotlink taking us closer to the desired page. This is the natural way users navigate on the Web. Note that in some cases this kind of navigation can be inefficient. SeeFigure 3.

If we took two hotlinks from an arbitrary path from the root to a leaf after running *simpleBFS*, one of the following four cases would hold: a) one hotlink is "inside" the other, or b) the two hotlinks are overlapped, or c) one hotlink starts exactly at the end of the other, or d) otherwise. These four cases are illustrated in Figure 4.

Observe that obvious navigation can be inefficient only in case b). Furthermore, the

[e] The levels of the tree are counted in increasing order starting from the root, such that the root is at the lowest level.

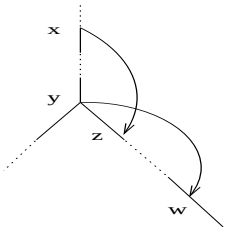

Fig. 3. Illustration of *"obvious navigation"* assumption. Suppose that a user is at page x and wishes to reach a page descendant of w. We assume that, as the user does not have a map of the site, the path $(x, z) + z \to w$ will be followed, even though path $x \to y + (y, w)$ would have been shorter. Therefore, obvious navigation is efficient only when $x \to y \geq z \to w$.

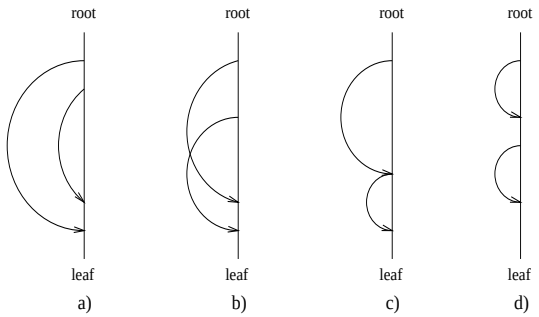

Fig. 4. Four possible scenarios of two hotlinks on a path, after executing algorithm *simpleBFS*. Each figure shows an arbitrary path from the root to a leaf. a) one hotlink is "inside" the other, b) the hotlinks are overlapped, c) one hotlink starts exactly at the end of the other, d) otherwise.

deepest hotlink will be wasted in this case and the source node will be unable to accommodate an efficient hotlink from it. Case b) never happens in algorithm *greedyBFS*. Therefore, under obvious navigation assumption, algorithm *greedyBFS* performs at least as well as algorithm *simpleBFS*.

2.3.1 Performance of Algorithm greedyBFS

Algorithm greedyBFS *Evaluated on Random Web Sites*

Algorithm *greedyBFS* is described in Section 2.3. We show the results of the simulations in Figure 5 and Table 2. We plot the average proportion of gain that can be attained by the algorithm. The average proportion of gain has a 95% confidence interval of at most ∓ 1.27. Observe that the gain of the algorithm oscillates between narrow intervals, which indicates that the size of the tree does not greatly affect the performance.

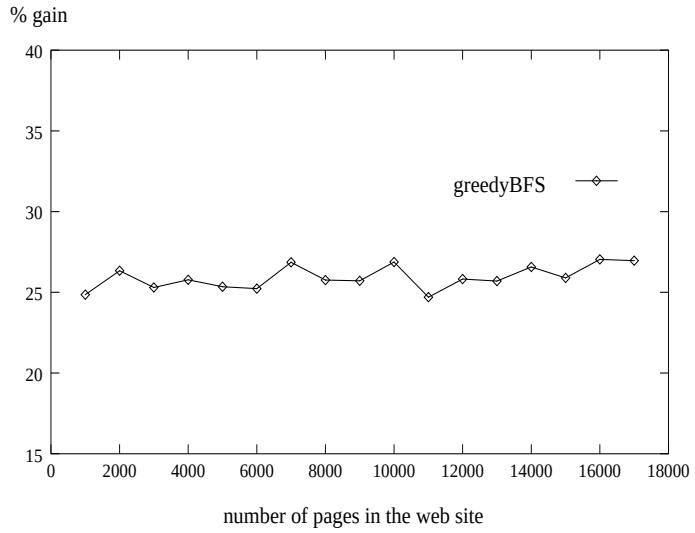

Fig. 5. Gain on the access cost obtained by applying *greedyBFS* to randomly generated Web sites of size $1,000$ to $17,000$. The average proportion of gain has a 95% confidence interval of at most ∓ 1.27.

Algorithm greedyBFS *Evaluated on Real Web Sites*

Algorithm *greedyBFS* is discussed in Section 2.3. The results of the simulations are displayed in Table 3 and depicted in Figure 6. The average number of nodes of the sample of Web sites is $9,669.2$. Observe that the proportion of gain stabilizes after assigning less than $1,000$ hotlinks, which is approximately 10% of the average number of Web pages.

2.3.2 Hotlink Assignments to the scs.carleton.ca *Domain*

We are able to use the *scs.carleton.ca* domain for testing, since we have access to both its hyperlink structure and its access logs, which contain information about the popularity of the Web pages.

Table 2. Proportion of gain on the access costs of randomly generated Web sites of $1,000$ to $17,000$ pages. The proportion of gain has a 95% confidence interval of at most ∓ 1.27.

size of Web site, $\|V\|$	% of gain	standard deviation	95% confidence interval
1,000	24.85	2.46	0.85
2,000	26.34	3.16	1.10
3,000	25.30	3.21	1.11
4,000	25.77	3.67	1.27
5,000	25.35	2.18	0.75
6,000	25.23	2.45	0.85
7,000	26.86	1.94	0.67
8,000	25.76	2.31	0.80
9,000	25.71	3.10	1.07
10,000	26.88	3.40	1.18
11,000	24.70	2.38	0.83
12,000	25.82	3.61	1.25
13,000	25.70	2.86	1.00
14,000	26.57	2.59	0.90
15,000	25.89	2.52	0.87
16,000	27.04	2.90	1.00
17,000	26.96	2.49	0.86

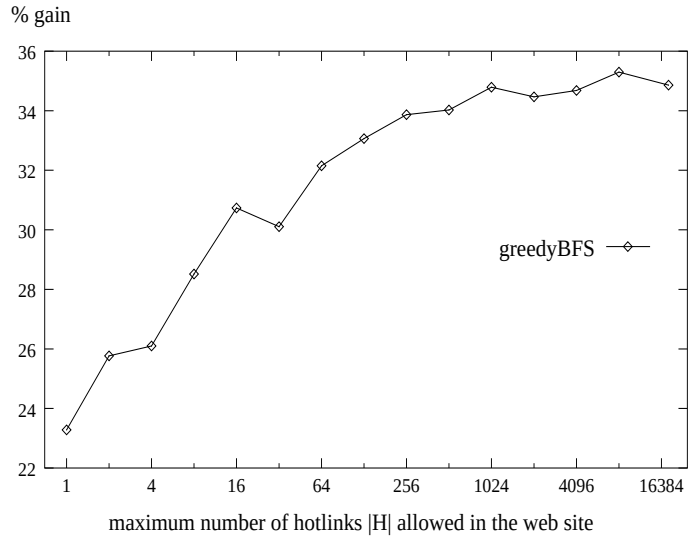

Fig. 6. Average gain obtained by applying *greedyBFS* to actual Web sites. The x axis are plotted in logarithmic scale base 2. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 5.53. The proportion of gain decreases at some points due to the different assignments of Zipf's distribution in each sample.

Table 3. Average proportion of gain over the access cost of eleven real Web sites when the total number of hotlinks is limited. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 5.53. The proportion of gain decreases at some points due to the different assignments of Zipf's distribution in each sample.

max total number of hotlinks, $\|H\|$	% of gain	standard deviation
1	23.28	4.95
2	25.77	6.53
4	26.10	5.58
8	28.51	6.51
16	30.74	5.58
32	30.11	5.35
64	32.15	6.72
128	33.06	6.57
256	33.87	5.57
512	34.02	5.79
1,024	34.79	5.81
2,048	34.46	6.30
4,096	34.68	6.64
8,192	35.30	6.11
16,384	34.86	6.25

To test an algorithm, we convert the structure of the *scs.carleton.ca* domain into a directed graph $G = (V, E)$, where each vertex $v \in V$ represents a Web page, and every edge $(v, w) \in E$ represents a hyperlink going from v to w. We then construct a tree $T = (V, E')$ by performing breadth first search on G from the home page. We call this process *link structure* and it is described in Section 3.2.1.

We use the access log files of the domain to assign access probabilities to the pages. The log files used are over 57.2 MBytes in size and contain the access logs for 7 days, for a total of $602,879$ file requests. The process that assigns popularities to the Web pages is called *access probabilities* and its description can be found in Section 3.2.1.

Figure 7 illustrates the gain obtained in the experiments. The *scs.carleton.ca* domain contains 798 pages. Observe that the maximum gain of *greedyBFS* and *recursive* is achieved at $|H| = 53 \ll 798$, which represents less than 10% of the number of pages in the site.

2.4 Algorithm approximateHotlinkAssignment

Algorithm *approximateHotlinkAssignment* was designed by Kranakis et al [12]. They prove the following upper bound on the access cost of a Web site, which is good for Web site trees of small outdegree δ:

$$E[T] \leq \frac{\mathcal{H}(p)}{\log(\delta+1) - \left(\frac{\delta \log \delta}{\delta+1}\right)} + \frac{\delta+1}{\delta},$$

where $\mathcal{H}(p)$ denotes the entropy of the probability distribution on the leaves.

Algorithm *approximateHotlinkAssignment* is a recursive algorithm that works as follows. Let T_c denote a tree rooted at page c. In the first call to the algorithm, c is the home page. Recall that δ is the maximum outdegree of the tree and p_c is the access probability of page c. The algorithm partitions the original tree into subtrees and then proceeds recursively for each of these subtrees until it is not possible to add more hotlinks. The subtrees are:

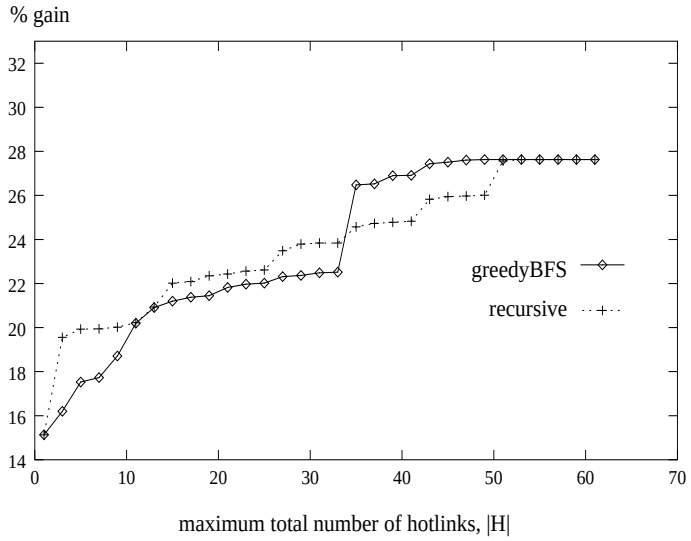

Fig. 7. Gain attained by *greedyBFS*, and *recursive* in the *scs.carleton.ca* domain. For $|H| < 53$ *greedyBFS* and *recursive* differ in performance. Observe that this behaviour is due to the different order on which each algorithm assigns the hotlinks; however, in the end they will always converge, as the latter is the recursive version of the former. Note that again the maximum gain is achieved with few hotlinks, as $53 \ll |V| = 798$.

1. The trees rooted at the children of c minus:

2. The tree rooted at a node whose weight is between $\frac{p_c}{\delta+1}$ and $\frac{\delta \cdot p_c}{\delta+1}$.

Algorithm *approximateHotlinkAssignment* is described next. The set of hotlinks, H, is initially empty.

2.4.1 Performance of Algorithm approximateHotlinkAssignment

We present the results of comparing the performance of algorithms *greedyBFS,* described in Section 2.3, and *approximateHotlinkAssignment,* described in Section 2.4, with the theoretical lower bound on the access cost of a Web site, presented in [11]:

$$E[T^H] \geq \frac{1}{\log(\delta + k)} \cdot \mathcal{H}(p) = \frac{1}{\log(\delta + k)} \cdot \sum_{i=1}^{m} p_i \log(1/p_i), \tag{11}$$

where $\mathcal{H}(p)$ is the Entropy of the probability distribution p.

The performance is compared on randomly generated Web sites of size $10,000$ with maximum outdegree 3 to 19. Figure 8 and Table 4 illustrate the results of the simulations. The average proportion of gain for *greedyBFS, approximateHotlinkAssignment* and the theoretical lower bound on the access cost of a Web site, have 95% confidence intervals of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29, respectively.

Algorithm 5 approximateHotlinkAssignment(T_c)

1. *if c has grandchildren*
2. *find a page u, descendant of c such that* $\frac{p_c}{\delta+1} \leq p_u \leq \frac{\delta \cdot p_c}{\delta+1}$
   (a) *if no such descendant exists let u be the descendant leaf page of maximum p*
   (b) *if distance from c to u is* ≥ 2
      i. $H = H \cup (c, u)$
   (c) *else let u be the (any) grandchild of c of maximum p*
      i. $H = H \cup (c, u)$
   (d) *let v be the ancestor of u that is child of c*
   (e) approximateHotlinkAssignment($T_v - T_u$)
   (f) approximateHotlinkAssignment(T_u)
   (g) *for every child w of c, except v*
      i. approximateHotlinkAssignment(T_w)

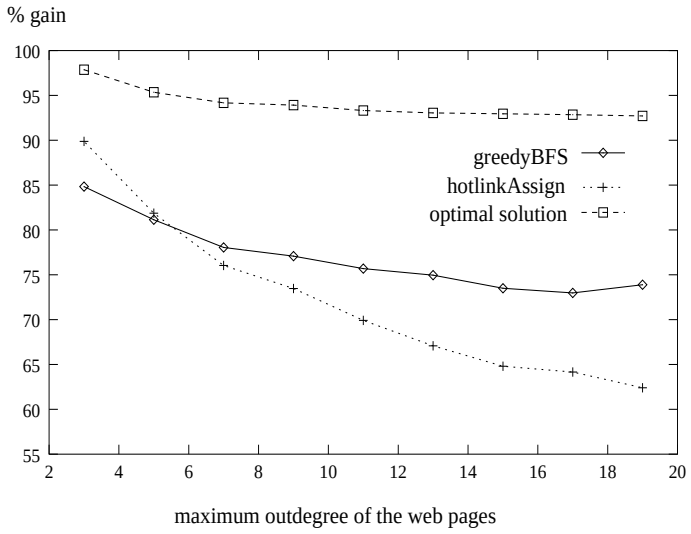

Fig. 8. Comparison of *greedyBFS* with *approximateHotlinkAssignment* and the theoretical lower bound on the access cost of a Web site (from Eq. 11) when applied to random Web sites of size 10,000 and maximum outdegree from 3 to 19. The average proportion of gain for *greedyBFS*, *approximateHotlinkAssignment* and the upper bound have a 95% confidence interval of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29 respectively.

Table 4. Comparison of the performance of *greedyBFS* with *approximateHotlinkAssignment* and the theoretical lower bound on the access cost of a Web site (from Eq. 11) when applied to random Web sites of size $10,000$ and maximum outdegree from 3 to 19. The average proportion of gain for *greedyBFS, approximateHotlinkAssignment* and the upper bound have a 95% confidence interval of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29 respectively.

maximum outdegree, δ	% of gain with *greedyBFS*	% of gain with *approximateHotlinkAssignment*	optimal % of gain
3	84.83	89.88	97.86
5	81.14	81.87	95.35
7	78.05	76.04	94.17
9	77.07	73.47	93.91
11	75.69	69.91	93.32
13	74.96	67.08	93.04
15	73.49	64.81	92.95
17	72.98	64.16	92.85
19	73.89	62.41	92.71

2.5 Algorithms k-greedyBFS *and* greedyBFS^k

This algorithm is a natural variation of *greedyBFS*. Again, the function *next_in_BFS_order* returns each of the nodes of the tree in breadth first search order, starting with the home page. Observe that the call *k-greedyBFS*($T, 1$) is equivalent to a call to *greedyBFS*(T). We also have a recursive version of *k-greedyBFS*, which we call *k-recursive*. This recursive algorithm will help us to validate the correctness of the simulations.

Algorithm 6 k-greedyBFS(T, k)

1. $H = \phi$
2. *while*((s = *next_in_BFS_order*) $\neq \phi$)
   (a) *for* $j = 1$ *to* k
      i. $t = v$: v *maximizes Eq. 3; and* v *is a descendant of* s*; and* v *does not have a hyperparent; and* v *is not descendant of a node* x*, which is at a higher level than* s *and already has a hyperparent*
      ii. *if* $t \neq \phi$ *then* $H = H \bigcup \{(s, t)\}$

We present another multiple hotlinks algorithm called *greedyBFS*^{k}, which assigns at most k hotlinks per page. This algorithm runs the algorithm *k-greedyBFS*($T, 1$) k times but creates a new breadth first search tree between each iteration,[f] treating the hotlinks as regular hyperlinks.

2.5.1 Performance of Algorithms k-greedyBFS *and* greedyBFS^k

Algorithms k-greedyBFS *and* greedyBFS^k *Evaluated on Random Web Sites*

Algorithms *k-greedyBFS* and *greedyBFS*^{k} are described in Sections 2.5 and 2.5. The results of the simulations are depicted in Figure 9 and Table 5. Observe that for values of $k > 5$,

[f] Observe that *k-greedyBFS*($T, 1$) is equivalent to *greedyBFS*(T), since $k = 1$.

Algorithm 7 greedyBFS^k(T, k)

1. *for* 1 *to* k
   (a) T^H = k-greedyBFS($T, 1$)
   (b) *delete the arcs of* T^H *that are not traversed in breadth first search starting from the home page. Call this tree* T.

greedyBFS^k outperforms *k-greedyBFS.*

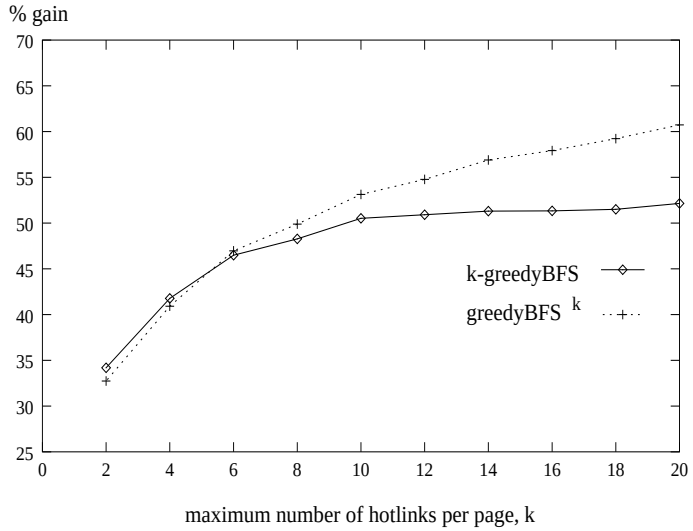

Fig. 9. Performance of *k-greedyBFS* and *greedyBFS*^{k}. Observe how *greedyBFS*^{k} outperforms *k-greedyBFS* when $k > 5$. The proportion of gains for *k-greedyBFS* and *greedyBFS*^{k} have a 95% confidence interval of at most ∓ 0.91 and ∓ 0.99 respectively.

2.6 Algorithm weighted-greedyBFS

One factor that slows down Web performance is the inevitable downloading of information that does not interest users. This problem happens to anyone who surfs the Web searching for information. For example, suppose we are on page v and want to get the information located in page w. Among the hyperlinks of v, however, there is no hyperlink (v, w) that takes us directly to w. Inevitably, we have to "download" others pages until we find a page x with the desired hyperlink (x, w). Given a Web site, we want to find an assignment of hotlinks that minimizes the expected data transfer, i.e., the necessary amount of bytes to be downloaded to reach one of its pages.

The previous sections dealt with the problem of improving Web performance by optimizing the average number of steps required to reach the pages of a Web site. As a consequence of minimizing the number of steps, we certainly reduce the amount of irrelevant information that needs to be transferred, yet, that was not the aim. In this section we study the assignment

Table 5. Performance of *k-greedyBFS* and $greedyBFS^k$. Observe how $greedyBFS^k$ outperforms *k-greedyBFS* when $k > 5$. The average proportion of gain for *k-greedyBFS* and $greedyBFS^k$ have a 95% confidence interval of at most ∓ 0.91 and ∓ 0.99 respectively.

maximum hotlinks per page, k	% of gain with *k-greedyBFS*	95% confidence interval	% of gain with $greedyBFS^k$	95% confidence interval
2	34.18	0.91	32.75	0.79
4	41.77	0.84	40.92	0.99
6	46.48	0.80	46.97	0.87
8	48.27	0.62	49.89	0.67
10	50.52	0.74	53.13	0.80
12	50.92	0.75	54.78	0.68
14	51.31	0.69	56.90	0.74
16	51.34	0.41	57.92	0.64
18	51.50	0.64	59.23	0.72
20	52.15	0.80	60.72	0.69

of hotlinks for optimizing the average data transfer. We restrict the problem to assigning at most one hotlink per node.

One is tempted to pursue a similar idea to that of [12] in order to find a balanced partitioning of the tree. Inevitably this will lead to a recursive algorithm like in [12]. However there are some difficulties to this approach. In this section we present a heuristic algorithm called *weighted-greedyBFS.*

Algorithm *weighted-greedyBFS,* described in Algorithm 8, operates in the same way as algorithm *greedyBFS,* presented in Section 2.3. The difference between these two algorithms is that the former attempts to maximize Eq. 4, whereas the latter attempts to maximize Eq. 3. Algorithm *weighted-greedyBFS* assigns hotlinks iteratively in breadth first search order beginning with the home page. Consider a hotlink (s, t). In each iteration of the algorithm, s corresponds to the next node in breadth first search order, and t corresponds to the descendant of s that maximizes the gain, but that is not a descendant of a node x, which is at a higher level than s and already has an incoming hotlink. See Figure 2. Recall that function *next_in_BFS_order* returns each node of the tree in breadth first search order, starting from the home page. The algorithm stops when no more hotlinks can be assigned. We will use a recursive version of *weighted-greedyBFS* called *weighted-recursive* for validating the correctness of the simulations.

Algorithm 8 weighted-greedyBFS(T)

1. $H = \phi$
2. *while(* $(s = next\_in\_BFS\_order) \neq \phi$ *)*
   (a) $t = v$: *v maximizes Eq. 4; and v is a descendant of s; and v does not have a hyperparent; and v is not a descendant of a node x, which is at a higher level than s and already has a hyperparent.*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s, t)\}$

2.6.1 Performance of Algorithm weighted-greedyBFS

Algorithm weighted-greedyBFS *Evaluated on Random Web Site Trees*

The results of the simulations are plotted in Figure 10 and displayed in Table 6. We plot the average proportion of gain that can be attained by the algorithm. The average proportion of gain has a 95% confidence interval of at most ∓ 2.27. Observe that the gain of the algorithm oscillates between narrow intervals, indicating that the size of the tree does not greatly affect the performance.

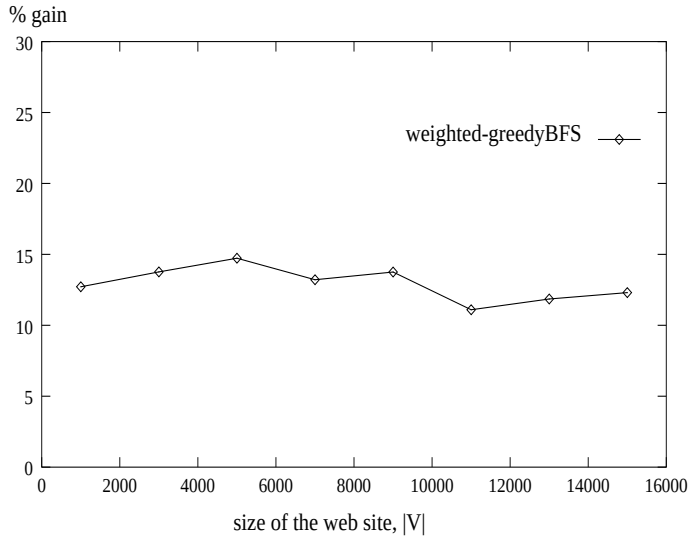

Fig. 10. Gain obtained by applying *weighted-greedyBFS* to randomly generated Web sites of size $1,000$ to $15,000$. The average proportion of gain has a 95% confidence interval of at most ∓ 2.27.

Table 6. Proportion of gain on the access costs of randomly generated Web sites of $1,000$ to $15,000$ pages. The proportion of gain has a 95% confidence interval of at most ∓ 2.27.

number of pages in Web site, $\|V\|$	% of gain	standard deviation	95% confidence interval
1,000	12.71	6.54	2.27
3,000	13.76	5.12	1.78
5,000	14.73	6.10	2.11
7,000	13.21	4.03	1.40
9,000	13.75	5.37	1.86
11,000	11.10	3.51	1.22
13,000	11.86	4.06	1.41
15,000	12.30	3.81	1.32

Algorithm weighted-greedyBFS *Evaluated on Real Web Sites*

We show the results of the simulations in Figure 11 and Table 7. The average number of nodes in the sample of Web sites is $9,669.2$. Observe that the proportion of gain stabilizes after assigning less than 500 hotlinks, which is roughly 10% of the average number of Web

pages.

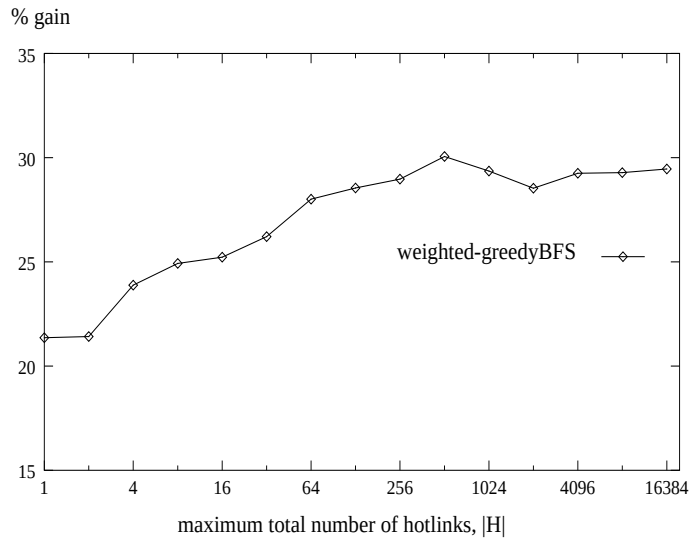

Fig. 11. Average gain obtained by applying *weighted-greedyBFS* to actual Web sites. The x axis are plotted in logarithmic scale base 2. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 7.40. The proportion of gain decreases at some points due to the different assignments of probabilities and file sizes in each sample.

Case Study

We are able to test our algorithms on the *scs.carleton.ca* domain because we have all the necessary information, i.e., the hyperlink structure, access logs, and file sizes.

The case study is conducted in the same way as described in Section 2.3.2. In this case, we need to associate real weights to the Web pages. The weight of a Web page is its own size plus the size of its embedded files (in bytes).

Figure 12 illustrates the proportion of gain obtained by the algorithm. The *scs.carleton.ca* domain contains 798 pages. Observe that the maximum gain of *greedyBFS* and *recursive* is achieved at $|H| = 53 \ll 798$, which represents less than 10% of the number of pages in the site.

2.7 Validation of the Simulations

To validate the correctness of the simulations, we use the model validation techniques presented by Jain [40]. We evaluate both the correctness of the implementation of the algorithms and the generation of realistic Web sites.

Correctness of the Algorithms

In order to verify the correctness of the implementation of the algorithms, we check that the outcomes of the algorithm and their recursive implementations are exactly the same. With this test, we can be confident that the implementation of the algorithms is correct since it is unlikely that different (logical or syntactical) errors in the implementation of the

Table 7. Average proportion of gain over the access cost of eleven real Web sites when the total number of hotlinks is limited. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 7.40. The proportion of gain decreases at some points due to the different assignments of probabilities and file sizes in each sample.

max total number of hotlinks, $\|H\|$	% of gain	standard deviation
1	21.36	8.40
2	21.42	9.33
4	23.88	11.26
8	24.92	11.04
16	25.23	9.85
32	26.21	11.14
64	28.01	12.17
128	28.55	12.30
256	28.97	11.24
512	30.10	13.54
1,024	29.35	12.75
2,048	28.53	11.93
4,096	29.25	11.88
8,192	29.28	13.57
16,384	29.45	13.40

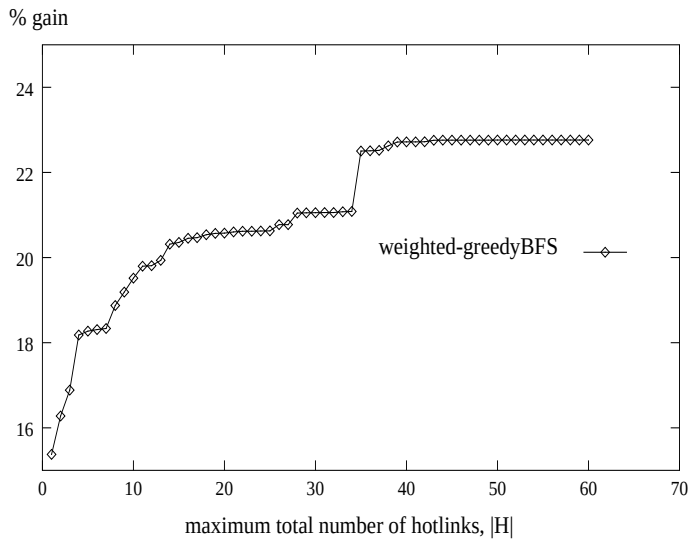

Fig. 12. Gain attained by algorithm *weighted-greedyBFS* in the *scs.carleton.ca* domain. The domain contains 798 pages and the maximum gain is attained at $|H| = 52$ hotlinks, which indicates that we can get good gain with just a "few" hotlinks.

algorithms produce the same results. Since algorithms *k-greedyBFS* and *greedyBFS*^{k} are variations of algorithm *greedyBFS,* we can also be confident that these algorithms are correctly implemented.

Correctness of the Generation of Random Web Sites

According to our criteria, a random Web site has a realistic structure if the outdegree follows Eq. 8. Figure 13 plots the outdegree frequencies that characterize our random Web sites. To validate the correct assignment of access probabilities to the intermediate, i.e., non-leaf, pages, we verify that the access probability of the home page is 1, since each node's access probability is the sum of the probabilities of the leaves descendant to it.

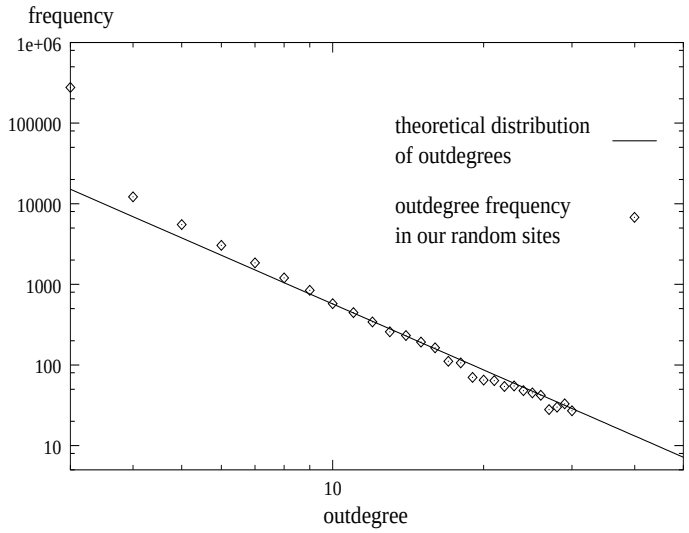

Fig. 13. The correctness of the structure of the random Web sites is proven by comparing their outdegree frequencies with a power-law distribution (given by Eq. 8) in a loglog scale.

Correctness of the Generation of File Sizes

To validate the correctness of the generation of file sizes, we plot the distribution of file sizes of the Web sites used in our simulations and verify that they follow the hybrid distribution discussed in Section 2.1.5. The distribution of file sizes used in the simulations is plotted in Figure 14.

3 The Hotlink Optimizer

In Section 2 we looked at the efficiency of the hotlink assignment algorithms and found that we can reduce the access cost of a Web site by as much as 35%. This result suggests that the development of a hotlink assignment tool would be useful. In this section we present the architecture and user interface for the Hotlink Optimizer (HotOpt), a powerful software tool that assists Web administrators and designers in restructuring their Web sites according to the needs of users. HotOpt is presented in [41] as well.

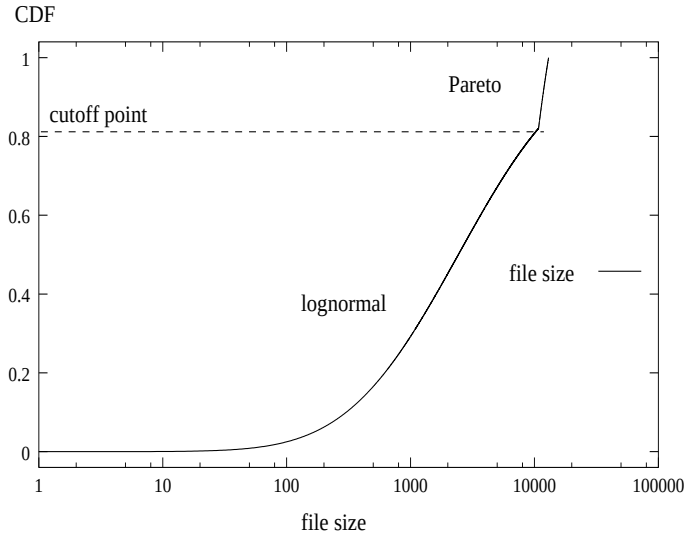

Fig. 14. The correctness of the file sizes used in our simulations is proven by plotting the cumulative distribution function of the hybrid (lognormal-Pareto) distribution, given in Formula 10. The file sizes are plotted in log scale. The cutoff point is such that approximately 83% of the file sizes fall in the lognormal distribution.

By analyzing the hyperlink structure of a Web site, and looking at the user's patterns, HotOpt is able to suggest a set of hotlinks to be added to the Web site. This is a semi-automatic process in the sense that the hotlinks are found automatically, but they are not automatically added to the Web site. For the moment, we want to assist Web designers, not to replace them.

HotOpt can find a set of hotlinks, H, that reduces the access cost of the Web site. The inputs are the *home page file*[9]and the *access log files*, and the output is the set of hotlinks H along with x, the proportion of gain offered by H. Figure 15 depicts a macroscopic view of HotOpt.

3.1 The User Interface

HotOpt has a user interface that displays the tree-shaped hyperlink structure of the Web site (cf. Figure 16). The user has to provide the location of the home page and the access log files.

3.1.1 Initial Set Up

In order to run HotOpt, the Web administrator needs to have both the html source files and the log files in an explicit path on his or her system. It is important to remark that HotOpt will not delete or modify any existing file of the Web site nor delete any hyperlinks.

[9]Observe that from the home page we can perform breadth first search on the Web site.

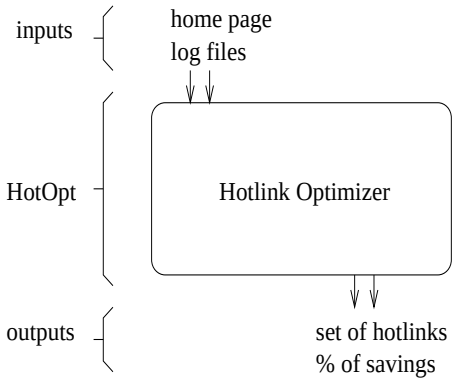

Fig. 15. Inputs and outputs of HotOpt.

3.1.2 Input

The user needs to provide the home page and the log files (top left corner of Figure 16). If no log files are available, HotOpt will use an arbitrary Zipf's popularity distribution. Refer to Section 2.1.4 to see how Zipf's distribution is used. The user may (optionally) specify the output file where the set of hotlinks is to be stored. The user can also limit the overall budget of hotlinks to be assigned, as well as the maximum number of hotlinks per page (top center part of Figure 16). The optimization process starts by pressing the "HotOpt" button located in the top right corner of the interface. See Figure 16.

3.1.3 Output

HotOpt takes the hyperlink structure of the Web site and transforms it into a tree. This tree is displayed in the main window of the user interface along with the hotlinks, as depicted in Figure 16. The hotlinks are easily identified by assigning a unique number to each of them. For example, in Figure 16, we can see that there is a hotlink going from *~kranakis\index.htm* to *~kranakis\513.html,* a second one going from *~kranakis\teach.html* to *~kranakis\523-refs.html,* and a third one from *~kranakis\513.html* to *~kranakis\513-projects00.html.* HotOpt displays the original access cost and the new cost, along with the proportion of gain (see the top right corner of Figure 16). The set of hotlinks are saved in the file specified by the user (cf. Figure 17).

3.2 Architecture

The tasks performed by HotOpt can be summarized as follows (cf. Figure 18). First, the Web site is transformed into a graph and then into a tree. Secondly, a probability distribution is assigned to the leaves of the tree. Finally, the optimization algorithm is applied.

3.2.1 Processes of HotOpt

HotOpt performs four basic processes.

1. *Link structure (graph).* Beginning at the home page file, build a directed graph $G = (V, E)$, where V is the set of Web pages and E is the set of hyperlinks between the Web

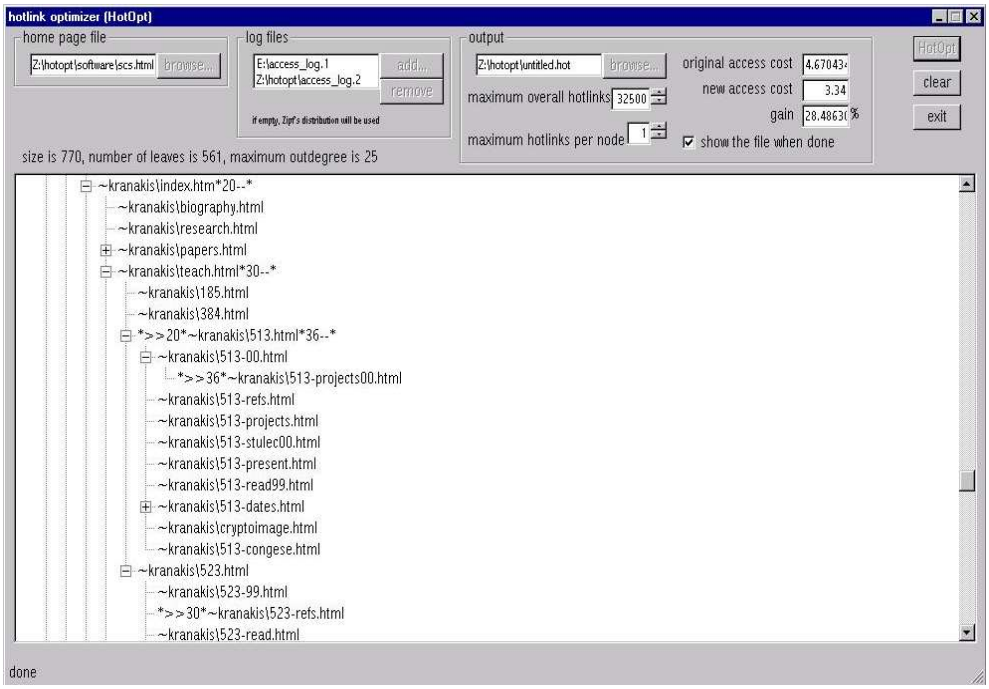

Fig. 16. User interface of HotOpt.

Fig. 17. Output of HotOpt.

pages.

2. *Link structure.* Construct a tree $T = (V, E')$ performing breadth first search on the graph G. The root of the tree is the node associated to the home page.

3. *Access probabilities.* Assign probabilities to the leaves of the tree by counting the number of times, according to the log files, that each leaf was requested.

4. *Optimization algorithm.* Find an assignment of hotlinks to T. Based on the simulations of Section 2, we conclude that the best hotlink assignment algorithm known so far is algorithm *greedyBFS*. This algorithm is implemented in HotOpt. Finally, output the set of hotlinks along with the proportion of gain that those hotlinks can achieve.

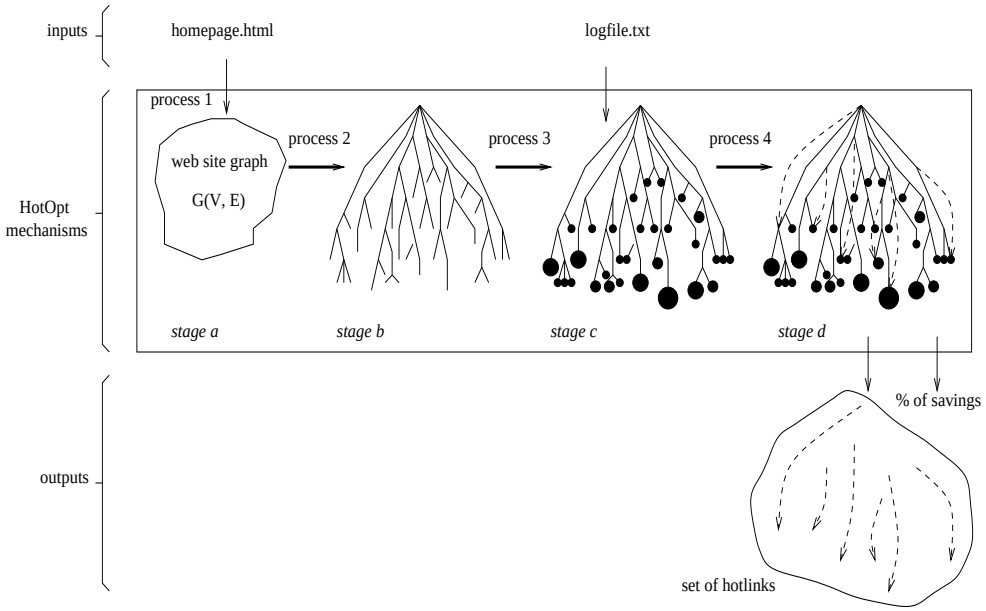

Fig. 18. HotOpt performs four basic processes. Process *link structure (graph)* constructs a directed graph, where the nodes are Web pages and the edges are the hyperlinks that connect the pages. Process *link structure* builds a Web tree in breadth first search order with the home page as the root. Process *access probabilities* assigns probabilities to the leaves of the tree based on the access log files. Process *optimization algorithm* is crucial since it is responsible for applying the optimization algorithm.

We describe these four processes in detail.

Link Structure

In this section we explain the *link structure* process, which transforms a Web site into a tree.

Consider a Web site as a directed graph, where Web pages are nodes connected by hyperlinks. The transformation of a Web site into a graph is performed by the process called *link structure (graph)*. It has been proven that some instances of the hotlink assignment problem are NP-hard (see [11]). Therefore, we approach the problem for trees. However, the tree

must be constructed without braking the semantic structure of the Web site, and keeping the shortest path from the home page to all the other pages of the site.

We construct a tree in breadth first search order so that the root is the node associated to the home page. The process is formally described in Algorithm 9. The algorithm receives as parameters a graph $G = (V, E)$ and a distinct node r associated to the home page of the Web site. The output will be a breadth first search tree $T = (V, E')$ rooted at node r.

It is convenient to identify each node by the path and file name of the source file of the Web page, so that the hotlinks can be easily identified.

Algorithm 9 websitetoBFStree($G = (V, E), r$)

---

1. *let q be an empty queue*
2. $E' = \phi$
3. $\forall v \in V$, *mark v as not visited*
4. *mark r as visited*
5. *enqueue r in q*
6. *while q is not empty*
   (a) $u =$*dequeue q*
   (b) *for each edge* $(u, v) \in E : v$
      i. *enqueue v in q*
      ii. *mark v as visited*
      iii. $E' = E' \cup (u, v)$
7. *return* $T = (V, E'), r$

Access Probabilities

In this section we describe the process of assigning a probability distribution to the leaf pages of the tree. The access probability of a Web page is determined by its popularity. We compute the popularity of a Web page by counting the number of times, according to the access logs of the Web site, that this page was visited by the users.

Consider the access log of a Web site (cf. Figure 19). Each entry in the access log contains information about one request: requester's *IP address*, *time* of request, *file* requested, *protocol* used, *http code*, and *size* of the file in bytes. Not all of the entries represent an actual hit to a page, since some represent the request for a file embedded in a page and others indicate an unsuccessful attempt to download a page, i.e., due to client or server errors. Therefore we need to filter the entries and extract only the actual hits to Web pages. We are interested on the entries with *http code* 200 or 304, i.e., "ok", and "not modified" respectively since they are the only ones that indicate an actual access to the page. Refer to [42] for a complete description of the http 1.1 protocol and [43] for a complete description of log files. Algorithm *extractHits,* given in Algorithm 10, receives as inputs a tree T and a log file. After the execution of the

algorithm, each leaf of the tree will have a popularity associated to it, depending on the number of times the leaf was visited.

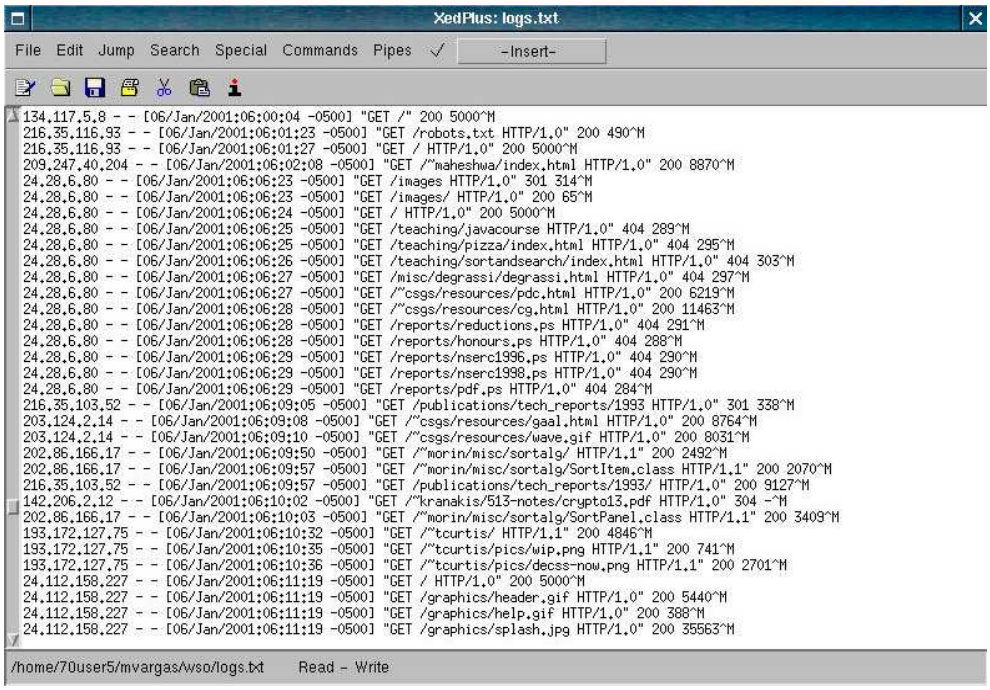

Fig. 19. Example of an access log file of a Web site. The fields are as follows: requester's *IP address*, *time* of request, *file* requested, *protocol* used, *http code*, and *size* of the file in bytes.

Algorithm 10 extractHits(T, *logFile*)

1. *for each entry of* logFile
2. *if* code= 200 *or* code= 304 *and* file *requested is a leaf* v *of* T
   (a) *increment by* 1 *the number of hits to leaf* v

Once algorithm *extractHits,* given in Algorithm 10, has been executed, we compute the access probability distribution of the leaf pages based on their popularity. Let a_l be the number of times that leaf page l was requested, and let $A = \sum_l a_l$, be the total number of times all the leaves were requested. The access probability of leaf l is defined by $p_l = \frac{a_l}{A}$.

After the access probability distribution has been computed, we need to assign weights to the internal nodes of the tree in a bottom-up fashion, such that the weight of page v, denoted as p_v, is the sum of the probabilities of the leaves that are descendants of v. Observe that for the home page r, $p_r = 1$.

Optimization Algorithm

This process runs the hotlink assignment algorithm. Among the algorithms tested, algorithm *greedyBFS* performed best. This algorithm is described in Section 2.3. Algorithm *greedyBFS* is the optimization algorithm implemented in HotOpt.

3.2.2 Modular Structure

Figure 20 illustrates the main structure of HotOpt. The *tree constructor* module takes the home page and the access log files as input. It is responsible for running *link structure (graph), link structure,* and *access probabilities* processes. These processes are depicted in Figure 18. The *tree constructor* module outputs a Web site tree, $T = (V, E)$ with a probability distribution p over its leaves.

The *optimization algorithm* sub-module applies the best hotlink assignment algorithm depending on the characteristics of the tree T. Such characteristics are extracted by the sub-module *tree characterizer*. Currently, HotOpt applies the algorithm *greedyBFS* to any Web site, regardless of its characteristics, as we have not finished determining which characteristics are significant. Moreover, algorithm *greedyBFS* has proven to be the best in all situations we have examined so far. See Section 2.3.1 to see the performance of algorithm *greedyBFS*. The final outputs are the set of hotlinks H and the proportion of gain, $x\%$, offered by H.

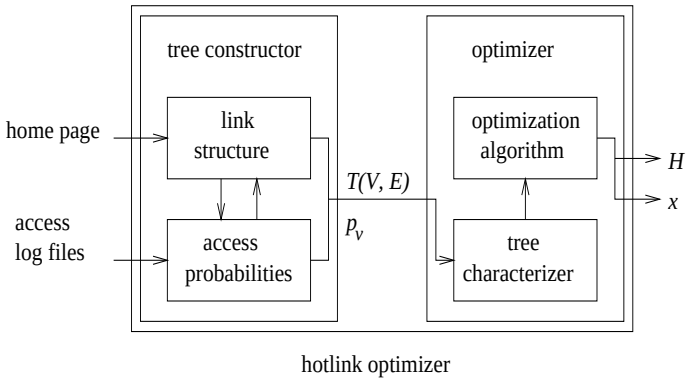

Fig. 20. Main structure of HotOpt.

4 Conclusions and Extensions

In this paper we studied the assignment of hotlinks for improving the access cost of Web sites. Two measures were used to determine the access cost of a Web site, namely, the expected number of steps and the expected data transfer. We presented hotlink assignment algorithms for arbitrary Web sites. These algorithms were tested using simulations on random and real Web sites. We found that the expected number of steps in a Web site can be reduced by at least 24% and as much as 35%, whereas the data transfer can be reduced by at least 11% and as much as 30%. These results, however, are theoretical because it is assumed that users would use the hotlinks to navigate on the Web site. It would be interesting to see to what extent users actually use the hotlinks to determine what is the actual gain of the algorithms.

Extensions to this work include the design of new hotlink assignment algorithms, especially for the case of multiple hotlinks per page, where many possible heuristics arise. We developed the Hotlink Optimizer (HotOpt), a software tool that assists Web designers to assign efficient hotlinks to a Web site. HotOpt has a friendly user interface and is able to display a tree visualization of the Web site including the suggested hotlinks. HotOpt implements our best hotlink assignment algorithm, but HotOpt can be upgraded if a better algorithm is found.

Acknowledgements

J. Czyzowicz, E. Kranakis and A. Pelc are supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant; E. Kranakis, D. Krizanc, and M. Vargas Martin are supported in part by MITACS (Mathematics of Information Technology and Complex Systems) NCE (Networks of Centres of Excellence) grant; M. Vargas Martin is supported in part by CONACYT (Science and Technology Council of Mexico).

References

1. L.L. Peterson and B.S. Davie (1996), *Computer networks: A systems approach*, Morgan Kaufmann Publishers (San Francisco).
2. L.D. Catledge and J.E. Pitkow (1995), *Characterizing browsing strategies in the World-Wide Web*, Computer Networks and ISDN Systems, vol. 27, no. 6, pp. 1065-1073.
3. M.C. Drott (1998), *Using Web server logs to improve site design*, in Proceedings of the ACM Sixteenth Annual International Conference of Computer Documentation (SIGDOC-98), New York, USA, 23-26 September 1998, pp. 43-50.
4. P. Pirolli, J. Pitkow, and R. Rao (1996), *Silk from a sow's ear: Extracting usable structure from the Web*, in Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems, Vancouver, Canada, 13-18 April 1996, vol. 1, pp. 118-125.
5. M. Perkowitz and O. Etzioni (1997), *Adaptive sites: An AI challenge*, in Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97), Nagoya, Japan, 23-29 August 1997, vol. 1, pp. 16-23.
6. M. Perkowitz and O. Etzioni (1999), *Towards adaptive Web sites: Conceptual framework and case study*, Artificial Intelligence, vol. 118, no. 1-2, pp. 245-275.
7. M. Spiliopoulou, L.C. Faulstich, and K. Winkler (1999), *A data miner analyzing the navigational behaviour of Web users*, in Proceedings of Workshop on Machine Learning in User Modeling of the ACAI'99, Creta, Greece, July 1999, pp. 588-589.
8. T. Nakayama, H. Kato, and Y. Yamane (2000), *Discovering the gap between Web site designers' expectations and users' behavior*, Computer Networks, vol. 33, no. 1-6, pp. 811-822.
9. Y. Fu, M. Creado, and M. Shih (2001), *Adaptive Web sites by Web usage mining*, in Proceedings of the International Conference on Internet Computing (IC 2001), Las Vegas, USA, 25-28 June 2001, pp. 28-34.
10. R. Srikant and Y. Yang (2001), *Mining Web logs to improve Website organization*, in Proceedings of the Tenth International World Wide Web Conference, Hong Kong, 1-5 May 2001, pp. 430-437.
11. P. Bose, J. Czyzowicz, L. Gąsieniec, E. Kranakis, D. Krizanc, A. Pelc, and M. Vargas Martin (2000), *Strategies for hotlink assignments*, in Proceedings of the Eleventh Annual International Symposium on Algorithms and Computation (ISAAC 2000), Taipei, Taiwan, December 2000, pp.23-34.
12. E. Kranakis, D. Krizanc, and S. Shende (2001), *Approximate hotlink assignment*, in Proceedings of the Twelfth Annual International Symposium on Algorithms and Computation (ISAAC 2001), Christchurch, New Zealand, 19-21 December 2001, vol. 2223, pp. 756-767.

13. J. Czyzowicz, E. Kranakis, D. Krizanc, A. Pelc, and M. Vargas Martin (2003), *Optimal assignment of bookmarks to Web pages*. In progress.
14. S. Fuhrmann, S.O. Krumke, and H.C. Wirth (2001), *Multiple hotlink assignment*, in Proceedings of the Twenty-Seventh International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001), Boltenhagen, Germany, 14-16 June 2001, pp. 189-200.
15. M. Faloutsos, P. Faloutsos, and C. Faloutsos (1999), *On power-law relationships of the Internet topology*, Computer Communication Review, vol. 29, no. 4, pp. 251-262.
16. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener (2000), *Graph structure in the Web*, Computer Networks, vol. 33, no. 1-6, June 2000, pp. 309-320.
17. B.M. Waxman (1988), *Routing of multipoint connections*, IEEE Journal on Selected Areas in Communications, vol. 6, no. 9, pp. 1617-1622.
18. K.L. Calvert, M.B. Doar, and E.W. Zegura (1997), *Modeling Internet topology*, IEEE Communications Magazine, vol. 35, no. 6, pp. 160-163.
19. E.W. Zegura, K.L. Calvert, and S. Bhattacharjee (1996), *How to model an internetwork*, in Proceedings of the Conference on Computer Communications (INFOCOM'96), San Francisco, USA, 24-28 March 1996, vol. 2, pp. 594-602.
20. E.W. Zegura, K.L. Calvert, and M.J. Donahoo (1997), *A quantitative comparison of graph-based models for Internet topology*, IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 770-783.
21. W. Aiello, F. Chung, and L. Lu (2002), *A random graph model for massive graphs*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, USA, 21-23 May 2002, pp. 171-180.
22. D.J. Watts (1999), *Small worlds*, Princeton University Press (New Jersey).
23. L.A. Adamic (1999), *The small world Web*, in Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries, ECDL, vol. 1696, pp. 443-452.
24. B. Saulnier (1998), *Small world*, Cornell Magazine On Line, vol. 101, no. 1, Available: http://cornell-magazine.cornell.edu/Archive/JulyAugust98/JulyWorld.html [Accessed 1 April 2002].
25. B. Hayes (2000), *Graph theory in practice: Part I*, American Scientist, vol. 88, no. 1, pp. 9-13.
26. B. Hayes (2000), *Graph theory in practice: Part II*, American Scientist, vol. 88, no. 2, pp. 104-109.
27. M.R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork (2000), *On near-uniform URL sampling*, in Proceedings of the Nineth International World Wide Web Conference, Amsterdam, The Netherlands, 15-19 May 2000, pp. 295-308.
28. G.K. Zipf (1949), *Human behaviour and the principle of least effort*, Addison-Wesley (Cambridge).
29. S. Glassman (1994), *A caching relay for the World Wide Web*, Computer Networks and ISDN Systems, vol. 27, no. 2, pp. 165-173.
30. M. Levene, J. Borges, and G. Loizou (2001), *Zipf's law for Web surfers*, Knowledge and Information Systems, vol. 3, no. 1, pp. 120-129.
31. J.E. Pitkow (1998), *Summary of WWW characterizations*, Computer Networks and ISDN Systems, vol. 30, no. 1-7, pp. 551-558.
32. D.E. Knuth (1973), *The art of computer programming*, vol. 3: Sorting and searching, Addison-Wesley (Reading).
33. M.F. Arlitt and C.L. Williamson (1997), *Internet Web servers: Workload characterization and performance implications*, IEEE/ACM Transactions on Networking, vol. 5, no.5, pp. 631-645.
34. S.I. Resnick (1997), *Heavy tail modeling and teletraffic data*, Annals of Statistics, vol. 25, no. 5, pp. 1805-1869.
35. M.E. Crovella and A. Bestavros (1997), *Self-similarity in World Wide Web traffic. Evidence and possible causes*, IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 835-846.
36. P. Barford and M. Crovella (1998), *Generating representative Web workloads for network and server performance evaluation*, in Proceedings of Measurement and Modeling of Computer Systems, Madison, USA, July 1998, pp. 151-160.
37. A.B. Downey (2001), *The structural cause of file size distributions*, in Proceedings of the 2001

ACM SIGMETRICS Performance Evaluation Review, vol. 29, no. 1, pp. 328-329.
38. M. Mitzenmacher (2002), *Dynamic models for file sizes and double pareto distributions*, preprint.
39. P. Barford, A. Bestavros, A. Bradley and M. Crovella (1999), *Changes in Web client access patterns: Characteristics and caching implications*, World Wide Web, vol. 2, no. 1-2, pp. 15-28.
40. R. Jain (1991), *The art of computer systems performance analysis*, John Wiley and Sons (New York).
41. E. Kranakis, D. Krizanc, and M. Vargas Martin (2002), *The hotlink optimizer*, in Proceedings of the International Conference on Internet Computing (IC '02), Las Vegas, USA, 24-27 June 2002, vol. 2, pp. 87-94.
42. Network Working Group (1997), *Request for Comments: 2068*, Hypertext Transfer Protocol HTTP/1.1. Available: http://www.ietf.org/rfc/rfc2068.txt [Accessed 23 November 2000].
43. Apache Group (1999), *Apache HTTP Server Project*. Available: http://httpd.apache.org/docs/logs.html [Accessed 11 January 2001].