Enhancing Hyperlink Structure for Improving Web Performance

by

Miguel Vargas Martín

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

December 2002

The undersigned recommend to

The Faculty of Graduate Studies and Research

acceptance of the thesis,

Enhancing Hyperlink Structure for Improving Web Performance

submitted by

Miguel Vargas Martín

Dr. Frank Dehne

(Director, School of Computer Science)

Dr. Evangelos Kranakis

(Thesis Supervisor)

Dr. Andrzej Pelc

(Thesis Co-Supervisor)

Dr. Sunil Shende

(External Examiner)

Carleton University

December 2002

Abstract

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 $c(v) = p_v \cdot c(r, v)$, where $c(r, v)$ is 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 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 problem.

Consider a directed graph $G = (V, E)$, where V represents the pages of a Web site connected via hyperlinks E. We prove that the hotlink assignment problem is NP-hard, and give some theoretical results related to this 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 average number of steps in a Web site. HotOpt is empowered by one of the algorithms presented in this dissertation.

To my mother, **Rosario Martín Cuellar**

Acknowledgements

I wish to sincerely thank my thesis supervisors, Dr. Evangelos Kranakis and Dr. Andrzej Pelc, for being such excellent advisors and outstanding models of discipline, and for their talent and arduous work. I would also like to thank Dr. Jurek Czyzowicz, for his collaboration with our research. Special thanks to Dr. Danny Krizanc for proposing the idea of hotlink assignments, and contributing with valuable insight. I learned a great deal from all of them.

The Mexican Council of Science and Technology, CONACYT, supported my doctorate studies with Scholarship No. 116198. This research was also supported in part by Mathematics of Information Technology and Complex Systems, MITACS, under project CANCCOM and the Natural Sciences and Engineering Research Council, NSERC.

It was wonderful to be able to work in a nice friendly environment such as that of the School of Computer Science. My sincere thanks to the administrative staff, Sandy Dare, Linda Pfeiffer, Maureen Sherman and Marlene Wilson, and to Andrew Miles for his technical support.

It was a pleasure to work among my colleagues at Carleton University, Jemal Abawajy, Abdi Abdolreza, Veronique Audet, Steffen Christensen, Alina Dusa, Darrell Ferguson, Jen Hall, Kenji Imasaki, Zheyin Li, Jason Morrison, Deepak Saraswat, Cindy Sawchuk, Peter Taillon, Eric Torunski, Dimitrios Tsiounis, Andrew Vardy, Tao Wan, and Ji Zhang.

Thanks to my noble friends Gerardo Reynaga, Lisette López, Nicki Enouy, Beatriz and Willy Gaertner, Gratien Girod, Javier Govea, Luis Rueda, André Samson, Derek Smith and Juan P. Zamora, for standing by me when all I could see was the distant horizon.

I would like to thank my nieces, nephews and friends abroad, who kept in touch with me during the completion of this research. Special thanks go to Jaime Robles Vargas for keeping me company for almost an entire year, despite the cold weather.

Although my father is no longer among us, his words encouraged me to achieve a high level of education. With appreciation for your guidance, Salvador Vargas Hernández (1923 - 1993).

My brothers and sisters are twelve giant rocks that stood firm in turbulent waters. With their support I cross the oceans without getting wet. Thank you Leobardo, Irene, Leticia, Olivia, Enrique, Imelda, Esther, Constanza, Salvador, Godofredo, Mariano and Guadalupe.

None of this would have been possible without God's help.

I have no words to express my gratitude to my mother. Each page of this thesis is dedicated to her, Rosario Martín Cuellar.

Miguel Vargas Martín.

Contents

List of Tables

List of Figures

List of Algorithms

Part I

Preliminaries

Chapter 1

Introduction

1.1 Hotlinks

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. This research focuses on improving the design of Web sites by assigning hotlinks (shortcuts) to the collection of Web 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.

A *hotlink* is defined as a hyperlink that links a Web page to a descendant of that page. A special case of a hotlink is a *bookmark*, which may only link the home page to one other page of the collection. Thus, every bookmark is a hotlink but only certain hotlinks are bookmarks.

1.2 Why Hotlinks

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, which are discussed later.

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.

We have interesting problems concerning the assignment of hotlinks, which are theoretically challenging, and their solutions appear to be of significant utility.

1.3 Important Trends in Web Design and Improvement

Since its appearance in the second half of the 20^{th} Century, the Internet has been the subject of arduous study. Until the last two 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.

There are numerous studies on Web sites. In this section, we look at some important trends. First, we present a brief survey of Web characterizations. Secondly, we summarize the history of Web improvement with caching and adaptive Web sites.

1.3.1 Topological Characteristics

To improve the Web it is important to know what it looks like and how it is expanding. These are two of the most challenging and intriguing questions about the Web. Some researchers have found interesting topological characteristics of the Web collected within a certain period of time. The Web is constantly expanding, and therefore it is not enough to find its topological characteristics but it is important to find out how it is expanding. Many researchers use random Web sites to test the performance of their algorithms. The generation of random Web sites is not an easy task. In this section we survey some important literature on Web characterization and topology modeling.

Some authors extract important characteristics of the Web. Bray [10] provides approximate answers to interesting questions, for example, "how big is the Web?", "what is the average page like?", "how richly connected is the Web?", and "what does the Web look like?." Pitkow [46] extracts certain characteristics of the Web, such as, requested file popularity, site popularity, and reading time per page. As a consequence of fast expansion, their results are only an outdated snapshot of the Web and do not tell us much about how it looks now, or how it is expanding.

Due to this uncertainty, many researchers base their work on random graphs. This approach gives rise to the necessity of having accurate random graphs generators. Calvert et al. [12] discuss how graph-based models can be used to generate large graphs with specific parameters of locality and hierarchy. Zegura et al. [57] compare different random graphs generators and present new ones that guarantee certain char-

acteristics of the graphs that are generated. Aiello et al. [3] describe a random graph model for reproducing sparse gigantic graphs with particular degree sequences. The model proposed by Aiello et al. fails in reproducing a connected graph, though the graph generated contains a gigantic connected component. Hayes [27] explains the difficulties of modeling the Web accurately. Hayes [28] indicates that gigantic graphs, such as the Web, tend to have three particular characteristics, they are *sparse*, *clustered*, and of *small diameter* (*diameter* $\approx \log(n)$, where n is the number of nodes). Graphs exhibiting those characteristics are called *small world* graphs. See [53] for a full description of the small world phenomenon. Hayes [28] describes some classical methods for generating gigantic graphs, such as lattices andErdös-Rényi graphs, however, he points out that lattices do not have small diameter andErdös-Rényi graphs are not clustered.

Faloutsos et al. [21] find that the Web obeys power-law relationships. They observe that the probability that a page has i hyperlinks pointing to other pages is proportional to i^{-c}, where c is a positive constant. This important observation is a big step toward the answer of what the Web looks like. Another important result has been published by Reka et al. [48]. They observe that the diameter of the Web is 19, where diameter means the average distance between two random pages. Broder et al. [11] tightened the diameter given by Reka et al. to be 17 if the path is directed, and 6 if it is not. Broder et al. also find precise values for the constant c for the indegree, i.e., the number of pages that point to a page, and the outdegree, i.e., the number of pages pointed to by a page. In addition, they present a macroscopic picture of the Web.

The question of "how the Web is expanding?" remains as yet unanswered. Does the Web expand obeying certain patterns? Is it expanding randomly? These intriguing questions still constitute an obstacle in the efforts toward the improvement of the Web.

1.3.2 Web Caching

One of the proposed ways of Web improvement is caching. A complete description of caching can be found in [18]. Caching consists in storing the most requested Web pages in the proxy for future access. See [39] for an overview of proxies. One of the intrinsic problems with caching is deciding what to cache and for how long. Pitkow et al. [47] present an adaptive caching algorithm, based on the analysis of a psychological model of human memory, that responds to the hit rates and access patterns of users requesting documents. Glassman [24] describes the design and performance of a proxy based on a study on human behaviour. Lately, some commercial companies have explored the idea of Web mirroring, e.g., Akamai, Digital Island. Web mirroring consists of keeping copies (or mirrors) of a Web site (or part of it) in different strategic locations. Li et al. [37, 38] provide algorithms for optimally placing proxies in the Internet when the topology is a line [37] and a tree [38]. With the emergence of large multimedia traffic such as video and audio, different techniques have been deployed; for example, Wu et al. [55] propose a scheme where the time an object remains in the cache depends on a combination of the object's size and the last time it was fetched. Jung et al. [31] propose a pre-fetching approach that consists in taking advantage of the http protocol capability which allows to request partial documents when errors have occurred during the transmission, instead of requesting the whole document. Bouras et al. [9] propose to reduce data transfers over the Web by fragmenting pages, and whenever a user requests a page that is in the cache, only the outdated fragments (if any) of the page will be retrieved from the source server. Bauer et al. [7] analyse maximal forward paths[1] to retrieve pages to the user's cache before these pages are actually requested.

Even though memory is becoming inexpensive along with increased capacity, new

[1] A forward path is the path followed by a user in a single session until the user returns to a previously visited page.

applications require faster access to large amount of information. From this point of view, caching techniques must constantly adapt to the demands of new applications.

1.3.3 Access Patterns

Some researchers have focused their attention on the optimal design of Web sites based on user access patterns. Some of them suggest analysing user access patterns to help design better Web pages, sites, and browsers. Catledge et al. [13] and Drott et al. [20] analyse user access patterns to suggest improvements that help to design better Web pages. For example, 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. [45] propose to create aggregation of Web pages according to their importance or their content.

Perkowitz et al. [42] propose the design of adaptive Web sites by promoting and demoting pages, highlighting hyperlinks, adding hyperlinks and clustering related pages. Perkowitz et al. [43] present an algorithm that analyses user access logs in order to identify candidate hyperlink sets to be included in index pages. Their algorithm performs the following steps:

1. process the access log into visits;
2. find clusters of linked pages and create an adjacency matrix;
3. find maximal cliques;
4. rank the cliques found; and
5. for each cluster, create a Web page consisting of hyperlinks to the documents in the cluster.

The algorithm is tested only in a particular Web site. 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. [51] 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. [41] 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 analysing 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. [22] 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. [52] 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.

1.4 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."

1.4.1 The Problem

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 $c(v) = p_v \cdot c(r, v)$, where $c(r, v)$ is 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. A hotlink is an additional hyperlink that provides a shortcut between two pages. 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, we prove it is NP-hard. The problem and its NP-hardness are studied in Chapter 2 and can also be found in [8].

1.4.2 Principles of Hotlink Design

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 1.1 illustrates the idea. In Chapter 2 we formally define the access cost of a Web site.

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" ([44], 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

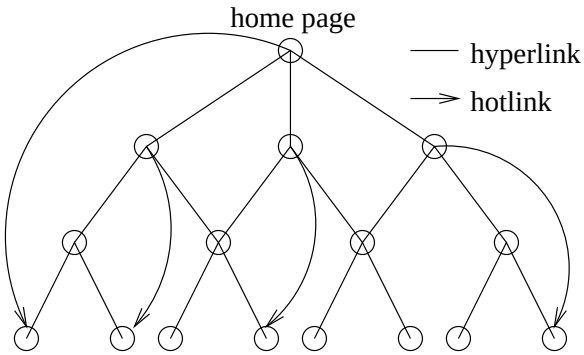

Figure 1.1: Example of an assignment of at most one hotlink per page. The figure shows a small Web site modeled with a directed graph (edges going downward).

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, as we endeavour to do in this dissertation, we may significally decrease latency. See the example illustrated in Figure 1.2.

One Hotlink per Page

The graph of a Web site contains a naturally embedded tree that can be constructed starting from the home page of the site. The home page has hyperlinks to internal pages that also have hyperlinks to other internal pages and so on. This perception of the Web site gives rise to a breadth first search (BFS) tree with the home page as the root. This breadth first search tree respects the most natural and basic struc-

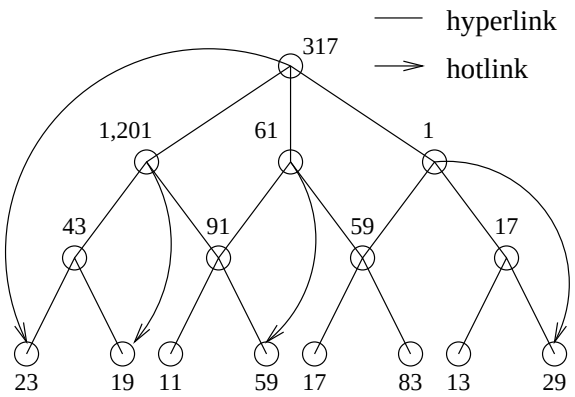

Figure 1.2: Example of how a hotlink assignment reduces the latency of a Web site by reducing the average data transfer. The sizes of the Web pages are in KBytes.

ture originally conceived by the Web site designer. In [16] we present our heuristic algorithms and study their performance.

Our simplest hotlink assignment algorithm is called *simpleBFS*. This algorithm assigns hotlinks iteratively in breadth first search order, starting from the home page. Let $A = \{v_1, v_2, ..., v_N\}$ be the set of nodes of the tree in increasing breadth first search order in such a way that v_1 is the home page. Now in the *i-th* iteration of the algorithm, we assign a hotlink (s, t), where $s = v_i$, and t is a descendant of s that minimizes the access cost. The algorithm stops when there are no more possible hotlinks to assign.

The next algorithm that we present is a slight modification of *simpleBFS*. Here, we also assign hotlinks iteratively in breadth first search order starting from the home page. The only difference is that in each iteration of the algorithm, we do not allow two hotlinks to cross each other, i.e., t is a descendant of s that minimizes the access cost, but is not a descendant of a node x, which is at a higher level than s and already has an incoming hotlink. See Figure 5.1. The algorithm stops when there are no more possible hotlinks to assign. We call this algorithm *greedyBFS* because certainly none of the hotlinks is wasted, as we will see in Chapter 5.

A third algorithm is a recursive version of *greedyBFS*. The basic idea is to assign a hotlink (s, t), where s is the home page and t is the node that offers the biggest savings on the access cost. The original tree is then split into subtrees consisting of the subtrees rooted at the children of s and the subtree rooted at t. This procedure proceeds recursively for each of these subtrees. See Figure 5.4. We simply call this algorithm *recursive*.

We evaluate the performance of the algorithms on random and real Web sites. The simulation reveals that we can save at least 24%, and as much as 35% of the original access cost of a Web site. These algorithms are studied in Chapters 5 and 8, and the test performed are described in Chapters 6 and 9.

Multiple Hotlinks per Page

A multiple hotlinks algorithm based on *greedyBFS* is called *k-greedyBFS,* and consists of assigning at most k hotlinks on each iteration of algorithm *greedyBFS*.

Another multiple hotlinks algorithm based on *greedyBFS* is called *greedyBFS*^{k}. This algorithm assigns at most k hotlinks per node. Algorithm *greedyBFS*^{k} runs k times the algorithm *greedyBFS* but after each of those k iterations creates a new breadth first search tree, and in the subsequent iterations treats the previous hotlinks as regular hyperlinks. Figure 5.5 illustrates the principle of the algorithm.

We compare the performance of the algorithms *k-greedyBFS* and *greedyBFS*^{k} on random Web sites. The simulation reveals that *greedyBFS*^{k} has a better performance for $k > 5$. These algorithms are studied in Chapter 5 and the experiments are described in Chapter 6.

The Hotlink Optimizer

In Chapter 7 we describe our *Hotlink Optimizer* (HotOpt). HotOpt is a powerful software tool that assists administrators and designers in structuring their Web sites

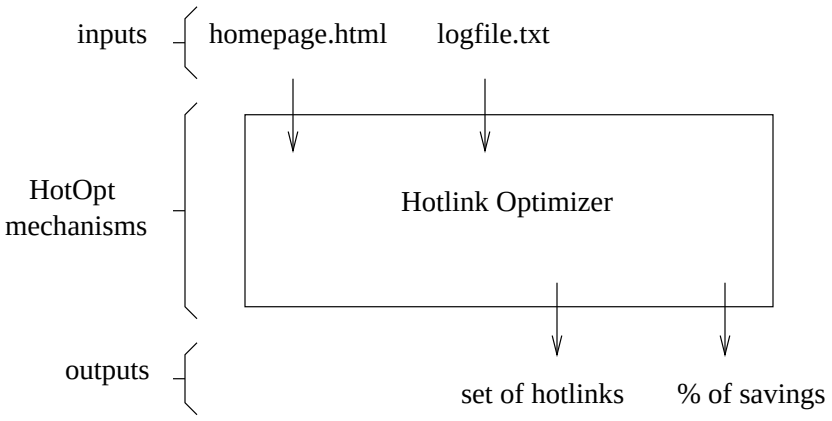

Figure 1.3: Inputs and outputs of HotOpt.

in such a way that users need fewer steps to reach the desired Web pages.

HotOpt is able to suggest a set of hotlinks to be added to the Web site by analysing its hyperlink structure, and taking into account the patterns of the users. This is a semi-automatic process in the sense that the hotlinks are found automatically, but are not automatically added to the Web site. For the moment, we want to assist Web designers, not to replace them.

The inputs of HotOpt are the *home page file*[2] and the *access log files*, and the output are the set of hotlinks H and the proportion of the access cost that can be reduced with this set of hotlinks, $x\%$. Figure 1.3 depicts a macroscopic view of HotOpt. HotOpt is also presented in [34].

Special Structure: Full Binary Trees

To improve our understanding of the problem we find it instructive to look at versions of the problem which are structurally simple but at the same time interesting enough to yield meaningful mathematical solutions. Following this approach, we looked me-

[2]Observe that from the home page we can obtain the structure of the Web site by performing breadth first search.

thodically at full binary trees.

Hotlink Assignments In Chapter 3 we introduce hotlink assignment algorithms for full binary trees with $m = 2^n$ leaves and present upper and lower bounds where the access probabilities of the trees are drawn from one of the following distributions:

- Uniform: $p_1 = p_2 = \cdots = p_m = 1/m$
- Geometric: $p_i = a^{-i}$, for $i = 1, ..., m-1$, and $p_m = 2 - (1 - a^{-m})/(1 - a^{-1})$, where $a > 1$
- Arbitrary: $p_1 + p_2 + \cdots + p_m = 1$
- Zipf: $p_i = \frac{1}{iH_m}$, where $H_m = \sum_{i=1}^{m} 1/i$ is the harmonic number

These results are also presented in [8].

Optimal Structure of Bookmarks An important technique for improving the response rate of large, distributed documents is to design the assignments of bookmarks in a careful and methodical manner. A bookmark is a special case of a hotlink. However, unlike general hotlinks, bookmarks can only connect the home page with any other page. This is the approach we follow in [17].

We pose the following *bookmark assignment problem* (k-BAP): Find an assignment of k bookmarks in a Web site that minimizes the expected number of steps to reach any page of the site from the home page.

1.5 General Notation and Terminology

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[3]. 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 weights[4] 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 page* v, denoted by $c(v)$ as:

$$c(v) = p_v \cdot c(r, v), \tag{1.1}$$

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 by the path, i.e., the number of bytes that need to be downloaded in order to traverse the path.

The *access cost of a Web site* T, is the expected number of steps (or the expected data transfer) to reach a leaf from the root, which is defined by:

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

[4] The terms *weight*, *probability,* and *popularity* will be used interchangeably throughout this dissertation, except in Part IV, where *weight* refers to the size (in bytes) of a Web page.

$$E[T] = \sum_{v \text{ is a leaf}} c(v)$$

1.6 Contributions of the Thesis

In this thesis we make several contributions. We start by looking at the mathematical structure of the problem and try to assess its intrinsic difficulty by looking at its NP-hardness and specific topologies. We consider some variants of the problem, namely, one hotlink per page and multiple hotlinks per page. We use simulations to test the performance of our algorithms. In particular, we use randomly generated Web sites, real Web sites and a case study. We develop the Hotlink Optimizer, a hotlink assignment software tool. In this section we summarize our main contributions.

1.6.1 Hotlink Assignments

First we study the mathematical structure of the problem for full binary trees and then we proceed to the general case.

Hotlinks in Specific Structures

We look at the assignment of at most one hotlink per node of full binary trees with well known probability distributions and prove that the hotlink assignment problem is NP-hard for directed acyclic graphs. Table 1.1 summarizes our results.

Bookmarks in Specific Structures

We prove a NP-hard result for directed acyclic graphs and present an optimal bookmark placement algorithm. More specifically, our results concerning the bookmark assignment problem are the following:

algorithm	distribution	lower bound	upper bound
bottomUpStrategy	geometric	$\Omega(1)$	$O(1)$
lengthTwoHotlinks	arbitrary	$\frac{H(p)}{\log 3}$	$\frac{3n+1}{4}$
lengthTwoHotlinks	uniform	$\frac{3n+1}{4}$	$\frac{3n+1}{4}$
sortedZipfHotlinks	Zipf (sorted)	$\frac{n}{2 \cdot \log 3}$	$\frac{n}{3} + O(\sqrt{n} \cdot \log n)$
lengthTwoHotlinks	Zipf	$\frac{n}{2 \cdot \log 3}$	$\frac{3n+1}{4}$

Table 1.1: Lower and upper bounds on the access cost of a full binary tree.

1. The bookmark assignment problem is NP-hard even for arbitrary directed acyclic graphs, even with uniform distribution of access probabilities.

2. We present a characterization of an optimal assignment of $k \leq \sqrt{N+1}$ bookmarks in a full binary tree of N nodes.

Hotlink Assignments to Web sites

Our algorithms for Web sites were tested with simulations on randomly generated Web sites, real sites, and a case study. Tables 1.2, 1.3 and 1.4 summarize the performance of our algorithms.

algorithm	maximum number of hotlinks per page	minimum % of reduction on the access cost	maximum % of reduction on the access cost
greedyBFS	1	24	27
k-greedyBFS	2	34	–
	20	–	52
greedyBFS^k	2	32	–
	20	–	60
weighted-greedyBFS	1	11	14

Table 1.2: Performance of our hotlink assignment algorithms tested on randomly generated Web sites. Algorithms *greedyBFS, k-greedyBFS,* and *greedyBFS*^{k} reduce the access cost in terms of the expected number of steps, while algorithm *weighted-greedyBFS* reduces the access cost in terms of the expected data transfer.

algorithm	maximum number of hotlinks per page	minimum % of reduction on the access cost	maximum % of reduction on the access cost
greedyBFS	1	34	35
weighted-greedyBFS	1	28	30

Table 1.3: Performance of our hotlink assignment algorithms tested on real Web sites. Algorithm *greedyBFS* reduces the access cost in terms of the expected number of steps, while algorithm *weighted-greedyBFS* reduces the access cost in terms of the expected data transfer.

algorithm	maximum number of hotlinks per page	% of reduction on the access cost
greedyBFS	1	27
weighted-greedyBFS	1	22

Table 1.4: Performance of our hotlink assignment algorithms tested in the *scs.carleton.ca* domain. Algorithm *greedyBFS* reduces the access cost in terms of the expected number of steps, while algorithm *weighted-greedyBFS* reduces the access cost in terms of the expected data transfer.

1.6.2 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. HotOpt is also presented in [34].

By analysing 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 7.2 shows the user interface of HotOpt.

1.7 Overview

This dissertation is organized in four parts. In Part I we provide the background for the study of hotlinks, explain our work intuitively, and define the problems and their difficulty. In Chapter 2 we formally define the problems, prove their NP-hardness and discuss our approach.

In Part II we present our work for full binary trees. Chapters 3 and 4 study the assignment of hotlinks and bookmarks.

In Part III we study the optimization of the expected number of steps to reach the pages of a Web site. In Chapter 5 we present a theoretical lower bound on the minimum number of steps required to reach a page of a Web site and present heuristic hotlink assignment algorithms. Chapter 5 is divided in two sections which study the assignment of at most one hotlink per page and the assignment of multiple hotlinks per page respectively. In Chapter 6 we evaluate the performance of the algorithms presented in Chapter 5. In Chapter 7 we introduce the Hotlink Optimizer, a software tool that assists Web administrators and designers by suggesting an assignment of

hotlinks.

Finally, in Part IV we study the optimization of the expected data transfer, i.e., the expected number of bytes that need to be transferred by the server when a user visits one of its pages. While in Chapter 8 we present a theoretical lower bound and a heuristic hotlink assignment algorithm, an evaluation of the algorithm is provided in Chapter 9.

Finally, in Chapter 10 we discuss the conclusions and some possible extensions to our work.

Chapter 2

What Makes Hotlink Assignments Difficult

2.1 Introduction

In this chapter we define the hotlink and bookmark assignment problem and prove that they belong to the set of NP-hard problems. Given the NP-hardness of the problems, we provide a framework for addressing them in this dissertation. This work is also presented in [8] and [17].

2.2 Optimal Hotlink Assignment Problem

We want to minimize the access cost of a Web site. The access cost of a Web site is equal to the average access costs of all its pages. The cost of a page is equal to the cost of the shortest path from the home page. 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 define the problems formally.

2.2.1 Optimizing the Expected Number of Steps

Assume there is a probability distribution over the leaves of a tree T. Let us say that a leaf i has probability p_i. We assign probabilities to the internal nodes in a bottom-up fashion, in such a way that the probability of a node is equal to the sum of the probabilities of the leaves that are its descendants. Consider the cost of page v,

$$c(v) = p_v \cdot c(r, v), \tag{2.1}$$

where $c(r, v)$ is the length of the (shortest) path between the home page, r, and page v. The length of the path from r to v, $c(r, v)$, is defined as the number of hyperlinks in the path.

The *expected number of steps* to reach a leaf from the root is defined by

$$E[T] = \sum_{v \text{ is a leaf}} c(v) \tag{2.2}$$

The *optimal k-hotlinks assignment problem* consists in minimizing Equation 2.2 by adding at most k hotlinks from each node of the tree. If $k = 1$ we call it *optimal hotlink assignment problem.* A *hotlink, h,* is an additional directed edge, (s, t), added to the original tree[1], such that t is a descendant of s. We say that s is the *hyperparent* of t, and t is the *hyperson* of s. Consider a set H of hotlinks assigned to the tree T; the resulting graph is denoted by T^H, and the *gain* of H is defined by

$$\mathcal{G}(H) = E[T] - E[T^H] \tag{2.3}$$

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

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

$$g(h) = p_t(d(t) - d(s) - 1) \tag{2.4}$$

A set of hotlinks H is optimal if $\mathcal{G}(H) \geq \mathcal{G}(H')$ for any hotlink set H'. The hotlink assignment problem is proven to be NP-hard in Section 2.3.

2.2.2 Optimizing the Expected Data Transfer

Another way of measuring the access cost of a Web site is by considering the average data transfer, i.e., the amount of bytes that need to be transferred in order to reach a Web page. All definitions in Section 2.2.1 remain the same, except the ones redefined here.

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

$$c(v) = w(v) \cdot p_v$$

The *expected data transfer* to reach a leaf from the root is defined by

$$E[T] = \sum_{v \text{ is a leaf}} c(v) \tag{2.5}$$

The *optimal hotlink assignment problem* consists in minimizing Equation 2.5 by adding hotlinks to the tree.

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

$$g(h) = p_t(w(t) - w(s) - w_t) \tag{2.6}$$

2.2.3 Optimal Bookmark Assignment Problem

Recall that a bookmark is a special case of a hotlink that connects the home page with any other page.

Recall that in the hotlink assignment problem, the probability distribution is on the leaves. In the bookmark assignment problem we assume that the probability distribution is over all the nodes V of T. Let us say that node i has probability p_i. Consider equation 2.1.

The *expected number of steps* to reach a node from the root is defined by

$$E[T] = \sum_{v \in V} c(v) \tag{2.7}$$

The *optimal bookmark assignment problem* (k-BAP) consists in minimizing Equation 2.7 by adding at most k bookmarks to the tree. A *bookmark* b, is an additional directed edge $(r, v), v \neq r$, added to the original tree. A bookmark set on T is a set $B \subset V$; thus we identify a bookmark (r, v) with the node v itself. Consider a set $B = \{b_1, \ldots, b_k\}$ of bookmarks assigned to the tree T; the resulting graph is denoted by T^B, and the *gain* of B is defined by

$$\mathcal{G}(B) = E[T] - E[T^B]$$

The *distance* from the root to a node v in T^B is denoted by $d_B(v)$. The *gain* of a single bookmark, $b \in B$, is denoted by

$$g_B(b) = d(b) - d_B(b)$$

The gain for a node v in T^B is denoted by

$$g_B(v) = d(v) - d_B(v)$$

A bookmark set B of size k is optimal if $\mathcal{G}(B) \geq \mathcal{G}(B')$ for any bookmark set B' of size k. We will prove that the optimal bookmark assignment problem is NP-hard.

2.3 NP-Hardness of the Hotlink Assignment Problem

In this section we prove that the *optimal hotlink assignment problem* is NP-hard even for arbitrary directed acyclic graphs with uniform distributions. Consider the following optimization problem.

HOTLINK ASSIGNMENT

Instance: Directed graph $G = (V, E)$, a node $s \in V$ from which every node of the graph can be reached, and positive integer g.

Question: Does there exist a hotlink assignment H for which the gain $\mathcal{G}(H)$ is at least g?

Theorem 1. *The problem HOTLINK ASSIGNMENT is NP-hard.*

Proof. The transformation is from the following NP-complete decision problem [23].

EXACT COVER BY 3-SETS (X3C)

Instance: Set S with $|S| = 3k$ and a collection C of 3-element subsets of S.

Question: Does C contain an exact cover for S, i.e., a subset $C' \subseteq C$ such that every element of S belongs to exactly one member of C'?

Given an instance (C, S) of X3C we construct an instance (G, g) of HOTLINK ASSIGNMENT as follows. Let the set $S = \{s_1, s_2, \ldots, s_m\}$ be of size $m = 3k$, and the set C of size t. The graph $G = (V, E)$ is defined as follows.

- The vertex set V consists of the vertices below:

  1. A_0 is the source node of the graph,

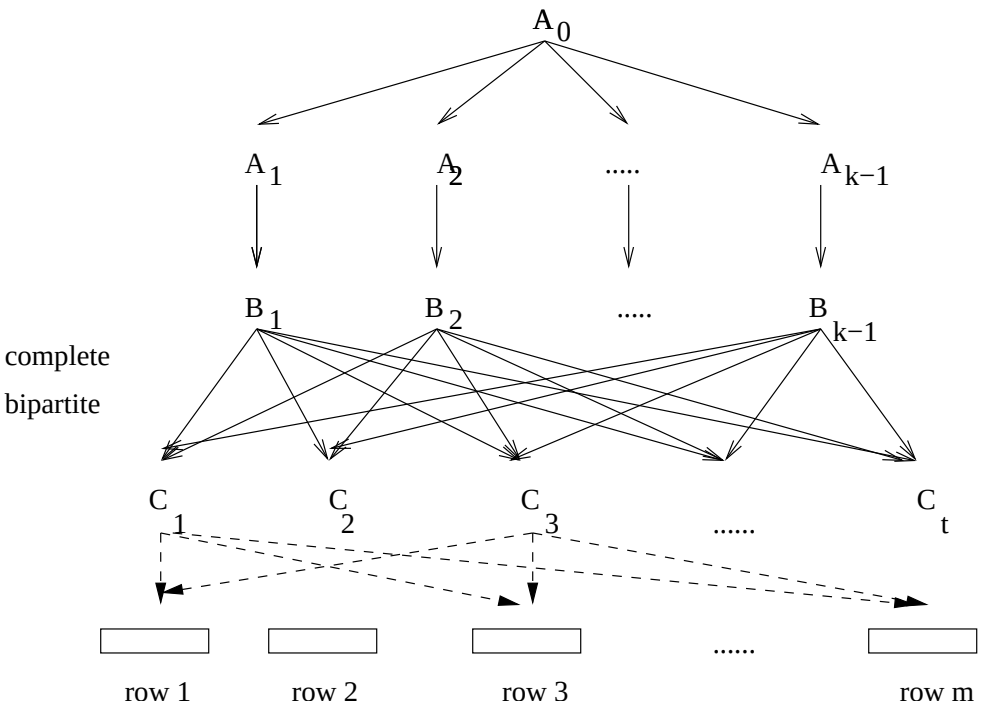

Figure 2.1: An instance of HOTLINK ASSIGNMENT. The i-th rectangular box at the bottom row represents the row $s_{i,1}, s_{i,2}, \ldots, s_{i,k}$, for $i = 1, 2, \ldots, m$. If $s_i \in C_j$ then there is a directed edge from C_j to each of the vertices of the i-th row.

2. $A_1, \ldots, A_{k-1}$. $B_1, B_2, \ldots, B_{k-1}$, and $C_1, C_2, \ldots, C_t$,
3. $s_{i,1}, s_{i,2}, \ldots, s_{i,k}$, for $i = 1, 2, \ldots, m$, are m rows of vertices each row of size k.

- The edge set E consists of the directed edges below:

1. (A_0, A_i), and (A_i, B_i), for all $i = 1, \ldots, k-1$,
2. (B_i, C_j), for all $i = 1, \ldots, k-1$, and $j = 1, 2, \ldots, t$,
3. if $s_i \in C_j$ then there exist directed edges $(C_j, s_{i,1}), (C_j, s_{i,2}), \ldots, (C_j, s_{i,k})$.

The directed graph G is depicted in Figure 2.1. We can prove the following lemma.

Lemma 1. *There is a set cover of size at most k if and only if the directed graph G has a hotlink assignment which attains a gain of at least $mk + 3k$.*

Proof. The lemma implies the theorem. To prove the lemma it is enough to prove the following two claims.

Claim 1: If there is an exact cover of S by 3-sets then there is a hotlink assignment for the graph G whose gain is at least $mk + 3k$.

Indeed, let $C_{i_0}, C_{i_1}, \ldots, C_{i_{k-1}}$ be such a set cover of the set S. Consider the following hotlinks.

$$A_0 \to C_{i_0}, A_1 \to C_{i_1}, \ldots, A_{k-1} \to C_{i_{k-1}}.$$

The gain resulting from the hotlink $A_0 \to C_{i_0}$ is $2(3k)$ and the gain resulting from the remaining $k-1$ hotlinks is $(m-3)k$. Hence the total gain is $mk + 3k$.

Claim 2: If there is no set cover of size k then the gain from any hotlink assignment on the graph G is less than $mk + 3k$.

Indeed, consider any hotlink assignment of G. On the one hand, if this hotlink assignment is using a hotlink from A_0 to some vertex B_i, for some $i = 1, 2, \ldots, k-1$, then the maximum gain that can be attained is at most $mk + 2(k-1) + (k-1) <$

$mk + 3k$. On the other hand, if this hotlink assignment is never using a hotlink from A_0 to any vertex B_i, for any $i = 1, 2, \ldots, k-1$, then again the maximum gain that can be attained is less than $2(3k) + (m-4)k + k - 1 < mk + 3k$.

■

This completes the proof of the lemma and hence also of the theorem.

■

We conclude this section with a useful observation that shows that the hotlink assignment problem on arbitrary graphs is reducible to the hotlink assignment problem on trees. Observe that for a given hotlink assignment, we can "selectively drop edges" without altering the average length of a shortest path from the source node to the destinations in such a way that the resulting underlying graph is a tree rooted at the source node s. More precisely we have the following result.

Lemma 2. *Let $G = (V, E)$ be a digraph with real weights associated to its vertices and some designated source vertex s such that every vertex of G is reachable from s. There exists a subgraph $G' = (V, E')$ such that $E' \subseteq E$ and G' is a tree and the average shortest path from s is the same for G and G'.*

Proof. We prove by contradiction that a subgraph G' of G with minimal number of edges and the same expected length must be a tree. Note that no incoming edge of s can be in G'. If G' is not a tree and the indegree of s is 0, some vertex v must have at least two incoming edges $e_1, e_2 \in E'$. We can discard at least one of these, thus contradicting the minimality of G'. Indeed, if the shortest path tree from s to v uses, say, edge e_1, then edge e_2 is never used in the shortest path from s to v or any of its descendants.

■

2.4 NP-Hardness of the Bookmark Assignment Problem

In this section we prove that the *optimal bookmark assignment problem* is NP-hard even in arbitrary directed acyclic graphs with uniform distributions. Consider the following optimization problem.

BOOKMARK ASSIGNMENT

Instance: Directed graph $G = (V, E)$ and positive integer $k < |V|$.

Task: Find a bookmark set $B \subset V$ of size k maximizing $\mathcal{G}(B)$.

Theorem 2. *The problem BOOKMARK ASSIGNMENT is NP-hard.*

Proof. The transformation is from the following NP-complete decision problem [23].

MINIMUM COVER

Instance: Collection C of subsets of a finite set S, positive integer $K \leq |C|$.

Question: Does C contain a cover for S of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \leq K$, such that every element of S belongs to at least one member of C'?

Consider an instance of MINIMUM COVER, let $C = \{c_1, ..., c_t\}$ and construct the following directed graph $G = (V, E)$. Let $A = \{a_1, ..., a_t\}$, $X = \{x_1, ..., x_t\}$ and $Y = \{y_1, ..., y_t\}$ be sets such that A, X, Y, S are pairwise disjoint. Let r be an element outside of $A \cup X \cup Y \cup S$. Define $V = A \cup X \cup Y \cup S \cup \{r\}$. The set E of arcs is defined as follows (cf. Fig 2.2). $E_1 = \{(a_i, s) : s \in c_i, i = 1, ..., t\}$, $E_2 = \{(x_i, a_i) : i = 1, ..., t\}$, $E_3 = \{(y_i, x_i) : i = 1, ..., t\}$, $E_4 = \{(r, y_i) : i = 1, ..., t\}$. Finally, $E = E_1 \cup E_2 \cup E_3 \cup E_4$.

The construction of graph $G = (V, E)$ can be done in polynomial time. Now, let $k = K \leq t$ and solve BOOKMARK ASSIGNMENT for the instance G, k. Let B be the obtained optimal bookmark set of size k on G. Hence B has maximum gain among all sets of size k. Observe that $B \subseteq A \cup X \cup S$ (bookmarks placed in $\{r\} \cup Y$ would be wasted). We now construct a bookmark set $B^* \subseteq A \cup S$ whose value is not smaller than that of B. Suppose that $b = x_i \in B$. There are two cases.

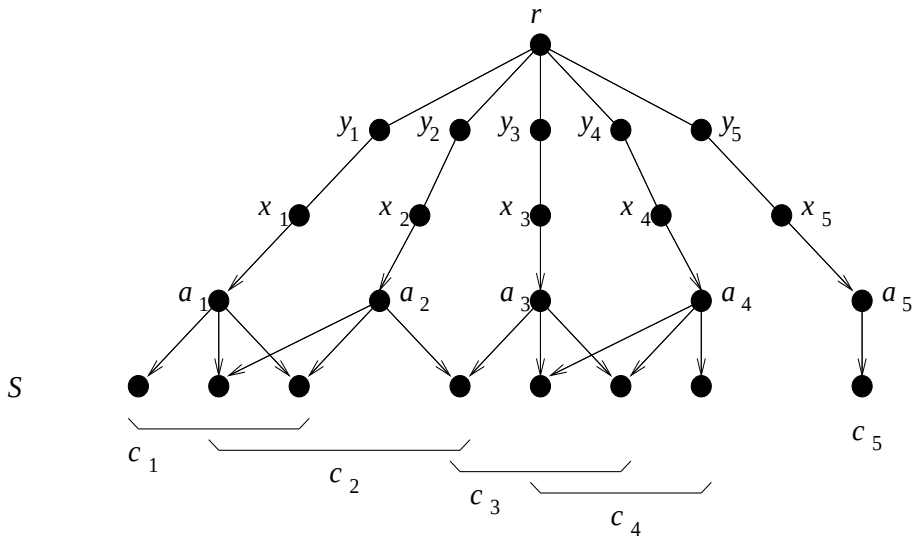

Figure 2.2: An instance of BOOKMARK ASSIGNMENT.

Case 1. $a_i \in B$. Since $|B| \leq t$, there exists an index $j \leq t$ for which none of the elements a_j, x_j, y_j belongs to B. Replacing b by a_j increases the value of the bookmark set because a bookmark at a_j contributes at least 2 to the gain, while a bookmark at b contributes only 1 in this case.

Case 2. $a_i \notin B$. Replacing b by a_i does not decrease the value of the bookmark set. Indeed, let α be the number of elements in c_i which do not belong to any set c_l with $a_l \in B$ and which are not in B. Then a bookmark at b contributes at most $2+\alpha$ to the gain, while a bookmark at a_i contributes $2(1+\alpha)$. Clearly, $2+\alpha \leq 2(1+\alpha)$.

A bookmark set B^* can be obtained from B in polynomial time by a repeated application of changes described above. It follows that if BOOKMARK ASSIGNMENT can be solved in polynomial time then a bookmark set $B^* \subseteq A \cup S$ of maximum gain among all bookmark sets of size k can also be obtained in polynomial time.

Now observe that, in fact, $B^* \cap S = \emptyset$. Indeed, suppose that some $s \in S$ belongs to B^*. Since $|B^*| \leq t$, there exists an index $j \leq t$ for which none of the elements a_j, x_j, y_j belongs to B^*. There are two cases. If s belongs to some set c_l with $a_l \in B^*$

then replacing s by a_j increases the gain of the bookmark set because a bookmark at s contributes only 1 to the gain, while a bookmark at a_j contributes at least 2. If s does not belong to any set c_l with $a_l \in B^*$ then choose any c_l such that $s \in c_l$. In this case replacing s by a_l increases the gain of the bookmark set because a bookmark at s contributes at most 3 to the gain, while a bookmark at a_l contributes at least $2 \cdot 2 = 4$. Hence in both cases the gain of B^* could be strictly increased which contradicts the optimality of B^*. This contradiction proves that $B^* \cap S = \emptyset$ and consequently $B^* \subseteq A$.

Suppose that $B^* = \{a_1, ..., a_k\}$. We have $\mathcal{G}(B^*) = 2(k + |c_1 \cup ... \cup c_k|)$. It follows that the family $\{c_1, ..., c_k\}$ maximizes union size among all subfamilies of C of size k. Consequently, C has a subfamily C' of size k or less covering S, if and only if $c_1 \cup ... \cup c_k = S$. Observe that $c_1 \cup ... \cup c_k$ can be computed in polynomial time knowing B^*. Hence MINIMUM COVER can be solved in polynomial time. This proves that BOOKMARK ASSIGNMENT is NP-hard.

■

2.5 From Web Sites to Trees

We have seen that the optimal assignment of hotlinks (and bookmarks) is a NP-hard problem for arbitrary directed acyclic graphs. In this dissertation we approach the problem for trees. This approach consists in reducing the access cost of the Web site by reducing the access cost of its leaves. Now, how do we determine which pages are leaf pages? Some authors use "maximal forward paths" in their approach to improve Web sites, e.g., [14], [52] and [7]. A forward path is a sequence of pages visited by a single user in a single session until the user goes back to a previously visited page in the same session. We may use maximal forward paths to determine which pages are leaf pages. A leaf page would be a page that is at the end of a path which was visited

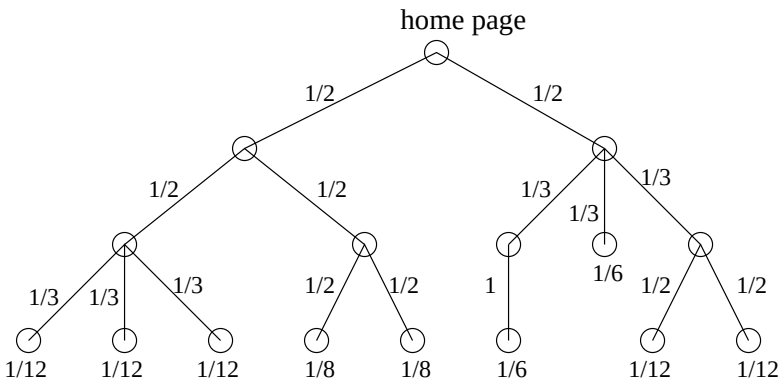

Figure 2.3: Equivalence of having weights on hyperlinks to having weights on pages.

at least a particular number of times. We also have the option of using an approach similar to the one presented by Fu et al. [22] to find leaf pages. Fu et al. classify pages into "index pages" and "content pages" according to the number of hyperlinks contained in a page. These methods for finding leaf pages are inaccurate and may not provide the shortest path from the home page. We determine what pages are leaf pages by performing breadth first search on the Web site beginning at the home page. By performing breadth first search we ensure that the path from the home page to the leaves is always the shortest one.

Suppose we found the set of leaf pages $L = \{l_1, l_2, ..., l_m\}$ of a Web site $G = (V, E)$, such that $L \subseteq V$ and no page $l_i \in L$ is in the shortest path between the home page and a page $l_j \in L, \forall i \neq j$. If we dropped the edges that are not in the shortest path from the home page to each leaf of L we would get a tree T. Suppose that we arbitrarily assign real weights to the leaves of T and that the weight of a page $v \notin L$ is equal to the sum of the weights of its descendant leaves. Observe that $E[G] = E[T]$.

It is also possible to assign weights to the pages if we are only provided with weights on the hyperlinks. In this situation the weight of a leaf page would be the product of the weights of the hyperlinks traversed in the shortest path between this page and the home page. This assignment of weights is illustrated in Figure 2.3.

Part II

Related Contributions: Full Binary Trees

Chapter 3

Assignment of Hotlinks to Full Binary Trees

3.1 Introduction

In this chapter we study the assignment of at most one hotlink per node in full binary trees with different probability distributions over the leaves. We present hotlink assignment algorithms and provide upper and lower bounds on the corresponding access cost that can be achieved. We consider a full binary tree of $N = 2^{n+1} - 1$ nodes with a probability distribution p over its leaves. The content of this chapter is also presented in [8].

3.2 Lower Bounds

A *code alphabet* is any nonempty set of letters. A *code word* in the code-alphabet is a word (i.e., any concatenation) formed from letters of the code-alphabet; the number of letters in the code-word is the length of the code-word. A *code* is any set of code-words. The code is called a *prefix-code* if no code-word in the code is a prefix of any

other code-word in the same code.

To prove lower bounds we use Shannon's theory and the entropy $\mathcal{H}(p)$ (see [1] for a complete description of entropy) of the probability distribution p, which is defined by Equation 3.1.

$$\mathcal{H}(p) = \sum_{i=1}^{N} p_i \log \frac{1}{p_i} \tag{3.1}$$

Shannon's theorem states the following.

Theorem 3. *Let $p_1, p_2, \ldots, p_m$ be a probability distribution. Given a prefix code $w_1, w_2, \ldots, w_m$ of respective lengths $l_1, l_2, \ldots, l_m$ in an alphabet of size s, the expected length $\sum_{i=1}^{m} l_i p_i$ of the code is at least $\mathcal{H}(p)/\log s$.*

A binary tree can be thought of as the encoding of the leaves with the two symbol alphabet $0, 1$. Adding a hotlink per node increments the alphabet by a single symbol to form a three symbol alphabet.

Consider a tree $T = (V, E)$. For a given hotlink assignment A, the distance of the i-th leaf from the root in T^A, $d(i)$, is the length of the encoding of the i-th leaf in this new alphabet. Notice that if two hotlinks are targeting the same node then the shortest one can be omitted without changing the value of $E[T^A]$. As a consequence, $E[T^A]$ is also the expected length of the "encoding" of the leaves of the tree T^A represented as code-words in a three letter alphabet. Moreover, the resulting encoding is a prefix code. In particular, Shannon's theorem applies and we have that

$$E[T^A] \geq \frac{1}{\log 3} \cdot \mathcal{H}(p) = \frac{1}{\log 3} \cdot \sum_{i=1}^{m} p_i \log(1/p_i) \tag{3.2}$$

We have proven the following theorem.

Theorem 4. *For any probability distribution p on the leaves of a full binary tree and any assignment of at most one hotlink per source node the expected number of steps*

to reach a Web page located at a leaf from the root of the tree is at least $\mathcal{H}(p)/\log 3$*, where* $\mathcal{H}(p)$ *is the entropy of* p*.*

Theorem 4 is directly applicable. Straightforward calculations give lower bounds for several distributions. For the uniform distribution we see that $\mathcal{H}(p) = n$. For the Zipf's distribution,

$$\begin{aligned}\mathcal{H}(p) &= \textstyle\sum_{i=1}^{m} p_i \log(1/p_i) \\ &= \frac{1}{H_m} \textstyle\sum_{i=1}^{m} \frac{\log i}{i} + \log(H_m) \\ &\geq \frac{1}{2} \cdot \log n + \Omega(\log \log n) \\ &= \frac{n}{2} + \Omega(\log n)\end{aligned}$$

In particular, this implies the $n/(2\log 3)$ lower bound on the expected number of steps for the Zipf's distribution.

3.3 Geometric Distribution

Recall the geometric distribution, $p_i = a^{-i}$, for $i = 1, ..., m-1$, and $p_m = 2 - \frac{1-a^{-m}}{1-a^{-1}}$, where $a > 1$.

We can assign hotlinks by a simple bottom-up heuristic. The idea is to sort the nodes in order of decreasing probabilities and assign to a leaf a source node not assigned before which is the furthest from it so that the leaf is in the subtree rooted at the source node. This simple heuristic will assign at most one leaf per node and can be efficient if the probability distribution is concentrated in "a few" nodes, as is the case of the geometric distribution. The algorithm is described in Algorithm 1.

A simple analysis of the performance of the algorithm is as follows. Let $l_1, l_2, \ldots, l_{n-1}$ be the $n-1$ leaves of the 1st, 2nd,..., $(n-1)$st highest probability in the given probability distribution p. The resulting gain is

Algorithm 1 bottomUpStrategy(T)

1. *sort the* m *leaves,* l_i*, by the size of the probability distribution* p_v*, i.e.,* $p_{l_i} \geq p_{l_{i+1}}$ *for* $i < m$*.*
2. $H = \phi$
3. *for* $j = 1$ *to* $j = m - 1$
   (a) *find an internal yet unassigned node* u *such that* l_j *is a leaf in the subtree rooted at* u *and the distance from* u *to* l_j *is maximized*
   (b) *if* $u \neq \phi$ *and* $l_j \neq \phi$ *then*
      i. $H = H \cup (u, l_j)$

$$\begin{aligned} \mathcal{G}(H) &\geq (n-1)p_{l_1} + (n-2)p_{l_2} + \cdots + (n-(n-1))p_{l_{n-1}} \\ &= n\sum_{i=1}^{n-1} p_{l_i} - \sum_{i=1}^{n-1} ip_{l_i} \end{aligned}$$

In particular, the expected number of steps satisfies

$$\begin{aligned} E[T^H] &\leq n - n\sum_{i=1}^{n-1} p_{u_i} + \sum_{i=1}^{n-1} ip_{u_i} \\ &= n\left(1 - \sum_{i=1}^{n-1} p_{u_i}\right) + \sum_{i=1}^{n-1} ip_{u_i} \end{aligned} \tag{3.3}$$

Inequality 3.3 can already provide good bounds for probability distributions which are heavily concentrated in "a few" nodes. In particular, we have the following result.

Theorem 5. *Consider a rooted full binary tree on* n *levels with a geometric probability distribution on its leaves. Algorithm* bottomUpStrategy *is linear in the number of vertices of the tree and assigns at most one hotlink per node in such way that the expected number of steps to reach a leaf of the tree is* $O(1)$*.*

Proof. This follows from Inequality 3.3. In particular, for the geometric distribution

$$\begin{aligned} E[T^H] &\leq n\left(1 - \sum_{i=1}^{n-1} p_{u_i}\right) + \sum_{i=1}^{n-1} i p_{u_i} \\ &= n\left(1 - \sum_{i=1}^{n-1} a^{-i}\right) + \sum_{i=1}^{n-1} i a^{-i} \\ &= O(1) \end{aligned}$$

This completes the proof of the theorem.

■

3.4 Arbitrary Distributions

In this section we give an algorithm for arbitrary distributions. The algorithm is described in Algorithm 2. The set of hotlinks H must be initially empty.

Algorithm 2 lengthTwoHotlinks (T, r)

1. *find the grandchild v of r of maximal weight whose parent has not been assigned a hotlink*
2. *if $v \neq \phi$ then*
   (a) $H = H \cup (r, v)$
   (b) *let u be the parent of v*
   (c) *let x be the sibling of u*
   (d) *if x has not been assigned a hotlink then*
      i. lengthTwoHotlinks(T, x)
   (e) lengthTwoHotlinks(T, u)
   (f) lengthTwoHotlinks(T, v)

Theorem 6. *Consider a rooted full binary tree on n levels with any probability distribution on its leaves. Algorithm* lengthTwoHotlinks *is linear in the number of vertices*

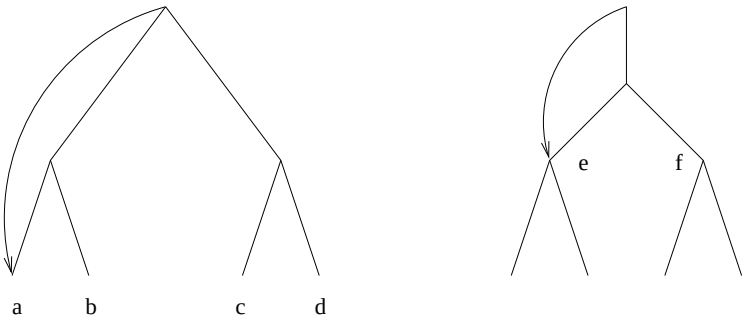

Figure 3.1: Assignment of hotlinks to subtrees. The leftmost tree is T_n and the rightmost tree is S_n.

of the tree and assigns a hotlink per node in such a way that the expected number of steps to reach a leaf of the tree is $(3n+1)/4$.

Proof. As mentioned before in $O(N)$ time we can propagate the original weights on the leaves of the tree through the entire tree using a bottom-up process. Once all these internal node weights are assigned we use a top-down method to assign hotlinks. Each hotlink is of length two, i.e., each time from level $i \geq n-2$ to level $i+2$. The root is assigned the hotlink to the level two node of highest weight. By symmetry we can suppose that this hotlink is to the leftmost descendant at distance two from the root (cf. T_n in Figure 3.1).

The assignment of hotlinks is done recursively. The recursive process assigns hotlinks to the two subtrees T_n and S_n as depicted in Figure 3.1. Let a, b, c, d be the probabilities at the second level of the tree T_n and e, f the probabilities at the second level of the tree S_n. Without loss of generality assume that

$$a \geq b, c, d \text{ and } e \geq f,$$

We select the hotlinks from the root to the node with highest probability at the second level of T_n and S_n, respectively. This leads to the hotlink assignments depicted

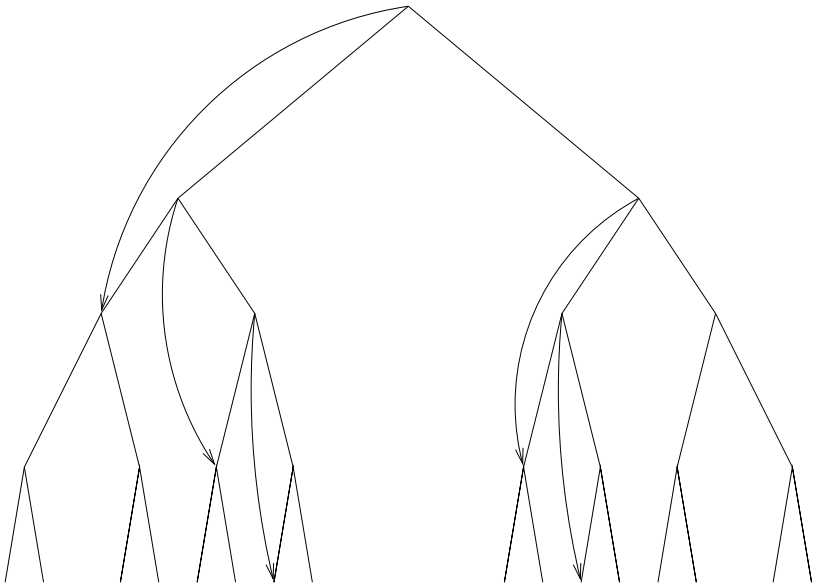

Figure 3.2: The iteration of the assignment of hotlinks by algorithm *lengthTwoHotlinks*, described in Algorithm 2.

in Figure 3.1. The overall algorithm is illustrated in Figure 3.2. Notice that by assumption we have that

$$a + b + c + d = 1$$
$$e + f = 1$$

Let s_n (respectively, t_n) be the expected number of steps to reach the leaves of the tree S_n (respectively T_n), We now have the recurrences

$$\begin{aligned} t_n &= 1 + (c+d)t_{n-1} + at_{n-2} + bs_{n-1} \\ s_n &= 1 + et_{n-2} + fs_{n-2}, \end{aligned}$$

where t_n and s_n are the weights of the subtrees T_n and S_n, respectively. We can prove the following claim.

Claim:

$$\begin{aligned} t_n &\leq 3n/4 + 1/4 \\ s_n &\leq 3n/4 \end{aligned}$$

Proof of the claim: The proof is by induction on n. Cases $n = 2, 3, 4$ can be proven by inspection. First we consider s_n.

$$\begin{aligned} s_n &= 1 + et_{n-2} + fs_{n-2} \\ &\leq 1 + e(3(n-2)/4 + 1/4) + f3(n-1)/4 \\ &= 1 + (e+f)3n/4 - 5e/4 - 3f/4 \\ &\leq 1 + 3n/4 - (e+f)3/4 - e/2 \\ &\leq 1 + 3n/4 - 1 \\ &= 3n/4, \end{aligned}$$

the last inequality being true because by assumption $e \geq 1/2$.

Next we consider t_n.

$$\begin{aligned} t_n &= 1 + at_{n-2} + (c+d)t_{n-1} + bs_{n-1} \\ &\leq 1 + a(3(n-2)/4 + 1/4) + (c+d)(3(n-1)/4 + 1/4) + 3(n-1)/4 \\ &= 1 + (a+b+c+d)3n/4 - 5a/4 - 3b/4 - (c+d)/2 \\ &= 1 + 3n/4 - 3a/4 - b/4 - (a+b+c+d)/2 \\ &= 1/2 + 3n/4 - 3a/4 - b/4 \end{aligned}$$

Now we have two cases depending on the size of a.

Case 1: $a \geq 1/3$.

In this case, the claim follows from the inequalities

$$\begin{aligned} t_n &\leq 1/2 + 3n/4 - 3a/4 - b/4 \\ &\leq 1/2 + 3n/4 - 1/4 \\ &\leq 1/4 + 3n/4 \end{aligned}$$

Case 2: $a < 1/3$.

In this case, write $a = 1/3 - x$, where $x > 0$, and notice that

$$b + c + d = 1 - a = 2/3 + x$$

and

$$c \leq a = 1/3 - x$$
$$d \leq a = 1/3 - x$$

Consequently,

$$b = 2/3 + x - c - d \geq 3x.$$

It follows that

$$\begin{aligned} t_n &\leq 1/2 + 3n/4 - 3a/4 - b/4 \\ &\leq 1/2 + 3n/4 - 3/4(1/3 - x) - 3x/4 \\ &= 1/4 + 3n/4. \end{aligned}$$

This completes the proof of Case 2 and hence also of the claim. The proof of Theorem 6 is now complete.

■

3.5 Uniform Distribution

Recall the uniform distribution, $p_1 = p_2 = \cdots = p_m = \frac{1}{m}$.

Consider the uniform distribution in a full binary tree. Adding a hotlink from the root to a node at distance d will save exactly a fraction of $(d-1)2^{-d}$ of the total weight of the tree. Moreover this saving is maximized at $d = 2$ or $d = 3$ and the maximum value is $1/4$. Similarly, any hotlink we add to a node can save at most $1/4$ of the total weight of the subtree rooted at this node. This indicates that the maximal gain attained by a hotlink on a given node is at most $1/4$ of the weight of the subtree rooted at this node. Therefore it is not difficult to see that by adding one

hotlink per node on a tree with the uniform distribution we can never attain a gain higher than $(n-1)/4$. The previous discussion and Theorem 6 imply the following result.

Theorem 7. *Consider a rooted full binary tree on n levels with a uniform probability distribution on its leaves. Algorithm* lengthTwoHotlinks *(Algorithm 2) is linear in the number of vertices of the tree and assigns at most one hotlink per node in such a way that the expected number of steps to reach a leaf of the tree is at most $(3n+1)/4$. Moreover, $(3n+1)/4$ is a lower bound on any algorithm designed for the same purpose.*

3.6 Zipf's Distribution

Recall the Zipf's distribution $p_i = \frac{1}{iH_m}$, where $H_m = \sum_{i=1}^{m} 1/i$ is the harmonic number.

In this section we consider the sorted Zipf's distribution, i.e. the i-th leaf of the tree is assigned probability $\frac{1}{iH_m}$, and prove an $n/3 + O(\sqrt{n}\log n)$ upper bound on the expected number of steps to reach a leaf.

Recall the following estimate for the harmonic number $H_j = \sum_{i=1}^{j} 1/i$ from [32] (page 75),

$$H_j = \ln j + \gamma + \frac{1}{2j} - \frac{1}{12j^2} + \frac{1}{120j^4} - \epsilon,$$

where $0 < \epsilon < \frac{1}{252j^6}$ and $\gamma = 0.5772\ldots$ is Euler's constant. Consequently we have that

$$\gamma + \ln j < H_j < 1 + \ln j$$

It follows that for $l \leq n$ we have the Inequalities

$$\frac{l}{n} + \frac{1}{n} \geq \sum_{i=1}^{2^l} p_i = \sum_{i=1}^{2^l} \frac{1}{iH_m} = \frac{1}{H_m} \sum_{i=1}^{2^l} \frac{1}{i} = \frac{H_{2^l}}{H_{2^n}} \geq \frac{l}{n} \cdot \left(\frac{n}{n+2}\right) \tag{3.4}$$

We call our algorithm *sortedZipfHotlinks*. It is described in Algorithm 3.

Algorithm 3 sortedZipfHotlinks(T)

1. *define the sequence* $r_0, r_1, \ldots, r_j$ *of nodes, where* $j \leq n$, *such that* r_0 *is the root of the tree, then for each* i, r_i *is the left child of* r_{i-1}
2. $i = 0$
3. *while* $i < \lceil \frac{n}{2^{i+1}} \rceil$
   (a) $H = H \cup (r_i, r_{\lceil \frac{n}{2^{i+1}} \rceil})$
   (b) $i = i + 1$

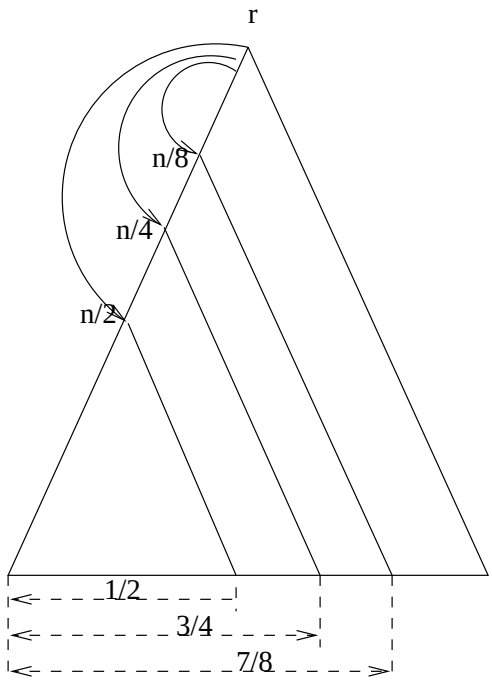

Figure 3.3: The iteration of the assignment of hotlinks by algorithm *sortedZipfHotlinks*, described in Algorithm 3.

Theorem 8. *Consider a rooted full binary tree on n levels with the sorted Zipf probability distribution on its leaves. Algorithm* sortedZipfHotlinks *is linear in the number of vertices of the tree and assigns at most one hotlink per node in such a way that the expected number of steps to reach a leaf of the tree is at most $n/3 + O(\sqrt{n}\log n)$.*

Proof. The weight of the subtree rooted at $r_{\lceil n/2^{i+1}\rceil}$ is equal to $\sum_{i=1}^{2^{l_{i+1}}} p_i$, where $l_{i+1} = n - \lceil n/2^{i+1}\rceil$. Since $n/2^{i+1} \leq \lceil n/2^{i+1}\rceil \leq 1 + n/2^{i+1}$ we can use Inequality 3.4 to obtain that

$$\left(1 - \frac{1}{2^{i+1}}\right)\frac{n}{n+2} - \frac{1}{n+2} \leq \sum_{i=1}^{2^{l_{i+1}}} p_i \leq \left(1 - \frac{1}{2^{i+1}}\right) + \frac{2}{n} \tag{3.5}$$

In view of Inequality 3.5, it is not difficult to see that the gain resulting from adding a single hotlink from the node r_i to the node $r_{\lceil n/2^{i+1}\rceil}$ is at least

$$\left(\frac{n}{2^{i+1}} - i - 1\right)\sum_{i=1+2^{l_i}}^{2^{l_{i+1}}} p_i \geq \left(\frac{n}{2^{i+1}} - i - 1\right)\cdot\frac{1}{2^{i+1}}\cdot\left(\frac{1}{2^{i+1}} - \frac{5}{n}\right) \tag{3.6}$$

The hotlink assignment is illustrated in Figure 3.3. By summing the terms in Equation 3.6 for i such that $i < \lceil n/2^{i+1}\rceil$, we obtain that the gain resulting from all these hyperlinks is at least

$$\left(\frac{n}{2} - 1\right)\cdot\left(\frac{1}{2} - \frac{5}{n}\right) + \left(\frac{n}{4} - 2\right)\cdot\left(\frac{1}{4} - \frac{5}{n}\right) + \left(\frac{n}{8} - 3\right)\cdot\left(\frac{1}{8} - \frac{5}{n}\right) + \cdots \tag{3.7}$$

Let $k := \lceil(\log n)/3\rceil - 1$. Clearly, k satisfies $k2^{k+1} \leq \sqrt{n} < n$. Summing Equation 3.7, we obtain that the total gain achieved from all these hotlinks is at least

$$n\sum_{i=1}^{k}\frac{1}{2^{2i}} - \sum_{i=1}^{k}\left(\frac{i}{2^i} - \frac{5}{n}\right) - \frac{5k}{2} \geq \frac{n}{3} - O(\sqrt{n}) \tag{3.8}$$

Let $c > 0$ be the constant in Inequality 3.8. We now apply the previous hotlink

assignment recursively to each subtree rooted at the node $r_{\lceil n/2^i \rceil}$, respectively, for $i \leq l = \lfloor \log n \rfloor$. Using Inequality 3.8, we can see that the total gain achieved must be at least

$$\sum_{i=0}^{l-1} \left(\frac{n2^{-i}}{3} - c\sqrt{n/2^i} \right) \geq \frac{n}{3} \sum_{i=0}^{k-1} 2^{-i} - c\sqrt{n} \log n \geq \frac{2n}{3} - O(\sqrt{n} \log n)$$

It follows that after adding these hotlinks to the tree the expected length of a path to reach a leaf is at most

$$n - \frac{2n}{3} + O(\sqrt{n} \log n) = \frac{n}{3} + O(\sqrt{n} \log n)$$

Thus, we have proven Theorem 8.

■

Chapter 4

Assignment of Bookmarks to Full Binary Trees

4.1 Introduction

An important way of improving the response rate of large, distributed documents is via a careful and methodical design of bookmark assignments to Web pages. Recall that a bookmark is a hotlink that connects the home page with any other page of the Web site.

In this chapter, we study the assignment of k bookmarks to a full binary tree with uniform probability distribution over the nodes[1]. Consider a full binary tree $T = (V, E)$ with $|V| = N = 2^{n+1} - 1$ nodes and a uniform probability distribution over the nodes $v \in V$. Our work on bookmarks is also presented in [17].

Consider a Web site consisting of a collection $V = \{v_1, ..., v_N\}$ of Web pages connected by hyperlinks. These hyperlinks have been placed a priori by design in the initial construction of the Web pages. Assume that there exists a directed path

[1]Recall that in the *optimal bookmark assignment problem* we assume that the probability is distributed over all the nodes of the tree.

of hyperlinks from the home page r to any other page of the collection. A *bookmark* is an additional hyperlink from the home page r to any other page of the domain. Let p_v be the probability that a user, currently located at r, wants to access page v. Assuming this, we pose the following *bookmark assignment problem* (k-BAP): Find an assignment of k bookmarks in a Web site with known access probabilities, that minimizes the expected number of steps required to reach any page of the domain from the home page. See Section 2.2.3 for a full description of the problem.

4.2 Characterizing an Optimal Assignment of Bookmarks with Uniform Probability Distribution

We need some more definitions. Consider a set of bookmarks B assigned to tree $T = (V, E)$. We say that a bookmark $x \in V$ *dominates* node $v \in V$, if on the path $(x = v_0, v_1, \cdots, v_l = v)$ only x may belong to B. When a bookmark b dominates v, the shortest path from root r to v uses the edge (r, b). The set of all nodes of V dominated by b is called the *domain* of b. We say that two bookmarks b_1 and b_2 are *independent* if there is no directed path from one to the other. Observe that the domains of two bookmarks b_i and b_j are always disjoint and the family of domains of the set $B \cup r$ partitions V. We say that B *covers* T if each leaf v of T belongs to the domain of some $b_i \in B$. A bookmark $b \in B$ is called *exposed* in T^B if its domain does not contain any other bookmark of B. For $l \geq 0$, the *level* l of tree T consists of all nodes at distance l from r. If $l < q$, we say that l is *above* q or q is *deeper* than l. The *depth* of the tree is the number of its deepest level, n.

We construct an optimal assignment of k bookmarks in the complete binary tree of $N \geq 15$ nodes with uniform probability distribution, under some condition relating k to N. First observe that if $k \leq 4$ then the optimal assignment of bookmarks is to place all of them on level 2, since bookmarks on level 0 and 1 would be wasted.

Hence we may assume $k > 4$.

Theorem 9. *For a complete binary tree T of $N \geq 15$ nodes, a set B of k bookmarks, such that $4 < k \leq \sqrt{N+1}$, is optimal, if and only if all of the following conditions hold.*

1. *B covers T,*
2. *all bookmarks in B are independent,*
3. *all bookmarks in B are placed on two consecutive levels of T.*

The existence of a set of bookmarks satisfying the above conditions is guaranteed by the following lemma.

Lemma 3. *For a complete binary tree T with m leaves and for any number $k \leq m$, there exists a set of k independent bookmarks covering T, placed on at most two consecutive levels of T.*

Proof. Let $l = \lfloor \log(k) \rfloor$. If we take any set S of $2^{l+1} - k$ bookmarks on level l, we can place exactly $2k - 2^{l+1}$ bookmarks independent on S on level $l+1$. They all cover T. ■

Note that the distribution of bookmarks given in Lemma 3 is unique up to the choice of its subset on level l. Thus in the remainder of this section we will suppose without loss of generality that in the set of bookmarks from Theorem 9 the bookmarks from the higher level are the leftmost nodes of T on this level (cf. Figure 4.1). Theorem 9 implies the following corollary.

Corollary 1. *Under the assumptions of Theorem 9, an optimal set of bookmarks can be constructed in time $O(k)$.*

In order to prove Theorem 9, we formulate several lemmas showing which transformations of a bookmark set increase its gain.

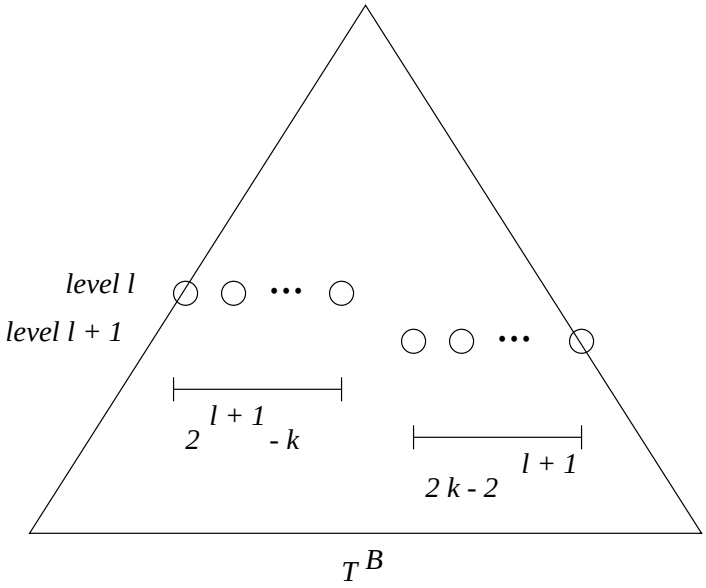

Figure 4.1: The set B of bookmarks from Theorem 9. B covers T and all the bookmarks are independent and placed on two consecutive levels.

Lemma 4. *Let B_i be a set of bookmarks defined for a tree T. Suppose that B_i has been altered by replacing some of its bookmarks $b_{i_1}, b_{i_2}, \ldots, b_{i_m}$ by $b_{j_1}, b_{j_2}, \ldots, b_{j_m}$ thus forming a new bookmark set B_j. For each node v which is not in the domain of any node from $X = \{b_{i_1}, b_{i_2}, \ldots, b_{i_m}, b_{j_1}, b_{j_2}, \ldots, b_{j_m}\}$, we have $g_{B_i}(v) = g_{B_j}(v)$.*

Proof. The result follows because the shortest oriented path from r to v in T^{B_1} and T^{B_2} must use exactly the same edges.

■

Lemma 4 states that, when replacing one bookmark set by another, $g_B(v)$ may change only for the nodes v which are in the subtrees rooted at the nodes of X. Let $E[T_{-X}]$ denote the expected number of steps to reach a node in the tree T diminished by the nodes of X and their descendants. Let $\mathcal{G}_{-X}(B)$ denote the gain attained by the set of bookmarks B in the tree diminished by the nodes of X and their descendants. The statement of Lemma 4 then says that $E[T^{B_i}_{-X}] = E[T^{B_j}_{-X}]$ and $\mathcal{G}_{-X}(B_i) = \mathcal{G}_{-X}(B_j)$.

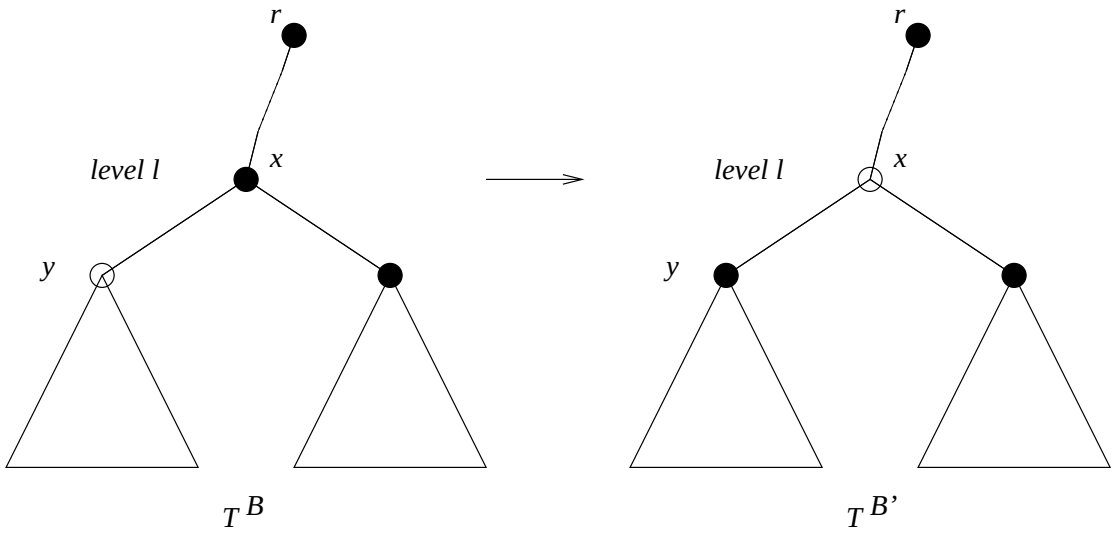

Figure 4.2: *Transformation lift.* If $B' = B \setminus \{y\} \cup \{x\}$ then $\mathcal{G}(B') > \mathcal{G}(B)$.

Lemma 5. ***Transformation lift.*** *Suppose that for a set of bookmarks B on T there exists a node $x \notin B$ on level $l \geq 2$, such that, among the descendants of x, the only bookmark $y \in B$ is a child of x. Then for $B' = B \setminus \{y\} \cup \{x\}$, we have $\mathcal{G}(B') > \mathcal{G}(B)$ (cf. Figure 4.2).*

Proof. Let w be the number of nodes in the subtree rooted at y and s be the distance from r to x in T^B.

Note that if there is a bookmark on the oriented path from r to x then $s < l$, otherwise $s = l$. By Lemma 4, $\mathcal{G}_{-\{x\}}(B) = \mathcal{G}_{-\{x\}}(B')$ because $g_B(v) = g_{B'}(v)$, for each node v not in the subtree rooted at x. Therefore, to evaluate the change in the gain function between the sets B and B' we should consider only nodes in the subtree rooted at x. Hence, $g_B(v) = l$, for all w nodes in the subtree rooted at y and $g_B(v) = l - s$, for x and all remaining w descendants of x, while $g_{B'}(v) = l - 1$, for all $2w + 1$ nodes in the subtree rooted at x. Since $l \geq s \geq 2$ we conclude that $\mathcal{G}(B) = \mathcal{G}_{-\{x\}}(B) + lw + (l - s)(w + 1) < \mathcal{G}_{-\{x\}}(B) + (l - 1)(2w + 1) = \mathcal{G}(B')$.

■

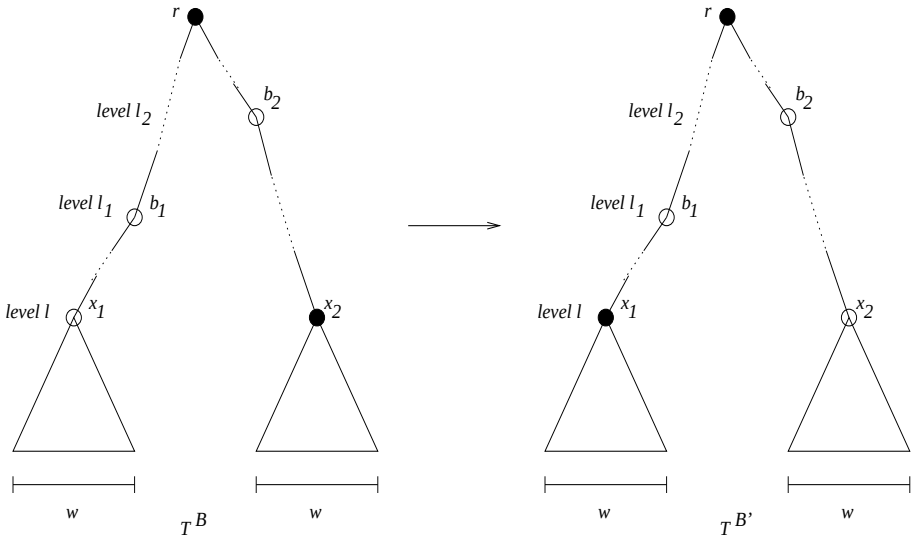

Figure 4.3: *Transformation adopt.* If $B' = B \setminus \{x_1\} \cup \{x_2\}$ then $\mathcal{G}(B') > \mathcal{G}(B)$.

Lemma 6. ***Transformation adopt.*** *Suppose that T^B contains two nodes x_1 and x_2 at the same level l, such that $x_1 \in B$ and no other node in the subtrees rooted at x_1 and x_2 belongs to B. Suppose that the last bookmark on the oriented path from r to x_1 in the tree T is the node $b_1 \in B$ on level $l_1 < l$ in T. Suppose as well, that if bookmark $b_2 \in B$ on level l_2 dominates x_2, we have $l_1 > l_2$. Then for $B' = B \setminus \{x_1\} \cup \{x_2\}$ we have $\mathcal{G}(B') > \mathcal{G}(B)$ (cf. Figure 4.3).*

Proof. Let w be the number of nodes in the subtrees rooted at x_1 or x_2, cf. Figure 4.3.

Let b_2 denote the bookmark dominating x_2 or $b_2 = r$ if such a bookmark does not exist, and let l_2 be the level of b_2 in G. Again, $\mathcal{G}_{-\{x_1,x_2\}}(B) = \mathcal{G}_{-\{x_1,x_2\}}(B')$. We have $g_B(v) = l - 1$, for each node v, descendant of x_1 and $g_B(v) = l_2 - 1$, for each v, descendant of x_2 (observe that, for this particular case, if $b_2 = r$ then we have to make $g_B(v) = 0$). Similarly, $g_{B'}(v) = l_1 - 1$, for each v, descendant of x_1 and $g_{B'}(v) = l - 1$, for each v, descendant of x_2. Hence

$\mathcal{G}(B) = \mathcal{G}_{-\{x_1,x_2\}}(B) + w(l-1) + w(l_2-1) < \mathcal{G}_{-\{x_1,x_2\}}(B') + w(l_1-1) + w(l-1) = \mathcal{G}(B')$.

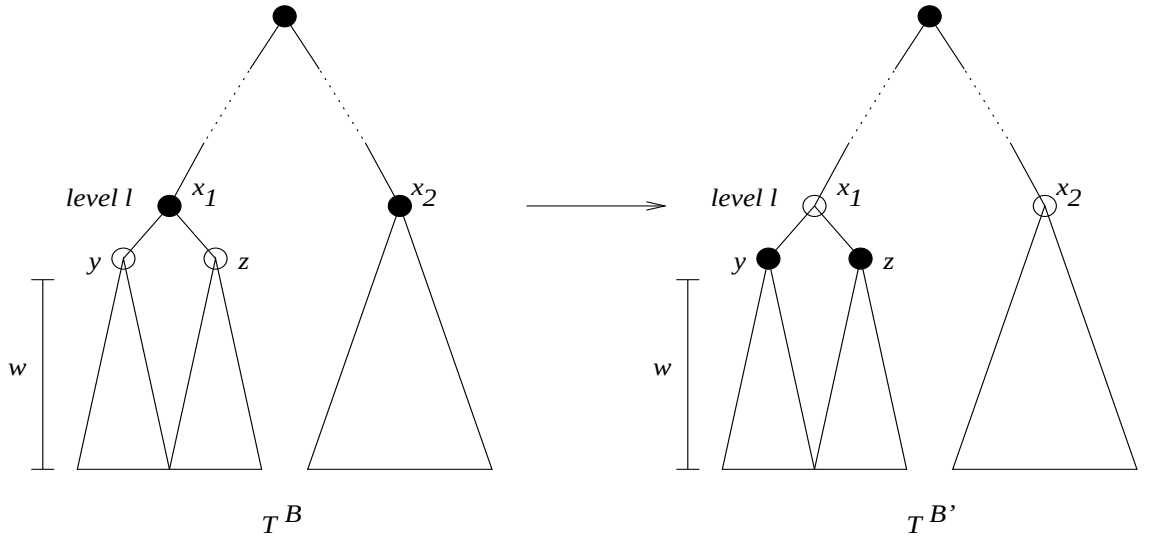

Figure 4.4: *Transformation spread.* If $B' = B \setminus \{y, z\} \cup \{x_1, x_2\}$ then $\mathcal{G}(B') > \mathcal{G}(B)$.

■

Lemma 7. ***Transformation spread.*** *Suppose that for a set of bookmarks B on T there exists a node $x_1 \notin B$ on level $l \geq 2$ in T, such that the only bookmarks contained in the subtree of T rooted at x_1 are its children, y and z. Suppose as well that there exists another node $x_2 \notin B$ on level l, such that the subtree rooted at x_2 contains no bookmarks. Then, for $B' = B \setminus \{y, z\} \cup \{x_1, x_2\}$ we have $\mathcal{G}(B') > \mathcal{G}(B)$ (cf. Figure 4.4).*

Proof. Let w be the number of nodes in the subtrees rooted at y or z, cf. Figure 4.4.

Again, $\mathcal{G}_{-\{x_1,x_2\}}(B) = \mathcal{G}_{-\{x_1,x_2\}}(B')$. For each node v, descendant of x_1 we have $g_B(v) = l$, while $g_B(x_1) = l - d_B(x_1)$. For all w nodes in the subtree rooted at x_2, $g_B(v) = l - d_B(x_2)$. Similarly, $g_{B'}(v) = l - 1$, for all $4w + 2$ nodes in the subtrees

rooted at x_1 and x_2. Since $d_B(x_i) \geq 2$ for $i = 1, 2$, we conclude that

$$\begin{aligned}\mathcal{G}(B) &= \mathcal{G}_{-\{x_1,x_2\}}(B) + 2lw + (l - d_B(x_1)) + (l - d_B(x_2))(2w + 1) \\ &< \mathcal{G}_{-\{x_1,x_2\}}(B') + (l-1)(4w+2) \\ &= \mathcal{G}(B')\end{aligned}$$

■

Lemma 8. ***Transformation floor.*** *If a set B of k bookmarks is optimal then each bookmark is placed on level $l = \lceil \log k \rceil$ of T or on a level above l.*

Proof. Let b be a bookmark at the deepest level l_b among all the bookmarks of B. We will prove by induction, that if $l_b > \lceil \log k \rceil$, then we can always replace b by some b', such that $\mathcal{G}(B) < \mathcal{G}(B \setminus \{b\} \cup b')$. There are three possible cases:

1. Neither the parent nor the sibling of b belongs to B. We can apply *transformation lift* (Lemma 5) replacing b by b' on a level above l in T^B.
2. The node a which is the parent of b belongs to B. As for the level l_a of a we have $l_a \geq \lceil \log k \rceil$. Hence there are at least $p = 2^{\lceil \log k \rceil}$ nodes on level l_a. As $p \geq k$, there exists a node u on level l_a, such that the subtree of T rooted at u contains no bookmarks. Thus the conditions of *transformation adopt* (Lemma 6) are met, with $b_1 = a$, $x_1 = b$ and $x_2 = u$. We can choose $b' = u$.
3. The parent of b does not belong to B, but its sibling c does. Again, there must exist a node u on level l_x such that the subtree of T rooted at u contains no bookmarks and the conditions of *transformation spread* (Lemma 7) are met with $y = b$, $z = c$ and $x_2 = u$. We take $b' = u$.

In each case, replacing b by b' improves the gain of the bookmark set. This concludes the proof.

■

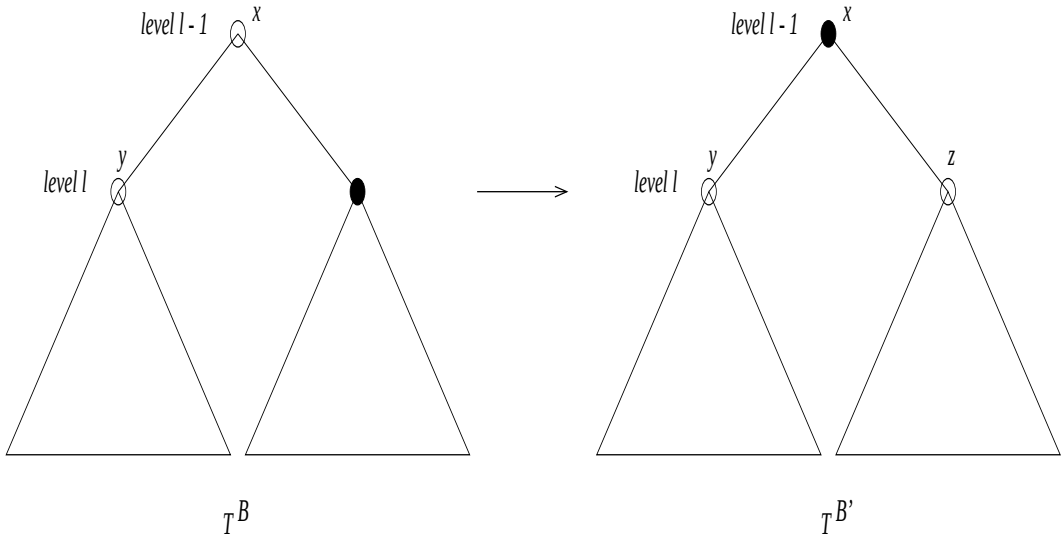

Figure 4.5: *Transformation inherit.* If $B' = B \setminus \{x\} \cup \{z\}$ then $\mathcal{G}(B') > \mathcal{G}(B)$.

Lemma 9. ***Transformation inherit.*** *Suppose that for a set B of k bookmarks, defined on the tree T of $N \geq 15$ nodes, there exists a pair of nodes $y, x \in B$ at levels l and $l-1$ respectively, such that $2 \leq l \leq \lceil \log k \rceil$. Suppose, as well, that y is the only bookmark among the descendants of x. If $k \leq \sqrt{N+1}$, then for $B' = B \setminus \{x\} \cup \{z\}$, where z is the sibling of y, we have $\mathcal{G}(B') > \mathcal{G}(B)$ (cf. Figure 4.5).*

Proof. Let w be the number of nodes in the subtree rooted at y, cf. Figure 4.5.

Again, $\mathcal{G}_{\{x\}}(B) = \mathcal{G}_{-\{x\}}(B')$. For each node v, descendant of y, we have $g_B(v) = l-1$, while for all $w+1$ remaining nodes in the subtree rooted at x we have $g_B(v) = l-2$. Similarly, $g_{B'}(v) = l-1$ for all $2w$ descendants of x, and $g_{B'}(x) = l-1-d_B(x)$. We have to prove that

$$\begin{aligned} \mathcal{G}(B) &= \mathcal{G}_{-\{x\}}(B) + (l-2)(w+1) + (l-1)w \\ &< \mathcal{G}_{-\{x\}}(B') + 2(l-1)w + (l-1-d_{B'}(x)) \\ &= \mathcal{G}(B'), \end{aligned}$$

which is equivalent to

$$w + 1 > d_B(x) \tag{4.1}$$

Note that $d_B(x) \leq l$. Observe also that, since the depth of T equals $\log(N+1) - 1$, we have $w = 2^{\log(N+1)-l} - 1$. Therefore, since $k \leq \sqrt{N+1}$, which for $N \geq 15$ implies $\log k \leq \frac{\sqrt{N+1}}{2}$, we have $w + 1 \geq 2^{\log(N+1)-l} \geq 2^{\log(N+1)-\lceil \log k \rceil} \geq 2^{\log(N+1)-\log k - 1} = \frac{N+1}{2k} > \frac{\sqrt{N+1}}{4} \geq \frac{\log k}{2} > \lceil \log k \rceil \geq l \geq d_B(x)$

■

Lemma 10. ***Transformation expose.*** *Suppose that for a set B of k bookmarks on T there exists $b \in B$ at level $t \geq 3$, such that the only bookmarks contained in the subtree of T rooted at b, are $\tilde{B} = \{b_1, \cdots, b_m\}$, and*

1. *$\{b_1, \cdots, b_p\}$, where $1 \leq p \leq m$ are placed on level l,*
2. *$\{b_{p+1}, \cdots, b_m\}$ are placed on level $l + 1$,*
3. *all bookmarks of $\tilde{B}$ are independent,*
4. *each leaf of T^B is in the domain of some bookmark from level l or $l + 1$.*

If $k \leq \sqrt{N+1}$, then for $B' = B \setminus \{b, b_i\} \cup \{c_i, d_i\}$, where $b_i \in \tilde{B}$ is on level l and c_i, d_i are children of b_i we have $\mathcal{G}(B') > \mathcal{G}(B)$ (cf. Figure 4.6).

Proof. As $\mathcal{G}_{-\{b\}}(B) = \mathcal{G}_{-\{b\}}(B')$, we need only to compute the gain function for the nodes in the subtree rooted at b.

Let n be the depth of T. Observe that there are $w = 2^{n-l} - 1$ nodes in the domain of each node $b_i, i = p+1, \ldots, m$, and for each of them $g_B(v) = l$. Similarly $g_B(v) = l - 1$, for each of $2w + 1$ descendants of every node $b_i, i = 1, \ldots, p$. Finally, for v being one of $2^{l-t+1} - p - 1$ nodes in the domain of b, we have $g_B(v) = t - 1$. In the graph $T^{B'}$, there are $p - 1$ bookmarks which are descendants of b on level l and $m - p + 2$ bookmarks on level $l + 1$. Similarly as in T^B, for each of $2w + 1$ nodes in the domain of some bookmark on level l, we have $g_B(v) = l - 1$ and for each of w nodes

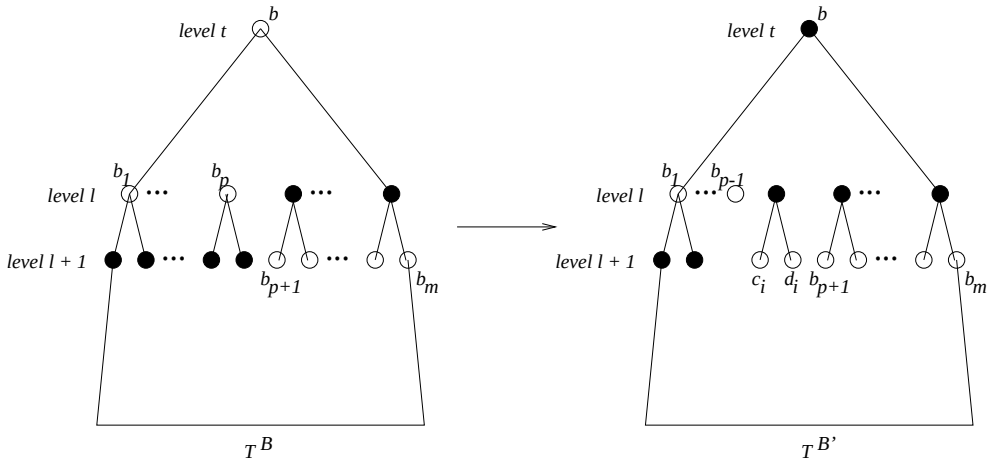

Figure 4.6: *Transformation expose.* "Exposing" a node from a higher level. If $B' = B \setminus \{b, b_p\} \cup \{c_i, d_i\}$ then $\mathcal{G}(B') > \mathcal{G}(B)$.

from the domain of a bookmark of level $l+1$, $g_B(v) = l$. The subtree of $T^{B'}$ rooted at b has $2^{l-t+1} - p - 2$ nodes, which are not in the domain of any of $b_i, i = 1, \ldots, m$. For every such node $g_B(v) = t - d_B(x)$. Thus we have to prove that

$$\begin{aligned}
\mathcal{G}(B) &= \mathcal{G}_{-\{b\}}(B) + (t-1)(2^{l-t+1} - p - 1) + p(2w+1)(l-1) + (m-p)wl \\
&< \mathcal{G}_{-\{b\}}(B') + (t - d_{B'}(b))(2^{l-t+1} - p) \\
&\quad +(p-1)(2w+1)(l-1) + (m-p+2)wl \\
&= \mathcal{G}(B')
\end{aligned}$$

or, equivalently, that $(t-1)(2^{l-t+1} - p - 1) + (2w+1)(l-1) < (t - d_{B'}(b))(2^{l-t+1} - p) + 2wl$. However, since $2 \leq d_{B'}(b) \leq t$ and $1 \leq p \leq m$, it is sufficient to prove a stronger condition substituting $d_{B'}(b) = t$ and $p = 1$. For these values, the above inequality becomes $(t-1)(2^{l-t+1} - 2) + (2w+1)(l-1) < 2wl$, which is equivalent to $(t-1)(2^{l-t+1} - 2) + l < 2w + 1$. As $t \geq 2$, it is sufficient to prove the stronger condition $(t-1)2^{l-t+1} + (l-2) < 2w + 1$. The function $f(t) = (t-1)2^{l-t+1} + l - 2$ attains its maximum when $f'(t) = 2^{l-t+1} - (t-1)2^{l-t+1} \ln 2 = 0$, i.e. when $t = \frac{1}{\ln 2} + 1$.

Since t is a level number and f is an integer argument unimodal function, it follows that that $f(t)$ obtains the same maximal value for $t = 2$ and $t = 3$. Hence, we can strengthen again the inequality we have to prove, getting $2^{l-1} + (l-1) < 2w + 1$. Since $w = 2^{n-l} - 1$ and $N = 2^{n+1} - 1$, the condition becomes

$$2^{l-1} + (l-1) < (N+1)2^{-l} \tag{4.2}$$

However, since $2^{l-1} \geq l - 1$, once again we can strengthen the condition obtaining $2^l < (N+1)2^{-l}$ or

$$2^{2l} < (N+1) \tag{4.3}$$

Observe that, since each leaf of T^B is in the domain of some bookmark from level l or $l+1$, each of 2^l nodes on level l, or one of its children, must belong to B. As $b \in B$, we identified at least $2^l + 1$ bookmarks belonging to B, proving that $k > 2^l$. Since $k \leq \sqrt{N+1}$, we conclude that $2^{2l} < k^2 \leq (N+1)$, thus proving Inequality 4.3.

■

4.2.1 Correctness of the Characterization

In this section we prove the correctness of the characterization given in Theorem 9.

Proof. Let B be an optimal set of bookmarks. By *transformation floor* (Lemma 8), we can suppose that each bookmark $b \in B$ is on or above the *floor* level $l = \lceil \log k \rceil$. Take a bookmark $b \in B$ on the deepest level l_b. Let c be the sibling of b and let a be its parent. Observe that at most one node among a and c can belong to B, otherwise we could improve B by means of *transformation expose* (Lemma 10). However, in such a case, $a \notin B$, since we could apply *transformation inherit* (Lemma 9), again improving the gain of B. If $c \notin B$, we could apply *transformation lift* (Lemma 5),

once again getting a better bookmark set. So $c \in B$ and $a \notin B$. Similarly, for any other bookmark on level l_b its sibling must belong to B and its parent must not. Take any node x on level $l_b - 1$, whose children do not belong to B. $x \in B$, otherwise we could apply *transformation spread* (Lemma 7) using $x_1 = a$ and $x_2 = x$. So for each node x on level l_b, either x itself or both its children belong to B. Consider the deepest level bookmark $y \in B$ above level $l_b - 1$. If y exists, y along with all the bookmarks of B which are descendants of y verify the conditions of *transformation expose* (Lemma 10). If y does not exist, the bookmarks of B are all independent, cover T and are placed on two consecutive levels $l_b - 1$ and l_b.

■

The following proposition shows that the property of optimal bookmark sets described in Theorem 9 does not hold for larger numbers of bookmarks.

Proposition 1. *Let T be a complete binary tree with N nodes and let $k = 2\sqrt{N+1} + 1$. There exists a set B of k bookmarks in T such that:*

1. *B covers T,*
2. *all bookmarks in B are independent,*
3. *all bookmarks in B are placed on two consecutive levels of T,*

but B is not optimal.

Proof. Consider a complete binary tree T of depth $n = 2l - 3$ and let B be a set of $k = 2^l + 1$ bookmarks. Note that $k = 2\sqrt{N+1} + 1$. According to Lemma 3 we can place 2^l bookmarks on level l and one bookmark on level $(l+1)$, so that the bookmarks are independent and cover T. To show that B is not optimal, let B' be a set of $2^l + 1$ bookmarks consisting of all nodes of level l and of one bookmark on level two, cf. Figure 4.7. We have

$$\mathcal{G}(B') - \mathcal{G}(B) \;=\; [(l-1)(2^l)(2^{n-l+1} - 1) + 1(2^{l-2} - 1)]$$

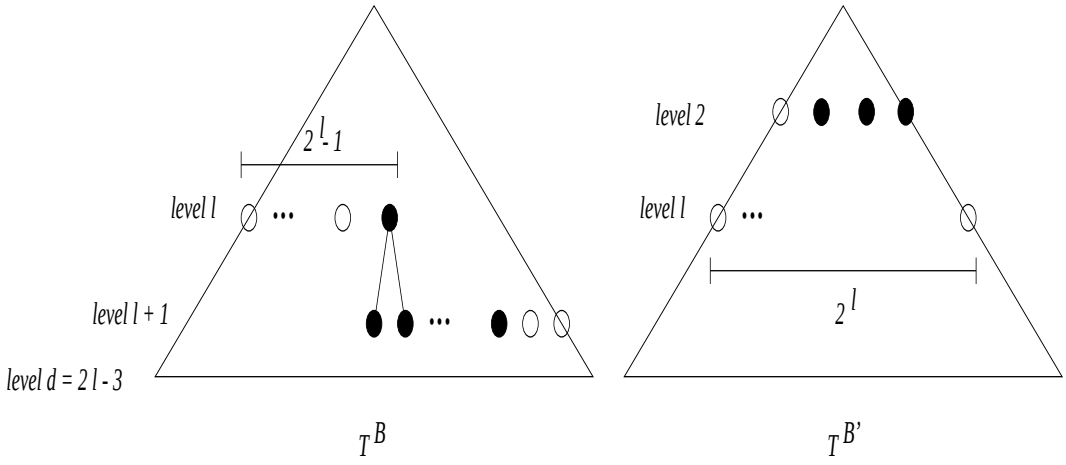

Figure 4.7: For Proposition 1. Suppose that B is optimal. Composing B' of all nodes at level l plus a node at level 2 will make $\mathcal{G}(B') > \mathcal{G}(B)$, proving that B is not optimal.

$$
\begin{aligned}
& -[(l-1)(2^l-1)(2^{n-l+1}-1)+2l(2^{n-l}-1)] \\
= \;& l+1>0
\end{aligned}
$$

thus proving that B is not optimal.

■

Part III

Optimizing the Expected Number of Steps

Chapter 5

Heuristic Algorithms for Hotlink Assignments

5.1 Introduction

In this chapter we present heuristic hotlink assignment algorithms. We give a lower bound, derived from Shannon's theory, on the access cost of a tree with at most k hotlinks per node. This work can be found in [16] as well.

5.2 Lower Bound

We generalize the lower bound discussed in Section 3.2.

Theorem 10. *For any probability distribution p on the leaves of a tree of maximum outdegree δ and any assignment of at most k hotlinks per source node, the expected number of steps from the root of the tree to reach a Web page located at a leaf is at least $\frac{\mathcal{H}(p)}{\log(\delta+k)}$, where $\mathcal{H}(p)$ is the entropy of p.*

Proof. A tree of maximum outdegree δ can be viewed as the encoding of the leaves

with the δ symbol alphabet $0, 1, ..., \delta - 1$. Adding k hotlinks per node increments the alphabet by k symbols to form a $\delta + k$ symbol alphabet.

Consider a tree $T = (V, E)$. For a given hotlink assignment A, the distance of the i-th leaf from the root in T^A, $d(i)$, is the length of the encoding of the i-th leaf in this new alphabet. Notice that if two hotlinks are targeting the same node, then the shortest one can be omitted without changing the value of $E[T^A]$. As a consequence, $E[T^A]$ is also the expected length of the "encoding" of the leaves of the tree T^A represented as code-words in a $\delta + k$ letter alphabet. Moreover, the resulting encoding is a prefix code. In particular, Shannon's theorem (Theorem 3) applies and we have that

$$E[T^A] \geq \frac{1}{\log(\delta + k)} \cdot \mathcal{H}(p) = \frac{1}{\log(\delta + k)} \cdot \sum_{i=1}^{m} p_i \log(1/p_i) \tag{5.1}$$

■

5.3 One Hotlink per Page

In this section we present our hotlink assignment algorithms that assign at most one hotlink per page. We also describe an algorithm by Kranakis et al. [35] called *hotlinkAssign*. This algorithm is important because it guarantees an upper bound on the access cost of a Web site according to the maximum outdegree of the site.

5.3.1 Algorithm *simpleBFS*

Algorithm *simpleBFS,* formally described in Algorithm 4, 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 4 simpleBFS(T)

1. $H = \phi$
2. *while(* $(\ s = next\_in\_BFS\_order\ ) \neq \phi$ *)*
   (a) $t = v$: *v maximizes Equation 2.4; 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.

5.3.2 Algorithm *greedyBFS*

Algorithm *greedyBFS,* described in Algorithm 5, 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[1]. See Figure 5.1. The algorithm stops when there are no more possible hotlinks to assign.

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 alwaysthe

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

Algorithm 5 greedyBFS(T)

1. $H = \phi$
2. *while((s = next_in_BFS_order) $\neq \phi$)*
   (a) $t = v : v$ *maximizes Equation 2.4; 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* $H = H \bigcup \{(s, t)\}$

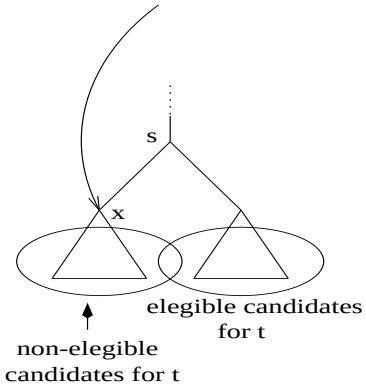

Figure 5.1: In algorithm *greedyBFS*, two hotlinks are not allowed to cross each other. Consider a hotlink (s, t) to be assigned in an iteration of the algorithm. t must be a descendant of s that minimizes the cost but is not a descendant of a node x, which is at a higher level than s and already has an incoming hotlink.

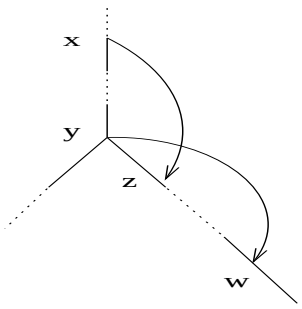

Figure 5.2: Illustration of *"obvious navigation"* assumption. Suppose that a user is at page x and wishes to reach a page descendant of w. We assume that, as the user does not have a map of the site, the path $(x, z) + z \to w$ will be followed, even though path $x \to y + (y, w)$ would have been shorter. Therefore, obvious navigation is efficient only when $x \to y \geq z \to w$.

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 5.2.

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 5.3.

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

5.3.3 Algorithm *recursive*

The basic principle of algorithm *recursive,* described in Algorithm 6, is to assign a hotlink (s, t), where s is the home page and t is the node that maximizes Equation 2.4,

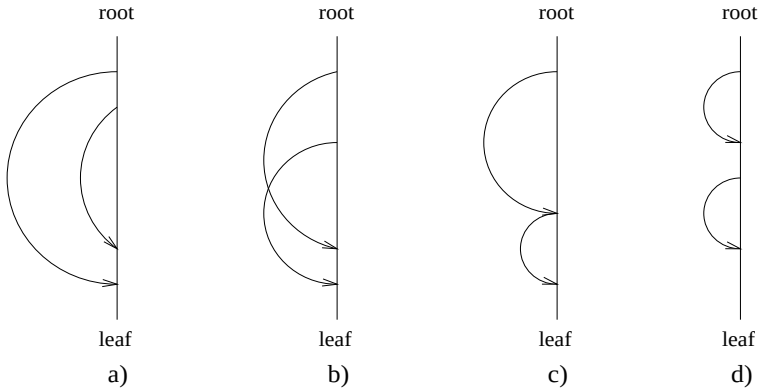

Figure 5.3: Four possible scenarios of two hotlinks on a path, after executing algorithm *simpleBFS*. Each figure shows an arbitrary path from the root to a leaf. a) one hotlink is "inside" the other, b) the hotlinks are overlapped, c) one hotlink starts exactly at the end of the other, d) otherwise.

then split the original tree into subtrees, i.e., the subtrees rooted at the children of s and the subtree rooted at t, and then proceed recursively for each of these subtrees. Note that one of the children of s, s_k, will be ancestor of t, and the subtree rooted at t will not form part of the subtree rooted at s_k. Figure 5.4 illustrates the algorithm. Assume that the set of hotlinks, H, is initially empty.

5.3.4 Algorithm *hotlinkAssign*

Algorithm *hotlinkAssign* was designed by Kranakis et al [35]. They prove the following upper bound on the access cost of a Web site that 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 *hotlinkAssign* is a recursive algorithm that works as follows. Let T_c

Algorithm 6 recursive(T)

1. $s = r$
2. *for* $i = 1$ *to* k
   (a) $t = v : v$ *maximizes Equation 2.4; and* v *is not a descendant of a node* x, *which is at a higher level than* s *and already has a hyperparent; and* v *has no hyperparent*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s, t)\}$
3. *for each children of* s, s_i *(including hypersons)*
   (a) $T =$ *subtree rooted at* s_i
   (b) recursive(T)

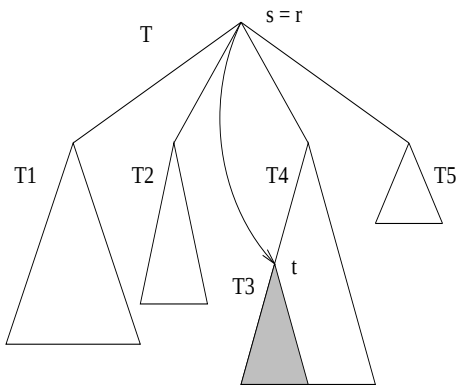

Figure 5.4: The first call to *recursive*(T). The algorithm assigns a hotlink from the home page s to the node t that minimizes the access cost and then proceeds recursively for T_i, $i = 1, ..., 5$. The recursive calls continue until it is not possible to assign more hotlinks. Note that T_3 is not part of T_4 and thus are treated as separate trees.

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:

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 *hotlinkAssign* is formally described in Algorithm 7. The set of hotlinks, H, is initially empty.

Algorithm 7 hotlinkAssign(T_c)

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

5.4 Multiple Hotlinks per Page

The algorithms described so far are restrained to assigning at most one outgoing hotlink per page. In this section we present an alternative version of algorithms *greedyBFS* and *recursive*, where the maximum number of outgoing hotlinks allowed per node is k. We also present a new greedy algorithm called *greedyBFS*^{k}.

5.4.1 Algorithm *k-greedyBFS*

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. Algorithm *k-greedyBFS* is described in Algorithm 8. Observe that the call *k-greedyBFS*($T, 1$) is equivalent to a call to *greedyBFS*(T).

Algorithm 8 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 Equation 2.4; 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* $H = H \bigcup \{(s, t)\}$

5.4.2 Algorithm *k-recursive*

Algorithm 9 describes our k-hotlink version of algorithm *recursive*.

Algorithm 9 k-recursive(T, k)

1. $s = r$
2. *for* $i = 1$ *to* k
   (a) $t = v : v$ *maximizes Equation 2.4; and* v *is not a descendant of a node* x*, which is at a higher level than* s *and already has a hyperparent; and* v *has no hyperparent*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s, t)\}$
3. *for each children of* s*,* s_i *(including hypersons)*
   (a) T = *subtree rooted at* s_i
   (b) k-recursive(T, k)

5.4.3 Algorithm *greedyBFS*^{k}

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[2], treating the hotlinks as regular hyperlinks. The algorithm is formally described in Algorithm 10. Figure 5.5 illustrates the operation of the algorithm.

Algorithm 10 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.

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

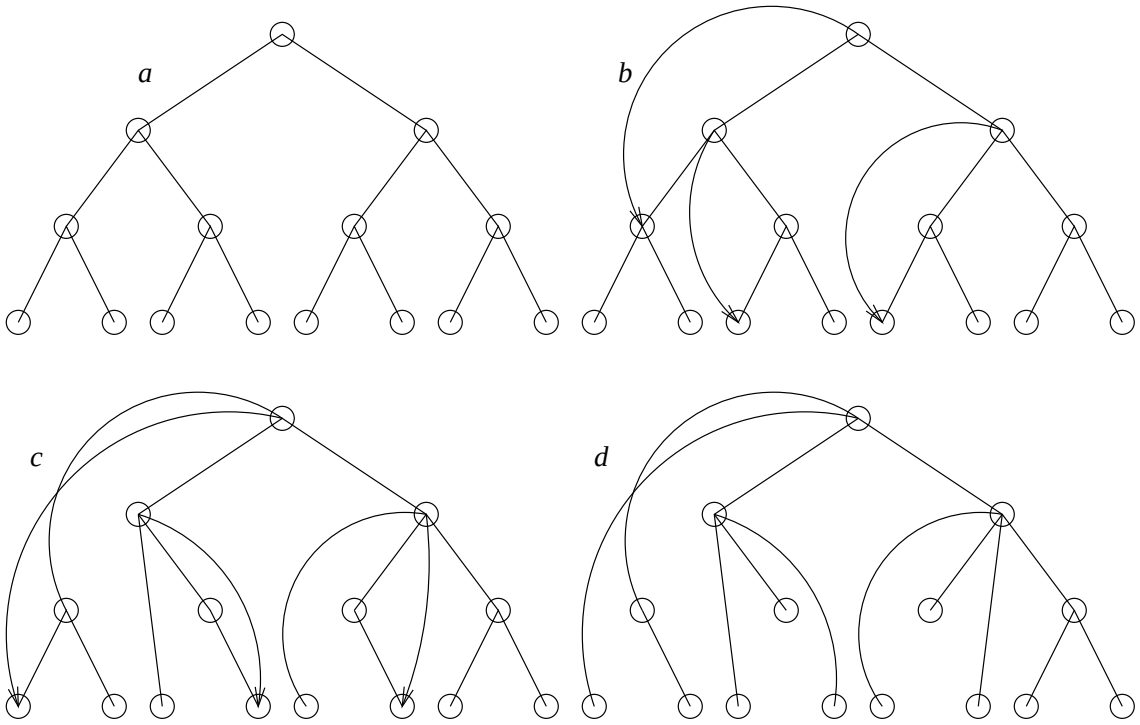

Figure 5.5: Illustration of the operation of algorithm *greedyBFS*^{k}. Observe that after assigning a set of hotlinks, those hotlinks are treated as regular hyperlinks in subsequent iterations of the algorithm.

Chapter 6

Simulations and Case Study

6.1 Introduction

In Chapter 5 we present heuristic hotlink assignment algorithms. In this chapter we evaluate those algorithms and present the results obtained. Some of these results can also be found in [16]. All the simulations are implemented with *Java.* We evaluate our 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.

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. We validate the correctness of the simulations by showing that the outdegree δ and the popularity p of the random Web sites actually

follow the theoretical probability distributions.

6.2 Web Sites Used for Simulation

In this section we discuss some theory of Web site structure and discuss the difficulty of generating accurate random Web sites.

6.2.1 Random Web Sites

Faloutsos et al. [21] point out that Internet topology can be modeled 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. [11] 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

$$P[\delta_v = i] \sim i^{-2.72} \tag{6.1}$$

6.2.2 Generation of Random Web Sites

There are many techniques for generating random graphs. Waxman [54] proposes a random graph generator for modeling geographical networks. Calvert et al. [12] discusses how graph-based models can be used to generate large graphs with specific parameters of locality and hierarchy. Zegura et al. [56, 57] 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. [3] 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 [53] 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., [2, 50]. Hayes [27, 28] analyses gigantic graphs such as the Web, and maintains that the Web is a small world graph. Hayes points out how lattices andErdös-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 the following power-law formula, derived from Formula 6.1:

$$\delta_i = s * i^{-2.72}$$

The generation of a random Web site tree structure is described in Algorithm 11. In Section 6.3.1, we discuss the assignment of access probabilities to the random Web site trees.

6.2.3 Actual Web Sites

In Section 6.2.2, we briefly survey the efforts to generate random graphs with particular characteristics [54, 12, 56, 57, 3, 27, 28]. 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. [29] suggest performing random walks on the Internet

Algorithm 11 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 6.2.2)
   (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* $T = (V, E)$

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[1] 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 6.3.1 we study the assignment of access probabilities to these Web sites.

6.3 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.

6.3.1 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 [58], one could conjecture that the

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

popularity of Web pages obeys Zipf's popularity law. Glassman [24] prove experimentally that the popularity of Web pages can actually be modeled with Zipf's popularity law. Glassman analyses 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. [36] show experimentally that Zipf's popularity law explains user's navigation, in the sense that longer navigations are less probable than shorter ones. Pitkow [46] 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, $H_m = \sum_{j=1}^{m} \frac{1}{j}$ [33]. Thus, given the *i-th* most popular Web page v of a Web site of m pages,

$$p_v = \frac{1}{iH_m} \tag{6.2}$$

In order to associate popularities to the m leaves of a Web site T, we generate a popularity ranking-list of m elements based on Equation 6.2, 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 12.

6.4 Results of the Simulations

We evaluate the algorithms presented in Chapter 5 on random and real Web sites. The maximum gain achieved with at most one hotlink per page is 35%, whereas the minimum is 24%. We also compare the performance of algorithms *greedyBFS* and *hotlinkAssign* with the theoretical optimal solution.

Algorithm 12 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 6.3.1)
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

6.4.1 One Hotlink per Page

Algorithm *greedyBFS* Evaluated on Random Web Sites

Algorithm *greedyBFS* is described in Section 5.3.2. We show the results of the simulations in Figure 6.1 and Table 6.1. 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.

Algorithm *greedyBFS* Evaluated on Real Web Sites

Algorithm *greedyBFS* is discussed in Section 5.3.2. The results of the simulations are displayed in Table 6.2 and depicted in Figure 6.2. 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.

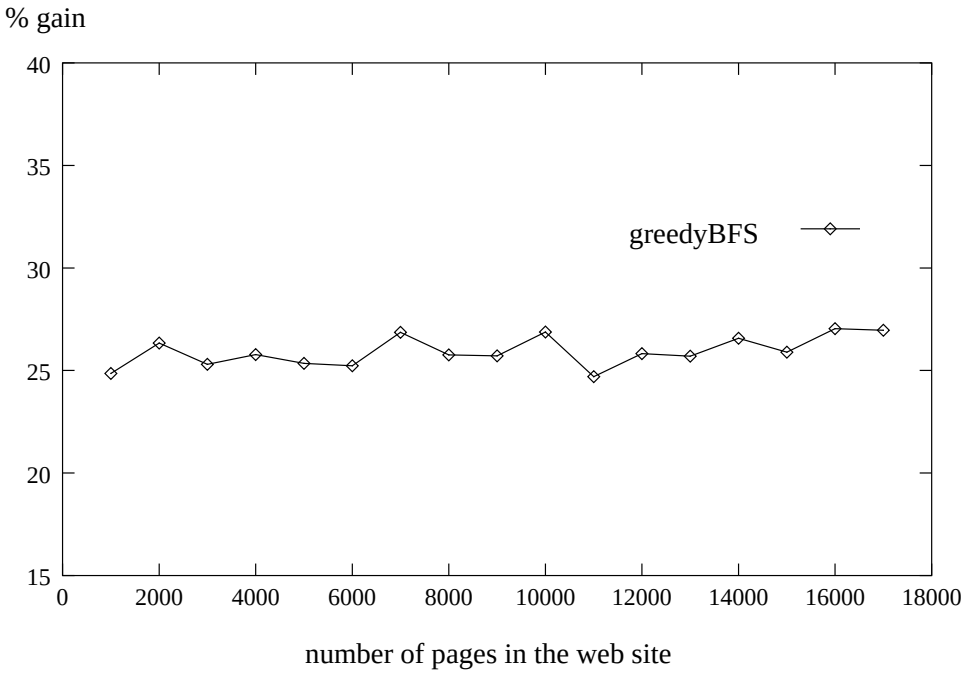

Figure 6.1: Gain obtained by applying *greedyBFS* to randomly generated Web sites of size $1,000$ to $17,000$. The average proportion of gain has a 95% confidence interval of at most ∓ 1.27.

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

Table 6.1: 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.

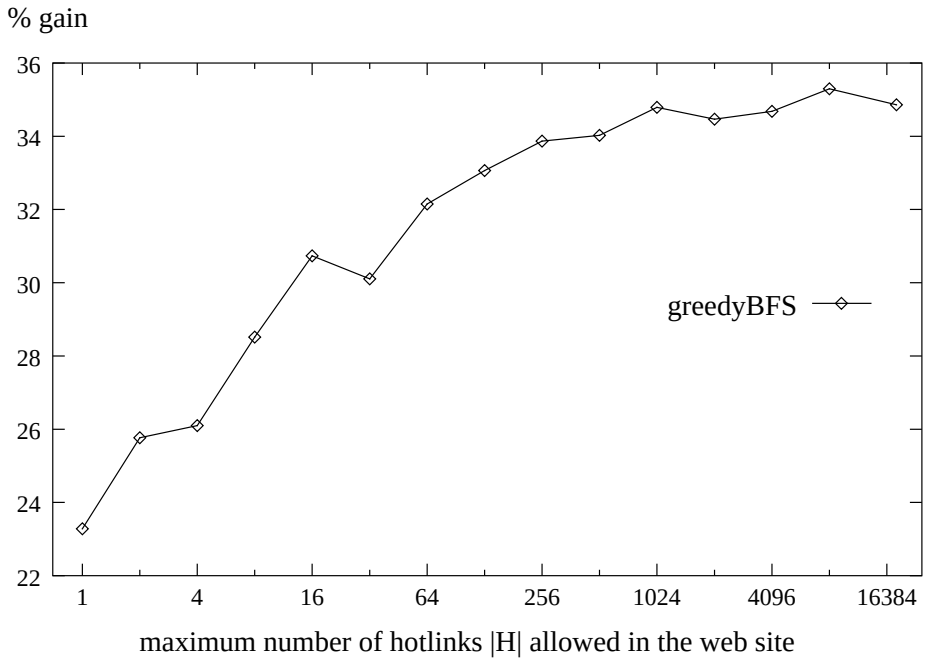

Figure 6.2: Average gain obtained by applying *greedyBFS* to actual Web sites. The x axis are plotted in logarithmic scale base 2. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 5.53. The proportion of gain decreases at some points due to the different assignments of Zipf's distribution in each sample.

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
1024	34.79	5.81
2048	34.46	6.30
4096	34.68	6.64
8192	35.30	6.11
16384	34.86	6.25

Table 6.2: 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.

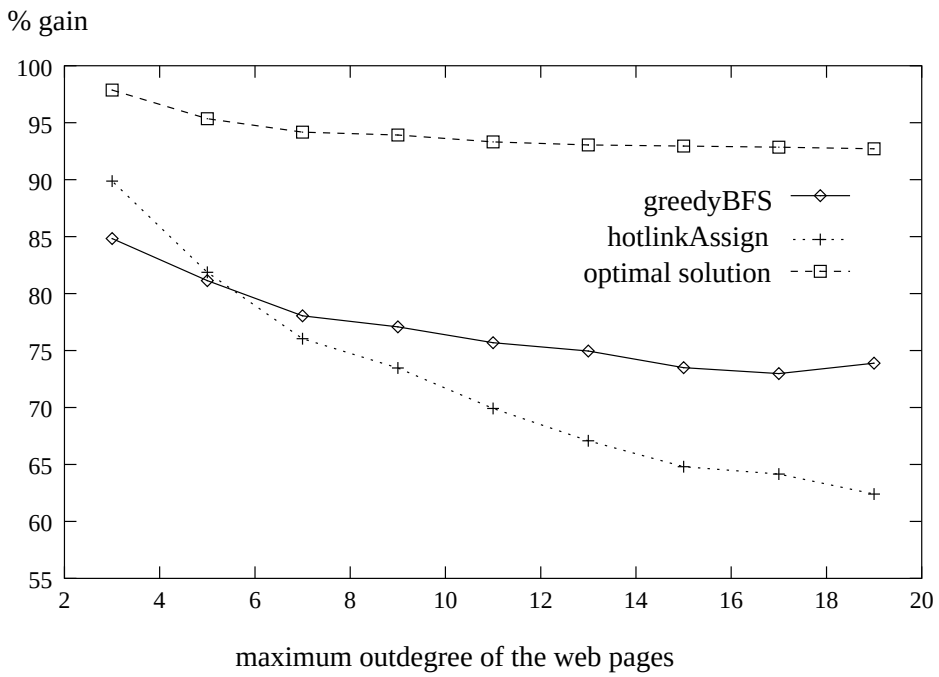

Figure 6.3: Comparison of *greedyBFS* with *hotlinkAssign* and the theoretical optimal solution (from Theorem 10) when applied to random Web sites of size $10,000$ and maximum outdegree from 3 to 19. The average proportion of gain for *greedyBFS, hotlinkAssign* and the optimal solution have a 95% confidence interval of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29 respectively.

Algorithms *greedyBFS* and *hotlinkAssign* and the Optimal Solution Compared on Random Web Sites

We present the results of comparing the performance of algorithms *greedyBFS,* described in Section 5.3.2, and *hotlinkAssign,* described in Section 5.3.4, with the theoretical optimal solution given in Theorem 10. The performance is compared on randomly generated Web sites of size $10,000$ with maximum outdegree 3 to 19. Figure 6.3 and Table 6.3 illustrate the results of the simulations. The average proportion of gain for *greedyBFS, hotlinkAssign* and the optimal solution have 95% confidence intervals of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29, respectively.

maximum outdegree, δ	% of gain with *greedyBFS*	% of gain with *hotlinkAssign*	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

Table 6.3: Comparison of the performance of *greedyBFS* with *hotlinkAssign* and the theoretical optimal solution (from Theorem 10) when applied to random Web sites of size $10,000$ and maximum outdegree from 3 to 19. The average proportion of gain for *greedyBFS, hotlinkAssign* and the optimal solution have a 95% confidence interval of at most $\mp 1.49, \mp 1.68$, and ∓ 0.29 respectively.

6.4.2 Multiple Hotlinks per Page

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

Algorithms *k-greedyBFS* and *greedyBFS*^{k} are described in Sections 5.4.1 and 5.4.3. The results of the simulations are depicted in Figure 6.4 and Table 6.4. Observe that for values of $k > 5$, *greedyBFS*^{k} outperforms *k-greedyBFS*.

6.5 Validation of the Simulations

To validate the correctness of the simulations, we use the model validation techniques presented by Jain [30]. According to Jain, the three key aspects to validate the model are:

1. Assumptions.

2. Input parameter values and distributions.

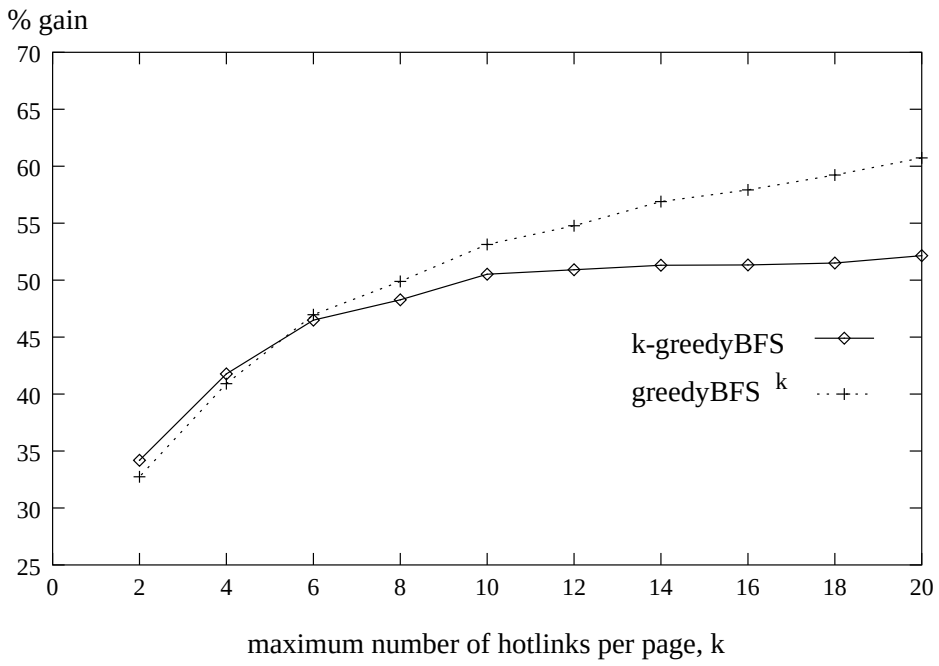

Figure 6.4: Performance of *k-greedyBFS* and *greedyBFS*^{k}. Observe how *greedyBFS*^{k} outperforms *k-greedyBFS* when $k > 5$. The proportion of gains for *k-greedyBFS* and *greedyBFS*^{k} have a 95% confidence interval of at most ∓ 0.91 and ∓ 0.99 respectively.

max. 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

Table 6.4: 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.

3. Output values and conclusions.

In our particular case, we tested the following:

1. The correctness of the implementation of the algorithms.
2. The generation of Web sites with realistic popularity on the Web pages and realistic hyperlink structure.
3. The consistency of the results as we vary the number of hotlinks allowed in the Web site, the number of hotlinks allowed per Web page, and the maximum outdegree of the Web pages.

Correctness of the Algorithms

In order to verify the correctness of the implementation of the algorithms, we check that the outcomes of algorithms *greedyBFS* and *recursive* are exactly the same, since the latter is a recursive version of the former. With this test, we can be confident that the implementation of *greedyBFS* is correct since it is unlikely that different (logical or syntactical) errors in the implementation of both *greedyBFS* and *recursive* 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 Equation 6.1. Figure 6.5 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.

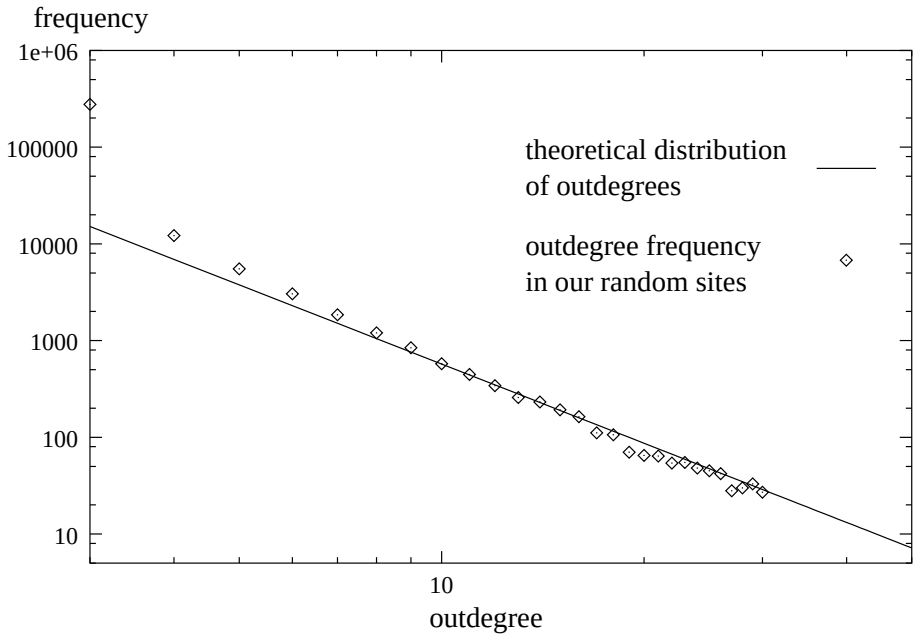

Figure 6.5: The correctness of the structure of the random Web sites is proven by comparing their outdegree frequencies with a power-law distribution (given by Equation 6.1) in a loglog scale.

6.6 Case Study

6.6.1 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.

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 7.3.1.

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 7.3.1.

Figure 6.6 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.

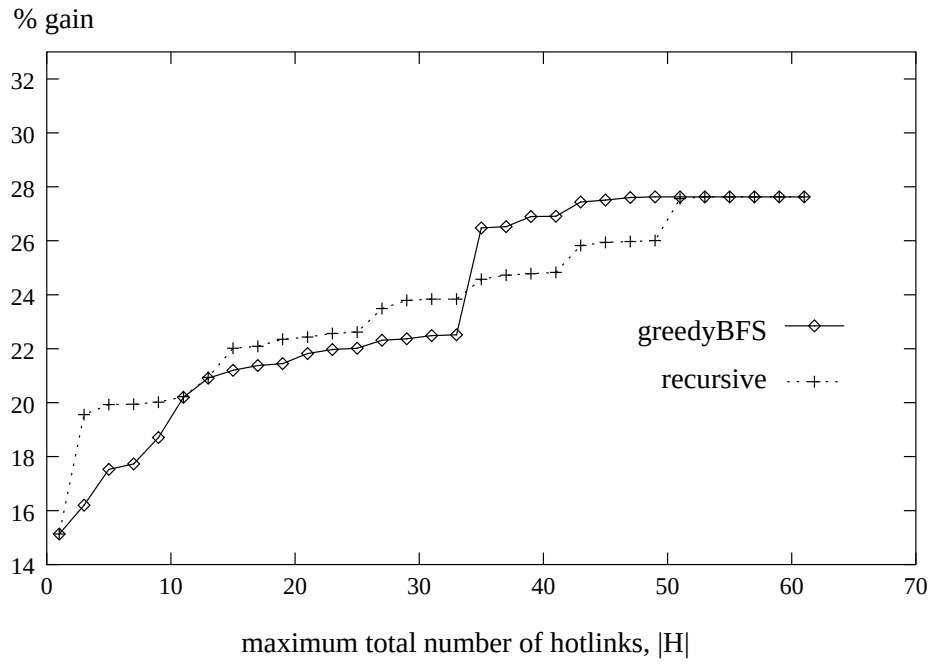

Figure 6.6: Gain attained by *greedyBFS*, and *recursive* in the *scs.carleton.ca* domain. For $|H| < 53$ *greedyBFS* and *recursive* differ in performance. Observe that this behaviour is due to the different order on which each algorithm assigns the hotlinks; however, in the end they will always converge, as the latter is the recursive version of the former. Note that again the maximum gain is achieved with few hotlinks, as $53 \ll |V| = 798$.

Chapter 7

The Hotlink Optimizer

7.1 Introduction

In Chapter 6 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 chapter 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 also presented in [34].

By analysing 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*[1] and the *access log files*, and the output is the set

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

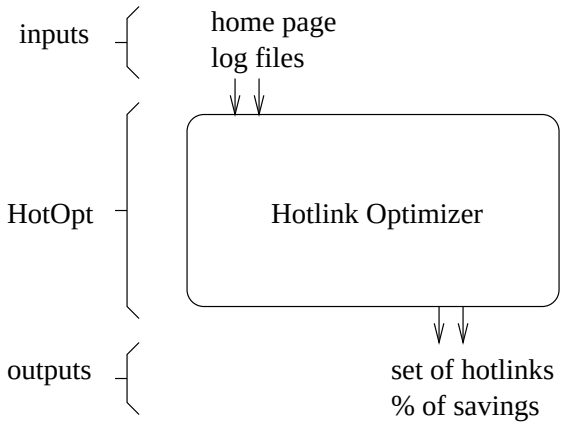

Figure 7.1: Inputs and outputs of HotOpt.

of hotlinks H along with x, the proportion of gain offered by H. Figure 7.1 depicts a macroscopic view of HotOpt.

7.2 The User Interface

HotOpt has a user interface that displays the tree-shaped hyperlink structure of the Web site (cf. Figure 7.2). The user has to provide the location of the home page and the access log files.

7.2.1 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.

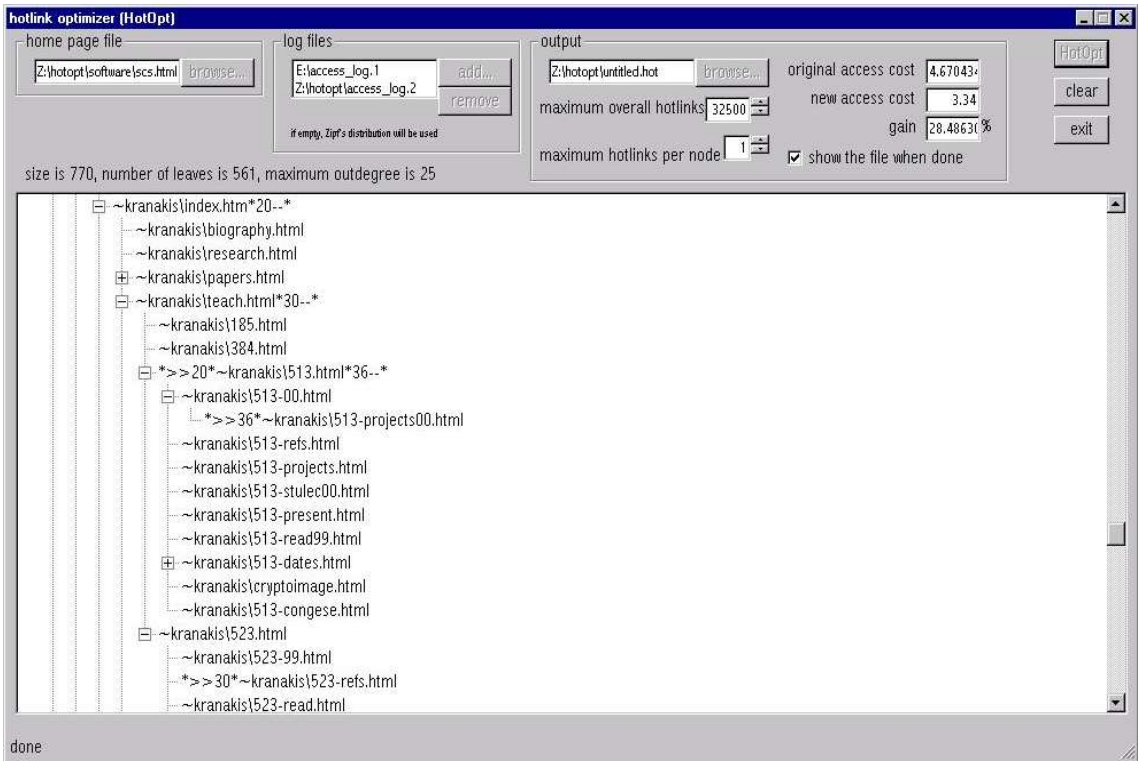

Figure 7.2: User interface of HotOpt.

7.2.2 Input

The user needs to provide the home page and the log files (top left corner of Figure 7.2). If no log files are available, HotOpt will use an arbitrary Zipf's popularity distribution. Refer to Section 6.3.1 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 7.2). The optimization process starts by pressing the "HotOpt" button located in the top right corner of the interface. See Figure 7.2.

Figure 7.3: Output of HotOpt.

7.2.3 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 7.2. The hotlinks are easily identified by assigning a unique number to each of them. For example, in Figure 7.2, 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 7.2). The set of hotlinks are saved in the file specified by the user (cf. Figure 7.3).

7.3 Architecture

The tasks performed by HotOpt can be summarized as follows (cf. Figure 7.4). 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.

7.3.1 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 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 Chapter 6, 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.

We describe these four processes in detail.

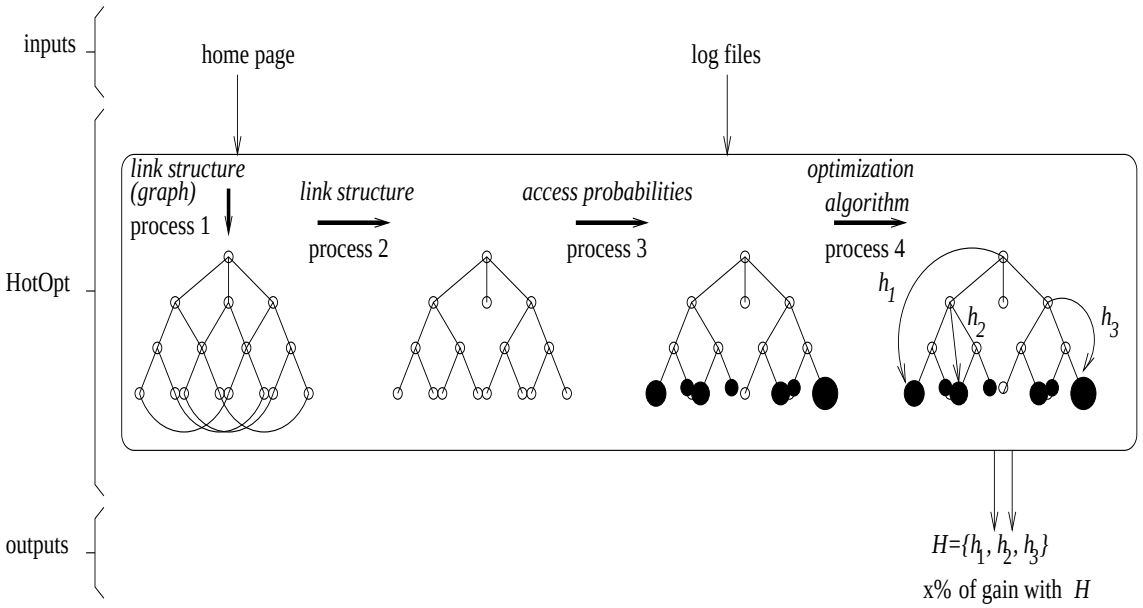

Figure 7.4: HotOpt performs four basic processes. Process *link structure (graph)* constructs a directed graph, where the nodes are Web pages and the edges are the hyperlinks that connect the pages. Process *link structure* builds a Web tree in breadth first search order with the home page as the root. 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.

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)*. In Chapter 2 we prove that the hotlink assignment problem is NP-hard. 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 13. 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.

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 (c.f. Figure 7.5). 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

Algorithm 13 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 *has not been visited*
      i. *enqueue* v *in* q
      ii. *mark* v *as visited*
      iii. $E' = E' \cup \{(u, v)\}$
7. *return*($T = (V, E'), r$)

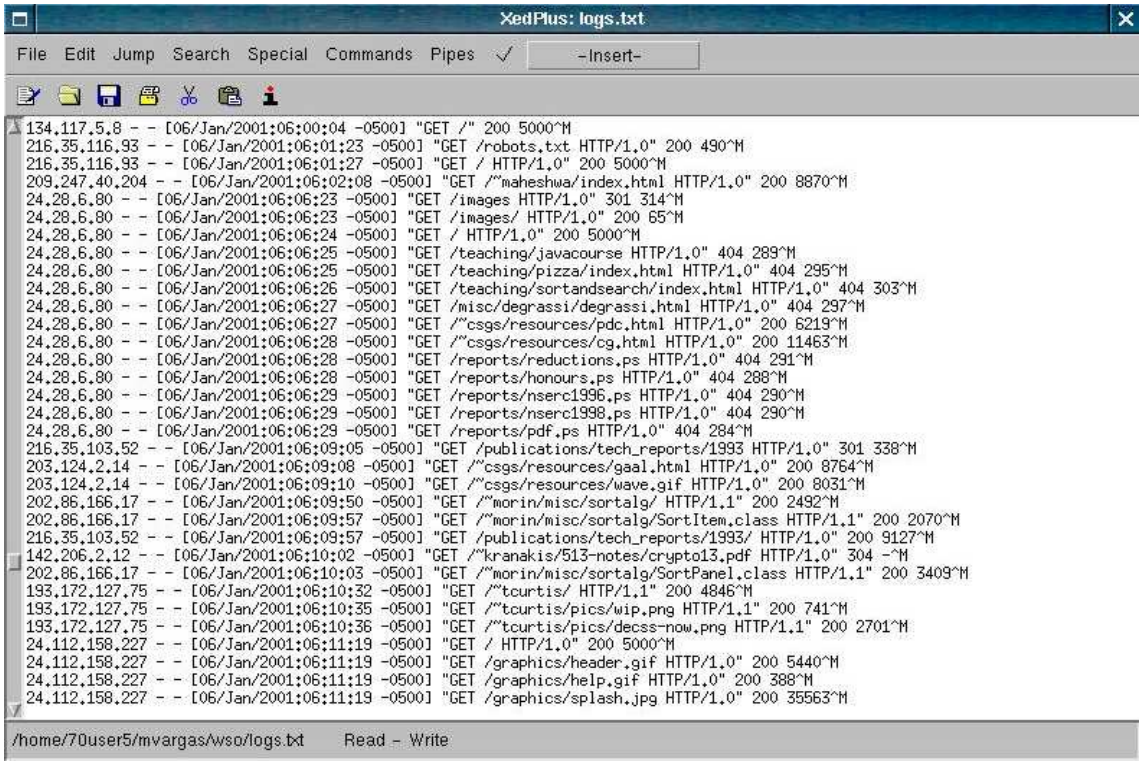

Figure 7.5: Example of an access log file of a Web site. The fields are as follows: requester's *IP address*, *time* of request, *file* requested, *protocol* used, *http code*, and *size* of the file in bytes.

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 [26] for a complete description of the http 1.1 protocol and [25] for a complete description of log files. Algorithm *extractHits,* given in Algorithm 14, 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.

Once algorithm *extractHits,* given in Algorithm 14, has been executed, we compute

Algorithm 14 extractHits(T, *logFile*)

1. *for each entry of* logFile

   (a) *if* code= 200 *or* code= 304 *and* file *requested is a leaf* v *of* T

      i. *increment by* 1 *the number of hits to leaf* v

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 in Chapter 6, algorithm *greedyBFS* performed best. This algorithm is described in Section 5.3.2. Algorithm *greedyBFS* is the optimization algorithm implemented in HotOpt.

7.3.2 Modular Structure

Figure 7.6 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 7.4. The *tree constructor* module outputs a Web site tree, $T = (V, E)$ with a probability distribution p over its leaves.

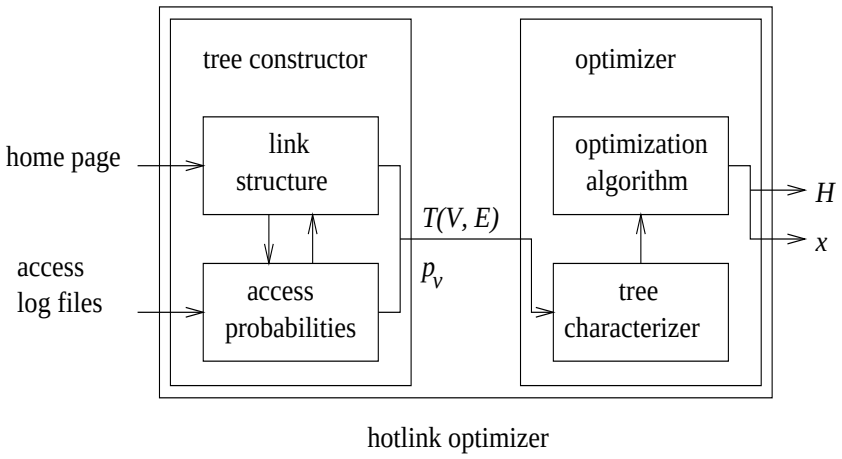

Figure 7.6: Main structure of HotOpt.

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 Chapter 6 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.

Part IV

Optimizing the Expected Data Transfer

Chapter 8

Heuristic Algorithm for Hotlink Assignments

8.1 Introduction

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 chapters 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

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

8.2 Notation and Terminology

A good way of measuring the access cost of a Web site is to consider the average data transfer generated for reaching a Web page. Under this perspective, we define the *weight* (in bytes) of a page v, w_v, as its own size plus the size of its embedded files. The *access weight of a page* v, $w(v)$, is equal to the sum of the weights of the pages contained in the shortest path between the home page and v. Consider a tree T with maximal outdegree δ and m leaves numbered $1, 2, \ldots, m$. Every node u of the tree has a weight w_u associated to it. To reach the Web page at u from the home page we must download all the Web pages in the path from the home page to u. Hence, if $\Pi(r, u)$ is the shortest path from the home page r to u then we have that the *access weight,* w_u, at u is given by the formula

$$w(u) = \sum_{v \in \Pi(r,u)} w_v \tag{8.1}$$

Without loss of generality, we may normalize the costs w_u.

At the same time, we have the access probabilities p_u at the nodes. This access probability is the frequency with which the node u may be visited from the home page. We have that

$$p_u = \sum_{v \,\text{child of}\, u} p_v \tag{8.2}$$

The *access cost* of page v is $c(v) = w(v) \cdot p_v$.

The *expected data transfer* for accessing a random leaf page from the root of a tree T with a probability distribution p on its leaves is given by the following equation.

$$E_w[T,p] = \sum_{u \text{ is a leaf}} w(u) \cdot p_u \tag{8.3}$$

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

$$w^A(u) = \sum_{v \in \Pi^A(r,u)} w_v \tag{8.4}$$

$$E_w[T^A,p] = \sum_{i=1}^{m} w^A(i) \cdot p_i \tag{8.5}$$

$$\mathcal{G}(A) = E_w[T,p] - E_w[T^A,p] = \sum_{i=1}^{m} (w(i) - w^A(i)) \cdot p_i \tag{8.6}$$

8.3 Lower Bound

We want to find a lower bound on the right-hand side of Equation 8.3. We will prove the following theorem.

Theorem 11. *The expected data transfer necessary to access a random Web page from the root of the tree T satisfies*

$$E_w[T,p] \geq \frac{1}{\log \delta} \cdot \sum_{i=1}^{m} \frac{w(i)}{d(i)} \cdot p_i \cdot \log\left(\frac{1}{p_i}\right)$$

where $w(i)$ is the weight at i-th *leaf as defined by Equation8.1, p_i is the access probability at the* i-th *leaf, δ is the maximum degree of the tree, and $d(i)$ is the distance of the* i-th *leaf from the root.*

Proof. Define:

$$\begin{aligned} E'_w[T,p] &:= \sum_{i=1}^{m} w(i)\cdot d(i)\cdot p_i \\ \mathcal{H}_w(p) &:= \sum_{i=1}^{m} w(i)\cdot p_i \cdot \log\left(\frac{1}{p_i}\right) \end{aligned}$$

Let the tree T have maximum degree δ and m leaves. With the normalization of the costs w_u, we can assume:

- $0 < w_i \leq 1$
- $w_1 + w_2 + \cdots + w_m \leq 1$

The leaves of the tree T represent an encoding with base alphabet of size δ, i.e., the maximum degree of the tree. By McMillan's theorem [1]:

$$\sum_{i=1}^{m} \frac{1}{\delta^{d(i)}} \leq 1 \tag{8.7}$$

It follows from Inequality 8.7 that

$$\sum_{i=1}^{m} w(i)\cdot p_i \cdot \log\left(\frac{1}{p_i}\right) \leq \sum_{i=1}^{m} w(i)\cdot p_i \cdot \log(\delta^{d(i)}) \tag{8.8}$$

To prove Inequality 8.8 we argue as follows. Define $r_i := \delta^{d(i)}$. Then we have:

$$\begin{aligned} \sum_{i=1}^{m} w(i)\cdot p_i \cdot \log\left(\frac{r_i}{p_i}\right) &= \frac{1}{\ln 2}\sum_{i=1}^{m} w(i)\cdot p_i \cdot \ln\left(\frac{r_i}{p_i}\right) \\ &\leq \frac{1}{\ln 2}\sum_{i=1}^{m} w(i)\cdot p_i \left(\frac{r_i}{p_i} - 1\right) \\ &= \frac{1}{\ln 2}\sum_{i=1}^{m} w(i)(r_i - p_i) \text{ note } (w(i) \leq 1) \end{aligned}$$

$$\begin{aligned} &\leq \frac{1}{\ln 2}\left(\sum_{i=1}^{m} r_i - 1\right) \\ &\leq 0 \end{aligned}$$

Therefore Inequality 8.8 is proven. Now we have the following:

$$\begin{aligned} \sum_{i=1}^{m} w(i)\cdot p_i \cdot \log\left(\frac{1}{p_i}\right) &\leq \sum_{i=1}^{m} w(i)\cdot p_i \cdot \log(\delta^{d(i)}) \\ &= \sum_{i=1}^{m} w(i)\cdot p_i \cdot d(i)\log(\delta) \\ &= \log\delta \cdot \sum_{i=1}^{m} w(i)\cdot d(i)\cdot p_i \\ &= \log\delta \cdot E'_w[T,p] \end{aligned}$$

Hence, we have proven the inequality

$$\frac{\mathcal{H}_w(p)}{\log\delta} \leq E'_w[T,p] \tag{8.9}$$

Now we turn to proving a lower bound on $E_w[T]$. Note that

$$\begin{aligned} E_w[T,p] &= \sum_{i=1}^{m} w(i)\cdot p_i \\ &= \sum_{i=1}^{m} d(i)\cdot\left(\frac{w(i)}{d(i)}\right)\cdot p_i \end{aligned}$$

Next, normalize the weights $w(i)$ by defining the new weights

$$w'(i) = \frac{\frac{w(i)}{d(i)}}{\sum_{j=1}^{m}\frac{w(j)}{d(j)}}.$$

Observe that $w'(i) \leq 1$ and

$$\begin{aligned} E_w[T,p] &= \sum_{i=1}^{m} d(i) \cdot \left(\frac{w(i)}{d(i)}\right) \cdot p_i \\ &= \left(\sum_{j=1}^{m} \frac{w(j)}{d(j)}\right) \cdot \sum_{i=1}^{m} d(i) \cdot \left(\frac{w(i)/d(i)}{\sum_{j=1}^{m} w(j)/d(j)}\right) \cdot p_i \\ &= \left(\sum_{j=1}^{m} \frac{w(j)}{d(j)}\right) \cdot E'_{w'}[T,p] \end{aligned}$$

By Inequality 8.9:

$$\begin{aligned} E_w[T,p] &\geq \left(\sum_{j=1}^{m} \frac{w(j)}{d(j)}\right) \cdot \frac{\mathcal{H}_{w'}(p)}{\log \delta} \\ &= \frac{1}{\log \delta} \left(\sum_{j=1}^{m} \frac{w(j)}{d(j)}\right) \cdot \sum_{i=1}^{m} w'(i) \cdot p_i \cdot \log\left(\frac{1}{p_i}\right) \\ &= \frac{1}{\log \delta} \cdot \sum_{i=1}^{m} \frac{w(i)}{d(i)} \cdot p_i \cdot \log\left(\frac{1}{p_i}\right) \end{aligned}$$

This completes the proof of Theorem 11.

■

Theorem 11 implies the following lower bound

$$E[T^A, p] \geq \frac{1}{\log(\delta+1)} \cdot \sum_{u \text{ is a leaf}} \frac{w^A(u)}{d_A(u)} \cdot p_u \cdot \log\left(\frac{1}{p_u}\right) \tag{8.10}$$

Note that this generalizes Shannon's lower bound for noiseless channels. Indeed, if $w(i) = d(i)$ for all i, then the right-hand side of Equation 8.3 is the expected number of steps to reach a random leaf of the tree T from the root, in which case Theorem 11 gives $E_w[T,p] \geq \frac{\mathcal{H}(p)}{\log \delta}$. In general, $w(i) \geq d(i)$ for all i, and hence the quantity in the right-hand side of the equation in Theorem 11 is bounded from below by $\frac{\mathcal{H}(p)}{\log \delta}$. Also,

note that in view of the definition in Equation 8.1, $w(i) \geq (d(i)+1)c$, where c is the minimum size of a file at a node of the tree. Hence, we obtain the following corollary.

Corollary 2. *The expected data transfer generated for accessing a random Web page from the root of the tree T satisfies*

$$E_w[T,p] > \frac{c}{\log \delta} \cdot \mathcal{H}(p),$$

where c is the minimum size of a Web page at a node of the tree.

8.4 Algorithm *weighted-greedyBFS*

One is tempted to pursue a similar idea to that of [35] in order to find a balanced partitioning of the tree. Inevitably this will lead to a recursive algorithm like in [35]. 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 15, operates in the same way as algorithm *greedyBFS,* presented in Section 5.3.2. The difference between these two algorithms is that the former attempts to maximize Equation 2.6, whereas the latter attempts to maximize Equation 2.4. 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 5.1. 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.

Algorithm 15 weighted-greedyBFS(T)

1. $H = \phi$
2. *while(* $($ *s = next_in_BFS_order* $) \neq \phi$ *)*
   (a) $t = v : v$ *maximizes Equation 2.6; 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* $H = H \bigcup \{(s,t)\}$

8.5 Algorithm *weighted-recursive*

This is a recursive version of algorithm *weighted-greedyBFS,* described in Section 8.4, and its operation is similar to algorithm *recursive,* described in Section 5.3.3. Observe that the only difference between algorithm *recursive* and algorithm *weighted-recursive* is that the former attempts to maximize Equation 2.4, whereas the latter attempts to maximize Equation 2.6. The basic principle of algorithm *weighted-recursive,* described in Algorithm 16, is to assign a hotlink (s,t), where s is the home page and t is the node that maximizes Equation 2.6, then split the original tree into subtrees consisting of the subtrees rooted at the children of s and the subtree rooted at t, then proceed recursively for each of these subtrees. Note that one of the children of s, s_k, will be ancestor of t, and the subtree rooted at t will not form part of the subtree rooted at s_k. In the algorithm, we assume that the set of hotlinks, H, is initially empty.

Algorithm 16 weighted-recursive(T)

1. $s = r$
2. *for* $i = 1$ *to* k
   (a) $t = v : v$ *maximizes Equation 2.6; and* v *is not a descendant of a node* x*, which is at a higher level than* s *and already has a hyperparent; and* v *has no hyperparent*
   (b) *if* $t \neq \phi$ *then* $H = H \bigcup \{(s, t)\}$
3. *for each children of* s*,* s_i *(including hypersons)*
   (a) $T =$ *subtree rooted at* s_i
   (b) recursive(T)

Chapter 9

Simulations and Case Study

9.1 Introduction

In Chapter 8 we present the heuristic hotlink assignment algorithm *weighted-greedyBFS*. In this chapter we use simulations to evaluate this algorithm. The simulations are implemented in *Java*. The algorithm is evaluated on three different kinds of structures: random Web sites, real Web sites with an unknown access probability distribution, and an actual Web site with a known access probability distribution.

In Chapter 6 we study the simulation of a random Web site along with the popularities of its pages. In this chapter, we need to assign file sizes (in bytes) to the pages of the random Web sites. Recall that the size of a page is equal to its own size plus the size of its embedded files. We associate a random size to each page and obtain 95% confidence intervals for all the results that we present. We validate the correctness of the simulations by showing that the file sizes of the random Web sites actually resemble the theoretical file sizes distribution.

9.2 Web Sites Used for Simulation

For the simulations, we use the same kind of Web sites discussed in Chapter 6, namely, random Web site trees and real Web sites. To test algorithm *weighted-greedyBFS* we need to enhance those Web sites by associating weights to the Web pages. The weight of a Web page represents the size in bytes of the Web page plus its embedded files, e.g., images, audio, etc. An interesting question arises: What is the distribution of file sizes on the Web?

9.2.1 The Distribution of File Sizes

Arlitt et al. [4] carefully analyse the traffic of a Web server. They prove experimentally that the file size distribution is *heavy tailed* (see [49] for a background of heavy tail models). The assertion of Arlitt et al. was confirmed by Crovella et al. [15], who show experimentally that file sizes greater than about $1,000$ bytes can be well modeled with a Pareto distribution. Barford et al. [6] show experimentally that file sizes can be modeled with a hybrid distribution. Files of "small" size can be well modeled with a lognormal distribution, whereas "big" files can be modeled with Pareto distribution. According to Barford et al., the body of the distribution is modeled with lognormal distribution for values smaller than 133KBytes. Downey [19] creates a model that suggests that file sizes are modeled only by a lognormal distribution. Mitzenmacher [40] 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. [6] is correct, we create a file size distribution as follows:

$$P[w_v = x] = \begin{cases} \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2/2\sigma^2} & \text{if } x < \text{cutoff point} \\ \alpha k^\alpha x^{-(\alpha+1)} & \text{otherwise,} \end{cases} \tag{9.1}$$

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. [5] 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. [5] who observe that 83% of the files fall in the body of the distribution. See Figure 9.3. The values used in Formula 9.1 are displayed in Table 9.1.

parameter	value
μ	7.796
σ	1.625
k	$8,863$
α	1.47
cutoff point	$10,790$

Table 9.1: Parameters used in Formula9.1.

For the case of random Web sites, we assign weights to the pages in a random fashion, according to the distribution of Formula9.1.

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 Formula9.1.

9.3 Results of the Simulations

We simulate the operation of algorithm *weighted-greedyBFS,* discussed in Section 8.4, on random Web sites. The maximum proportion of gain on the access cost is 30%, whereas the minimum is 11%.

9.3.1 Algorithm *weighted-greedyBFS* Evaluated on Random Web Sites

The results of the simulations are plotted in Figure 9.1 and displayed in Table 9.2. 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 ∓ 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.

number of pages in Web site, $\|V\|$	% of gain	standard deviation	95% confidence interval
1000	12.71	6.54	2.27
3000	13.76	5.12	1.78
5000	14.73	6.10	2.11
7000	13.21	4.03	1.40
9000	13.75	5.37	1.86
11000	11.10	3.51	1.22
13000	11.86	4.06	1.41
15000	12.30	3.81	1.32

Table 9.2: 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.

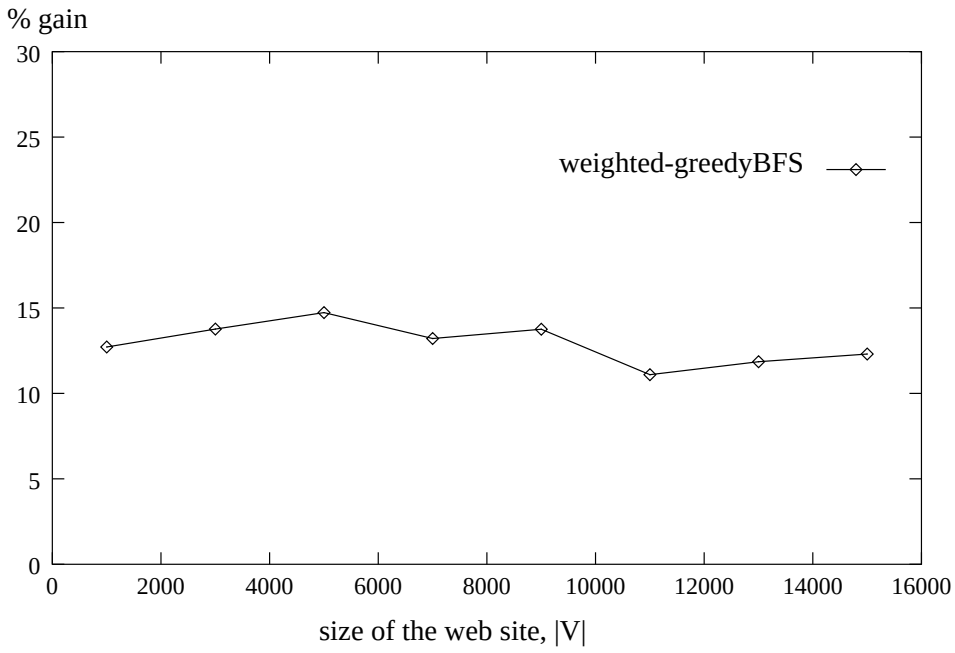

Figure 9.1: Gain obtained by applying *weighted-greedyBFS* to randomly generated Web sites of size $1,000$ to $15,000$. The average proportion of gain has a 95% confidence interval of at most ∓ 2.27.

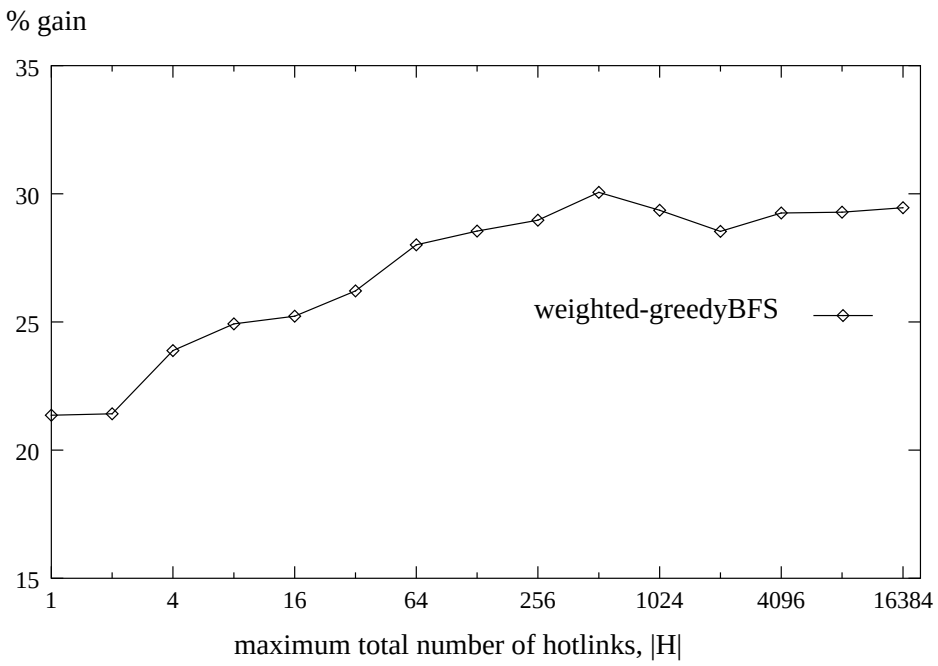

Figure 9.2: Average gain obtained by applying *weighted-greedyBFS* to actual Web sites. The x axis are plotted in logarithmic scale base 2. The average proportion of gain for each of the eleven Web sites has a 95% confidence interval of at most ∓ 7.40. The proportion of gain decreases at some points due to the different assignments of probabilities and file sizes in each experiment.

9.3.2 Algorithm *weighted-greedyBFS* Evaluated on Real Web Sites

We show the results of the simulations in Figure 9.2 and Table 9.3. 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.

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
1024	29.35	12.75
2048	28.53	11.93
4096	29.25	11.88
8192	29.28	13.57
16384	29.45	13.40

Table 9.3: 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 ∓ 7.40.

9.3.3 Validation of Simulations

To validate the correctness of the simulations, we observe the model validation techniques presented by Jain [30], which are summarized in Section 6.5. We evaluate both the correctness of the implementation of the algorithm and the generation of realistic file sizes. The correctness of the structure and popularity of Web sites is validated in Section 6.5.

Correctness of the Implementation of the Algorithm

In Section 6.5 we check that the results of algorithms *greedyBFS* and *recursive* are the same to ensure that the implementation is correct. Here, we perform the same test with algorithms *weighted-greedyBFS* and *weighted-recursive* and confirm that the outcomes are equal. This evidence indicates that the implementation of *weighted-greedyBFS* is correct.

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 9.2.1. The distribution of file sizes used in the simulations is plotted in Figure 9.3.

9.4 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 6.6. 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).

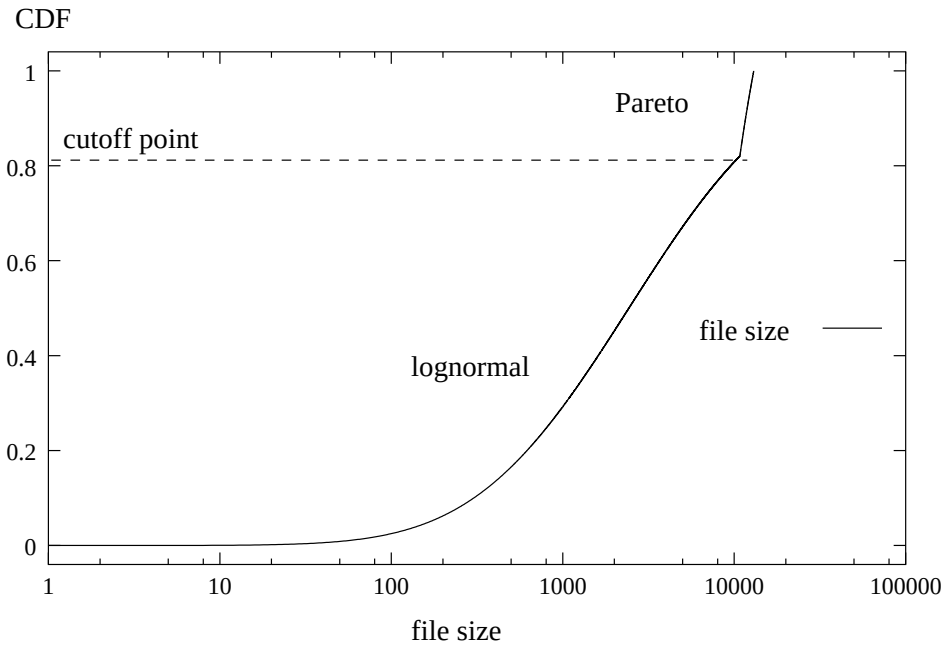

Figure 9.3: The correctness of the file sizes used in our simulations is proven by plotting the cumulative distribution function of the hybrid (lognormal-Pareto) distribution, given in Formula9.1. The file sizes are plotted in log scale. The cutoff point is such that approximately 83% of the file sizes fall in the lognormal distribution.

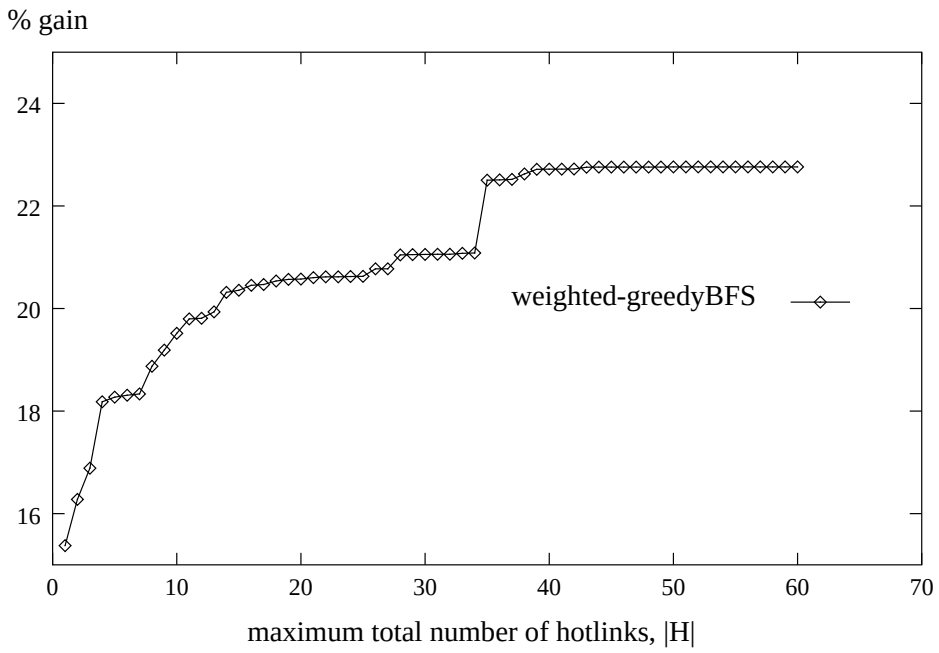

Figure 9.4: Gain attained by algorithm *weighted-greedyBFS* in the *scs.carleton.ca* domain. The domain contains 798 pages and the maximum gain is attained at $|H| = 52$ hotlinks, which indicates that we can get good gain with just a "few" hotlinks.

Figure 9.4 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.

Chapter 10

Conclusions and Extensions

In this dissertation 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 started by formulating the corresponding optimization problems, proving NP-hard results, and then explaining what our approach would be. We presented the mathematical background for the study of hotlink assignments and provided hotlink assignment algorithms for special cases of Web site structures. We also introduced hotlink assignment algorithms for arbitrary Web sites. These algorithms were tested using simulations on random and real Web sites. After that, we developed a software tool that implements our most efficient hotlink assignment algorithm and provides a friendly user interface.

In Part I we presented the necessary background for the study of hotlinks. In Chapter 1, we provided an overview of our work by discussing the problems and their solutions in an intuitive manner. In Chapter 2 we formally defined the problems, proved NP-hard results and discussed our approach.

In Part II we presented our work for full binary trees. In Chapter 3, we studied the assignment of hotlinks and bookmarks in full binary trees. We presented a lower

bound derived from Shannon's theory. We introduced hotlink assignment algorithms for special cases of access probability distributions, namely, geometric, arbitrary, uniform, and Zipf's. In particular, we presented a hotlink assignment algorithm which is optimal for uniform distributions. It would be interesting to find better algorithms for other distributions and to tighten the upper and lower bounds. In Chapter 4 we studied the assignment of bookmarks, which are a special case of hotlinks, to full binary trees. We introduced an optimal bookmark assignment algorithm for uniform access probability distributions. It would be of interest to study the assignment of bookmarks with other access probability distributions.

In Part III we studied the optimization of the expected number of steps to reach the pages of a Web site. In Chapter 5 we presented a theoretical lower bound on the minimum number of steps required to reach a page of a Web site, and presented heuristic hotlink assignment 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. In Chapter 6 we evaluated the performance of the hotlink assignment algorithms presented in Chapter 5. We found that the expected number of steps in a Web site can be reduced by at least 24% and as much as 35%. 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 real reduction on the number of steps. In Chapter 7 we introduced the Hotlink Optimizer (HotOpt), a software tool that assists Web administrators and designers on restructuring Web sites by assigning hotlinks, according to user access patterns. 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.

Finally, in Part IV we studied the optimization of the expected number of bytes

that must be transferred by the Web server when a user visits one of its pages. In Chapter 8, we presented a theoretical lower bound and a heuristic hotlink assignment algorithm. We believe that better algorithms can be designed. In Chapter 9 we evaluated the hotlink assignment algorithm presented in Chapter 8. We found that the data transfer can be reduced by at least 11% and as much as 30%. Again, it would be interesting to see to what extent users actually use the hotlinks and to determine the real reduction in data transfer.

Appendix A

Glossary

Access cost of a Web page Equals the access probability of the page times the cost of the shortest path from the home page.

Access cost of a Web site Equals the access costs of all the Web pages of the Web site.

Access patterns The manner in which users navigate the Internet.

Access weight of a Web page Equal the access probability of the page times its size in bytes.

Data transfer Number of bytes that need to be transferred when accessing a Web page.

Descendant Web page In a tree structure, page v is descendant of page u if there is a path of hyperlinks from u to v.

Distance Number of hyperlinks in the shortest path from one page to another.

Expected data transfer Average number of bytes that need to be transferred to access a random page of a Web site.

Expected number of steps Average number of hyperlinks that need to be taken to access a random page of a Web site.

Gain of a hotlink assignment Savings in the access cost obtained by placing hotlinks in a Web site.

Home page Starting page of a Web site. It is assumed that every other page of the Web site can be reached from the home page.

Hotlink Shortcut in the shortest path between the home page and another page of a Web site.

Hyperlink Link between two Web pages that is not a hotlink.

Hyperparent The hyperparent of page v is the page containing a hotlink that goes directly to v.

Hyperson A hyperson of page u is the end point of one of its hotlinks.

Indegree Number of hyperlinks pointing to a particular page.

Latency of a Web page Time elapsed from the moment when the user requests a page on his or her navigator to the moment when the page is completely shown on the screen.

Latency of a Web site Average sum of the latencies of the pages of the Web site.

Outdegree of a Web page Number of hyperlinks in a page.

Web caching Consist on storing Web documents in the local memory for future reference.

Web diameter Average distance between two random Web pages of the Internet.

Web mirroring Consists on keeping one or more copies of a Web site (or part of it) in different strategic locations.

Web page Block of information put together in a document and made accessible through the Internet. This document usually has hyperlinks to other Web pages, and may contain embedded files, e.g., text, audio, video, and images.

Web site Collection of Web pages administered by the same authority which are linked together to form a unified source of information.

Weight of a Web page Size in bytes of the page plus its embedded files.

Appendix B

List of Symbols

symbol	*meaning*
δ	maximum outdegree of the pages of a Web site
δ_v	outdegree of page v
$\Pi(r, u)$	a minimum shortest path between the home page r and page u
$\Pi^H(r, u)$	minimum shortest path between the home page r and page u given the hotlink assignment H
A_i	*i-th* partition of the leaf pages
b	bookmark
B	set of bookmarks
c	minimum size of a page in a Web site
$c^{(i)}$	minimum size of the pages that belong to the tree induced by the *i-th* partition of leaf pages
$c^{A,(i)}$	given the hotlink assignment A, minimum size of the pages of the subtree of T^A induced by the *i-th* partition of the leaf pages

$c(v)$	access cost of page v
$d(v)$	minimum distance between the home page and page v
$E[T]$	expected number of steps of a Web site T (with implicit access probability p)
$E_w[T,p]$	expected data transfer of a Web site T with access probability p
$g(h)$	gain attained with hotlink h
$g(b)$	gain attained with bookmark b
$g_v(B)$	gain attained for a page v in the tree T^B
$\mathcal{G}(B)$	gain attained by the set of bookmarks B
$\mathcal{G}(H)$	gain attained by the set of hotlinks H
$\mathcal{G}_{-X}(B)$	gain attained by the set of bookmarks B in a tree diminished by the pages of the set X and all its descendants
h	hotlink
H	set of hotlinks
H_m	harmonic number with respect to m
$\mathcal{H}(p)$	entropy of the probability distribution p
$\mathcal{H}_w(p)$	entropy of the probability distribution p in a weighted Web site
k	usually, maximum allowed number of hotlinks leaving a Web page
K	total number of hotlinks in a Web site
l	usually denotes a level of the tree
$\log x$	logarithm base 2 of x
m	number of leaf pages in a Web site
n	number of levels in a full tree
N	number of pages in a Web site
p	probability distribution

$p^{(i)}$	probability distribution computed for the leaves of the *i-th* partition
p_v	access probability of node v
r	home page of a Web site
s	usually, the starting point of a hotlink (s, t)
S_i	sum of the probability of each leaf page of the *i-th* partition
t	ending point of a hotlink (s, t)
$T = (V, E)$	a tree T with the set of nodes V and the set of edges E
T^B	tree T augmented with the set of bookmarks B
T^H	tree T augmented with the set of hotlinks H
w_v	size in bytes of page v and its embedded files
$w(v)$	access weight of page v
$w^A(v)$	access weight of page v given the hotlink assignment A

Bibliography

[1] Norman Abramson. *Information Theory and Coding.* McGraw Hill, 1963.

[2] Lada A. Adamic. The small world Web. In S. Abiteboul and A.M. Vercoustre, editors, *In Proceedings of the Third European Conference on Research and Advanced Technology for Digital Libraries, ECDL*, volume 1696, pages 443–452. Springer Verlag, 1999.

[3] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 171–180, Portland, Oregon, U.S.A., May 21–23 2002. ACM.

[4] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[5] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web*, 2(1–2):15–28, 1999.

[6] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Measurement and Modeling of Computer Systems*, pages 151–160, 1998.

[7] Michael Bauer and Ruya Tang. On the applicability of predictive caching to improve Web server performance. In *Proceedings of the International Conference on Internet Computing (IC '02)*, pages 111–116, Las Vegas, Nevada, U.S.A., June 24–27 2002. CSREA Press.

[8] Prosenjit Bose, Jurek Czyzowicz, Leszek Gąsieniec, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel Vargas Martin. Strategies for hotlink assignments. In *Proceedings of the Eleventh Annual International Symposium on Algorithms and Computation (ISAAC 2000)*, pages 23–34, Taipei, Taiwan, December 2000.

[9] Christos Bouras and Agisilaos Konidaris. Estimating and eliminating redundant data transfers over the Web: A fragment based approach. In *Proceedings of the International Conference on Internet Computing (IC '02)*, pages 95–102, Las Vegas, Nevada, U.S.A., June 24–27 2002. CSREA Press.

[10] Tim Bray. Measuring the Web. In *Proceedings of the Fifth International WWW Conference*, Paris, France, May 1996.

[11] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the Web. *Computer Networks*, 33(1–6):309–320, June 2000.

[12] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, June 1997.

[13] Lara D. Catledge and James E. Pitkow. Characterizing browsing Strategies in the World-Wide Web. In *Proceedings of the Third International World-Wide Web Conference*, Darmstadt, Germany, April 1995.

[14] Ming-Syan Chen, Jong Soo Park, and Philip S. Yu. Data mining for path traversal patterns in a Web environment. In *Sixteenth International Conference on Distributed Computing Systems*, pages 385–392, 1996.

[15] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic. Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[16] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel Vargas Martin. Evaluation of hotlink assignment heuristics for improving Web access. In *Proceedings of the International Conference on Internet Computing (IC 2001)*, volume 2, pages 793–799, Las Vegas, Nevada, U.S.A., June 25–28 2001. CSREA Press.

[17] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Miguel Vargas Martin. Optimal assignment of bookmarks to Web pages. In progress, 2002.

[18] Brian D. Davison. Web caching and content delivery resources. http://www.web-caching.com, 2002.

[19] Allen B. Downey. The structural cause of file size distributions. In *Proceedings of the Nineth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Cincinnati, Ohio, U.S.A., August 15–18 2001.

[20] M. Carl Drott. Using Web server logs to improve site design. In *ACM Sixteenth Annual International Conference of Computer Documentation (SIGDOC-98)*, Getting feedback on your Web site, pages 43–50, New York, New York, U.S.A., September 23–26 1998. ACM Press.

[21] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. *Computer Communication Review*, 29(4):251–262, 1999.

[22] Yongjian Fu, Mario Creado, and Ming-Yi Shih. Adaptive Web sites by Web usage mining. In *Proceedings of the International Conference on Internet Computing (IC 2001)*, volume 1, pages 28–34, Las Vegas, Nevada, U.S.A., June 25–28 2001. CSREA Press.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.

[24] Steven Glassman. A caching relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27(2):165–173, 1994.

[25] Apache Group. Apache HTTP server project. http://httpd.apache.org/docs/logs.html, 2002.

[26] Network Working Group. Request for comments: 2068. Hypertext transfer protocol HTTP/1.1. http://www.ietf.org/rfc/rfc2068.txt, 1997.

[27] Brian Hayes. Graph theory in practice: Part I. *American Scientist*, 88(1):9–13, January–February 2000.

[28] Brian Hayes. Graph theory in practice: Part II. *American Scientist*, 88(2):104–109, March–April 2000.

[29] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. On near-uniform URL sampling. In *Proceedings of the Nineth International World Wide Web Conference*, Amsterdam, The Netherlands, May 15–19 2000.

[30] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[31] Jaeyeon Jung, Dongman Lee, and Kilnam Chon. Proactive Web caching with cumulative prefetching for large multimedia data. In *Proceedings of the Nineth International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000.

[32] Donald Ervin Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 1973.

[33] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

[34] Evangelos Kranakis, Danny Krizanc, and Miguel Vargas Martin. The hotlink optimizer. In *Proceedings of the International Conference on Internet Computing (IC '02)*, volume 2, pages 87–94, Las Vegas, Nevada, U.S.A., June 24–27 2002. CSREA Press.

[35] Evangelos Kranakis, Danny Krizanc, and Sunil Shende. Approximate hotlink assignment. In *Proceedings of the Twelfth Annual International Symposium on Algorithms and Computation (ISAAC 2001)*, volume 2223, pages 756–767, Christchurch, New Zealand, December 19–21 2001. SVLNCS.

[36] Mark Levene, José Borges, and George Loizou. Zipf's law for Web surfers. *Knowledge and Information Systems*, 3(1):120–129, 2001.

[37] Bo Li, Xin Deng, Mordecai Golin, and Kazem Sohraby. On the optimal placement of Web proxies in the Internet: the linear topology. In *Proceedings of the Eighth IFIP Conference on High Performance Networking, HPN'98*, Vienna, Austria, September 1998.

[38] Bo Li, Mordecai Golin, Giuseppe F. Italiano, Xin Deng, and Kazem Sohraby. On the optimal placement of Web proxies in the Internet. In *Proceedings of the IEEE InfoCom Conference*, March 1999.

[39] Ari Luotonen and Kevin Altis. World-Wide Web proxies. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, April 1994.

[40] Michael Mitzenmacher. Dynamic models for file sizes and double pareto distributions. http://citeseer.nj.nec.com/mitzenmacher02dynamic.html, 2002. Preprint.

[41] Tekehiro Nakayama, Hiroki Kato, and Yohei Yamane. Discovering the gap between web site designers' expectations and users' behavior. In *Proceedings of the Nineth International World Wide Web Conference*, Amsterdam, The Netherlands, May 15–19 2000.

[42] Mike Perkowitz and Oren Etzioni. Adaptive sites: An AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97*, volume 1, pages 16–23, Nagoya, Japan, August 23–29 1997.

[43] Mike Perkowitz and Oren Etzioni. Towards adaptive Web sites: Conceptual framework and case study. *Artificial Inteligence*, 118(1–2):245–275, 2000.

[44] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach.* Morgan Kaufmann Publishers, Inc., 1996.

[45] Peter Pirolli, James Pitkow, and Ramana Rao. 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*, volume 1 of *PAPERS: World Wide Web*, pages 118–125, 1996.

[46] James E. Pitkow. Summary of WWW characterizations. *Computer Networks and ISDN Systems*, 30(1–7):551–558, 1998.

[47] James E. Pitkow and Margaret M. Recker. A simple yet robust caching algorithm based on dynamic access patterns. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, Atlanta, GA, 1994.

[48] Albert Réka, Jeong Hawoong, and Albert-László Barabási. Diameter of the World-Wide Web. *Nature*, 401:130–131, 1999.

[49] Sidney I. Resnick. Heavy tail modeling and teletraffic data. *Annals of Statistics*, 25(5):1805–1869, 1997.

[50] Beth Saulnier. Small world. *Cornell Magazine On Line*, 101(1), 1998. http://cornell-magazine.cornell.edu/Archive/JulyAugust98/JulyWorld.html.

[51] Myra Spiliopoulou, Lukas C. Faulstich, and Karsten Winkler. Data mining for the Web. In *Principles of Data Mining and Knowledge Discovery*, pages 588–589, 1999.

[52] Ramakrishnan Srikant and Yinghui Yang. Mining Web logs to improve Website organization. In *Proceedings of the Tenth International World Wide Web Conference*, pages 430–437, Hong Kong, May 2001.

[53] Duncan J. Watts. *Small Worlds*. Princeton University Press, 1999.

[54] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[55] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf. Segment-based proxy caching of multimedia streams. In *Proceedings of the Tenth International World Wide Web Conference*, Hong Kong, May 2001.

[56] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *INFOCOM 1996*, volume 2, pages 594–602, San Francisco, California, U.S.A., March 24–28 1996. IEEE.

[57] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, December 1997.

[58] George Kingsley Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, Cambridge, Massachusetts, U.S.A, 1949.